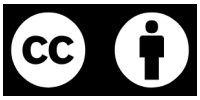
OPEN AND SECURE

Mastering 'Threat Modeling' in Assessing the Security of Open Source Projects

CLOUD NATIVE COMPUTING FOUNDATION

Open and Secure — Mastering 'Threat Modeling' for Security Assessments of Open Source Projects

by Justin Cappos with feedback and support of the CNCF Security Technical Advisory Group.



First Edition 2023

ACKNOWLEDGEMENTS

This document would not be possible without the effort from a number of other parties who helped this work tremendously. Among those who contributed to the text directly, Justin Cappos would like to single out Ragashree Shekar for her help with commentary, editing and diagrams. Andres Vega provided commentary and logistical support which was essential to this book's publication.

Other members of the TAG Security community made valuable contributions to this work in the form of commentary and/or edits. This includes Andrew Martin, Ash Narkar, Dan Lorenc, Jack Kelly, and Marco De Benedictis.

We also appreciate the feedback from our reviewers, including AKS and all other members of the TAG-Security community who provided feedback.

TABLE OF CONTENTS

INTRODUCTION

CONTEXT

Security is perhaps the most misunderstood field of computing. In part this is because many properties of it are hard to quantify. One of the key reasons is the existence of an intelligent adversary, which is fundamental to computer security.

To understand a bit why, suppose you had never played the game of chess before, but you knew how the pieces move. Now suppose that you were playing against a computer that would randomly choose and make a legal move with one of its pieces. You could pretty quickly come up with simple strategies that would make you overwhelmingly likely to be a winning chess player. You could even write this into a program which would win overwhelmingly often. If you were mathematically inclined, you could rank how well different algorithms do by their expected win percentages against this random adversary. This is conceptually similar to how one can think about reliability; failure probabilities in a random case.

Now, consider that you instead play against human opponents who can intelligently choose which pieces to move and where. All of the sudden all of your win percentages and strategies are very unlikely to be relevant. Even worse, suppose that those human opponents may have been studying your way of playing and devising strategies to counter your program!

There may even be hundreds of players all trying to defeat you at once. It's clear that this situation is a much more worrisome and dangerous one for your poor chess program. It is hard to quantify or reason about its win rate in this case because you don't know what the human opponent will do to counter you.

One interesting thing to consider about this example is that technically any moves that the human player makes against you, could also have been made by the random computer player. What matters here is that a human will tend to act to make events that would be very unlikely to happen randomly, occur in just the ways that cause the most harm.

Being able to reason about which defensive mechanisms will restrict an intelligent attacker and what avenues of attack are likely to be effective is at the heart of understanding computer security.

How to use the rest of this book

So, armed with some context, your next question may be, why am I reading this book and what will I learn? This book is meant for several different audiences.

- *A reader interested in learning about threat modeling and security in general.* To learn about threat modeling and how to assess the security of general projects, the majority of the sections are highly relevant. Perhaps the least relevant part are the portions of this book that relate to the specifics of TAG-Security Security Assessments (TSSA). However, these sections can serve as an example of how to implement some of the ideas in the rest of the book in the cloud native space.

- *A reader who wants to perform an assessment for TAG-Security.* To do so, you should read most of the book. The sections describing how to use an assessment and how to have your project assessed effectively are less helpful, but are still useful to read to understand the process from those perspectives.

- *A project maintainer who is preparing to have their software assessed.* Please do a quick read of the main portion of the book before diving into how to work with us on your self assessment and the remainder of the process.

- *Someone evaluating the security posture of a project with a security assessment.* You, the consumer of our hard work, need to understand how best to benefit from a security assessment. If you are looking at a TSSA, the section on consuming TSSA assessments is exactly what you need. If you have a more general security assessment document, most of the lessons there still apply. It may also be useful to read the following section on Security Assessments and Audits, to understand the difference and why you should expect to see relatively few CVEs raised after a security assessment versus a security audit.

ASSESSMENTS

VS

AUDITS

Security Assessments vs Security Audits

When looking at security, there are different levels at which you can do this. Roughly speaking, a security assessment can be thought of as examining the security architecture and posture of a software project. While the tooling, implementation strategy, deployment, etc. are important to a security assessment, the assessment is often detached from a specific deployment and the implementation itself. It focuses more on whether a software project as a whole is doing the sorts of things that lead to security.

Let's now consider an example security assessment using a real world example of a bank called TrashPanda Bank. TrashPanda Bank is a brick-and-mortar bank without any computers, which allows one to focus on non-technical attacks and defenses. A security assessment would look at a TrashPanda's security by examining the blueprints, vault types, alarm systems, accounting practices, policies for vetting employees, etc.

In contrast a security audit looks for instances of specific flaws in a security project. So, for a computing project, the goal of an audit is to find specific attack cases / issues in the source code that could enable an attacker to do something malicious.

To return to TrashPanda Bank, a security audit of this would look quite different from an assessment. The auditor might try to pick locks, actually smash a chisel into the mortar around a safe to see if it can be removed, understand if the gym using the floor above the bank could enable one to smash in through the vault's ceiling, or figure out the timing of the security guard bathroom breaks to see if there is a moment they can sneak past undetected. In other words, these look for problems that result in specific, detailed flaws in implementation and quirks of the deployment that cause weaknesses an attacker can exploit.

One final note is that the terms "audit" and "assessment" are not universally used this way in all literature. So if you read them elsewhere, please consult the author's definition.

Pros and Cons

There are merits to both audits and assessments. As a result, the best security firms will do both sorts of analysis (to different levels of detail) on a software project.

Assessments tend to be better at identifying more systemic problems, like design problems or issues in the procedures used to make software. These problems are extremely important to fix because they are often the cause of security issues. They can also increase the impact, turning a minor problem into a major one.

Assessments are often essential for an organization to tell if a software project is a good one to rely on.

The way in which things are fixed and the "quality" of the software project are exhibited as part of the assessment.

Assessments can be valid for a long period of time (years) so long as the project does not make substantial changes to how they make software. As such an assessment better represents the project's overall health. In contrast, a security audit will instead represent only a momentary snapshot of a project's set of vulnerabilities (often only a single release) and only for the deployment scenarios considered.

Similarly, an assessment is general enough that some properties (like the actors, actions, goals, etc.) will directly translate over to multiple implementations of the system in different languages and also will likely translate to a wide array of deployment environments. With an audit, bugs found in an audit are often specific to an implementation (unless the implementers looked at each other's code and copied them!).

Security audits are great for finding bugs in the project today. Hence a project can often fix problems found in an audit rather quickly. It is usually immediately apparent that these bugs were impactful because an auditor often provides an example exploit that causes a security failure due to the bug. This looks great to management as it is clear what value they have derived from the security audit and patching the bug.

To understand the difference between assessments and audits consider the following cases for TrashPanda Bank. The security assessment firm says that you should implement better personnel controls which the assessors claim will improve several different security aspects. In contrast, a security auditor may find out that Eve in accounting has been embezzling money, which then may lead the firm to fire and prosecute her.

On the surface, the audit sounds more pertinent at any particular moment because it has an actual example of a serious problem. The downside stems from the fact that an audit focuses on what someone found at that moment, it is even the case that different audits may lead to quite different results. For example, security firm A's audit may have caught Eve's embezzling, while security firm B's audit finds out that Tom the teller has a gambling problem and has been skimming deposits (i.e., stealing cash when a deposit is made). The two firms who did different audits found different problems, which is expected. With audits, you really don't know of any underlying deficiencies other than the bugs they found. In contrast, with a security assessment, you tend to focus on macro-level concerns and procedures at TrashPanda Bank. You may tighten up your personnel controls, which may lead to Eve silently stopping her behavior as she knows she would be caught and Tom the teller taking a job at another bank. So, although acting upon the results of an assessment may mitigate or prevent these issues from arising, you may never detect occurrences of a problem explicitly from an assessment, only that there is a potential problem, or if you do, you may not even associate them with the security assessment. This makes the value of a security assessment require more effort to quantify, such as factoring reduction of structural risk and mitigation of losses by reducing the likelihood and severity of a negative outcome should problems occur.

BASICS

SECURITY BASICS

There are so many foundational concepts and technologies you need to understand to reason about security of a cloud native application, that describing them well would require another entire book's worth of material. Rather than replicate that material here, the reader is directed to resources that contain this information. If you encounter an unfamiliar term in the text, kindly take the time to look it up and understand it. Most fundamentally, you should understand key concepts like integrity, non-repudiation, privacy, authentication, authorization, and trust. The Cloud Native Security Lexicon (available at tag-security/security-lexicon at main · cncf/tag-security (github.com)) has a quick overview of basic terms and concepts in computer security which covers these items.
For encryption, there are a lot of concepts you need to understand and cryptographic systems are very complex. Fortunately, you really just need to understand how to use them correctly and their strengths and weaknesses, instead of why they were designed in the way that they were. You will need to understand (at a minimum) public key cryptography, secret key cryptography, secure hash functions, key length, key distribution, root of trust, certificate formats (i.e., X.509), and certificate authorities. Depending on what you are assessing, understanding trust delegation, HMAC, post quantum cryptography, transparency logs, forward secrecy, and similar concepts may be useful.

For computational security on a system, you need a basic understanding of access control. This means understanding compartmentalization / isolation as it relates to the operating system or container environment you are using. Access Control

Justin Cappos's commentary: Beware of systems making broad promises due to the use of blockchain, Web 3.0, or decentralization. To date, the proponents of these systems have claimed far greater benefits than what the core technology has been able to deliver.

For example, a proof-of-work blockchain is fundamentally a way to keep a distributed, append-only log amongst a set of distributed computers that don't want to have a trusted centralized party. It is extremely slow and computationally wasteful compared to a centralized trusted server, but there is no longer a single point of compromise. That is, if you assume that the computational nodes have a protocol that provides this property, that the protocol is implemented correctly, and that some threshold (commonly 1/3 or 1/2) of the computational power isn't held by evil people, etc. It also, by itself, doesn't ensure that the information in the blockchain is actually valid or useful.

Interestingly enough, a transparency log uses a lot of the same mechanisms as a blockchain and thus has some similar weaknesses. However, transparency logs currently don't have the same stigma in the security community in part because the deployment environment and stakes are different. There are large deployments of transparency logs today but they are early enough in their lifecycle that as a community, we really don't fully understand how and when these systems fail to provide adequate security in the same way we do the other technologies in this section.

Lists (ACL) systems, file / device permissions, su (superuser) ability, system call filtering (seccomp), and capability / tokens are all very important to understand conceptually. Depending on your environment, knowledge of HSMs (Hardware Security Modules) and TPMs (Trusted Platform Modules) may also be relevant.

There is an additional set of things to understand around user identity, authentication, and authorization. This involves concepts like multi-factor authentication (also called two-factor authentication), hardware tokens (e.g., Yubikeys), and OIDC (a way to log in to a system via authentication through a third party like Google or Facebook). It is key to understand how users are identified and how this is tied to logging events for auditing purposes.

The last important concept to understand is the fundamental ways in which people design secure systems. Usually, you can find security design flaws by looking for situations that violate these principles and then reasoning about what problem occurs as a result. So understanding concepts like the principle of simplicity, least privilege, fail-safe defaults, least common mechanism, minimizing secrets, open design, complete mediation, and least astonishment [Saltzer and Schroeder, The Protection of Information in Computer Systems] are really fundamental and things every person thinking about security should internalize.

Justin Cappos's commentary: Note that these principles are not fundamental "laws" of computer security that should never be violated. They are guidelines that point to situations which most of the time lead to security problems if they are violated.

For example, the principle of simplicity indicates that the simpler the component, the easier it is to reason about it and thus secure it. Suppose that TrashPanda bank's system designer learns of this and decides to remove the need to verify client ID cards to simplify the system. Now anyone can withdraw money from anyone else's account, trivially! This "simplification" has clearly made the system's security worse.

So, instead think about the principles when looking at a design and reason if the security would be better or worse if they were followed. Usually, following the design principles will guide you toward security.


THREAT
MODELING

THREAT MODELING

Security is one of the most critical properties to have in computing today. Unfortunately, it is also one of the most misunderstood. A common mistake people make it to tout something as "secure" or "insecure". This doesn't make a lot of sense because it is missing an important context: the scenario.

The scenario in many non-security real world situations is something that is implicitly defined. For example, if I say "my car is reliable", you can assume that it almost certainly will not break down on the way to work. However, you should not expect that a "reliable" car would make a good submarine or perform well on Mars. Performing well on Mars is just not what is implied by a general statement of a car's reliability.

While usually, one could just look at likely scenarios and determine the rarity of events, there is another aspect of security which makes this not work well: the intelligent adversary. In security, one assumes that an adversary has some ability of control over the system or environment and crucially, that an intelligent adversary will choose to set things up in a way that favors them. So, you may have set up the communication properties on your network to detect or correct 99.9999% of errors in random noise. But unless some secret prevents the attacker from knowing how your error correction works, the attacker can generate network traffic that makes your error correction useless.

One way that we reason about security in a rigorous way is a process called threat modeling. Threat modeling is sort of like setting up a game between the defender and the attacker. The threat model describes the properties you are trying to provide and the capabilities of the attacker. If the attacker is able to find a way to defeat the defender's desired security properties, this is a viable avenue of attack. We call such a successful attack, a compromise, and the weakness that lets an attack occur, a vulnerability.

Note that the term bug and vulnerability are not the same thing. While many bugs do enable an attacker to launch a successful attack, many bugs are just anomalous, benign behavior. Similarly, a design flaw can cause a correctly implemented system to have a vulnerability. So, there need not be a bug in order to have a vulnerability.

ACTORS

We need a term to describe the parties in the system that perform all of the actions in the system and might be erroneous, compromised, or just plain malicious. We call these actors and the things they do actions. It is important to enumerate these up front as they are effectively the "players" in the threat modeling game.

Back in earlier days of computing, many computer systems were fairly isolated from each other and programs needed to be secure in this environment. Hence the number of actors was small, often just a server, a client, and an attacker. In modern systems that consist of many distributed and isolated components, the number of actors can be very large.

To see how large modern systems can get, consider an assessment for the Sigstore project and the way it might get integrated into an open, community software repository like PyPI, a community repository of software for the Python programming language. The actors include the PyPI server, the administrators of PyPI, the CA(s) trusted to issue PyPI's public key, parties that control BGP and/or routers, parties that control DNS, the developers who use PyPI for their software, the CDN that distributes PyPI software, the users downloading that software, and outsiders. If you think it seems overwhelming, consider also that at this point we haven't even listed the parties for Sigstore, which would be another 10 or so actors!

Is It Good Or Bad To Have Many Actors?

You may think that having more actors automatically makes a system have better or worse security properties. There are two factors that lead to having many actors and they impact the security of a system in opposing ways.

The first factor is the security principle that complexity tends to lead to insecurity. Simply put, if an attacker can bypass your system by finding a flaw, the more places the attacker can look, the easier it tends to be. Of course, this doesn't mean you should remove encryption code or security checks because they make the code longer! It just means that all other things being equal, more complexity (i.e. actors) tends to lead to more bugs.

The second factor includes the principle of least privilege, that a party should have as little privilege as possible, which is the main argument for compartmentalization. Compartmentalization means that when one portion of a system fails or is compromised, it is separated, much like a ship might have protections so if one part of the hull is breached and the internal part is flooded, it doesn't automatically spread to the entire ship. Compartmentalization helps to contain the attackers capabilities from a single compromise. Consider instead a system with a single point of failure; this has fewer actors, but is clearly weaker from a security standpoint.

Compartmentalization of Actors

A key aspect to consider is the mechanism by which actors are compartmentalized (i.e., isolated) from each other in a system. After all, if the private keys for Alice and Bob are stored on a file system that both have access to, then if either Alice or Bob is malicious, they steal the other one's key and then can do anything the other's private key is trusted to do as well. So, it is worth discussing why, how, and when actors are separated from each other.

Note that this also may depend on the deployment environment. Perhaps some deployments share storage for Alice and Bob for cost reasons. This is important to highlight, as it will become relevant later when we think about the impact of attacks.

One more note is that having different levels of compartmentalization between actors is common in a system. Most systems have a trusted actor who is responsible for indicating who the other actors are in the system. (This often a party like a CA, root of trust, root key, or similar.) As a result, this trusted actor can effectively issue false credentials and pretend to be any other party. In contrast, the other actors in the system may have strong isolation between them, making the act of compromising them effectively independent attacks that must be carried out. This degree to which the isolation between parties contains a compromise can be a critical aspect of the system's security.

ACTIONS

In addition to understanding the actors, it is important to know what actions they perform. This should include the actions that are desirable (at a high level) and how they are carried out, including any checks and balances.

For example, in TrashPanda Bank, customers may have a list of actions they perform such as opening an account, withdrawing money, checking a balance, renting a safety deposit box, visiting their safety deposit box, and making a deposit. For each of these actions, there needs to be a detailed description of how the process works and how the various steps are verified by different parties.

An example action may look something like the following:

Renting a safety deposit box:
Requires a customer with a current account to make an in-person request at TrashPanda Bank to a teller. The teller processing the request first verifies the customer's account exists, is linked to the customer (by checking their identification) and has a balance of at least $100.
The teller then gives the terms and conditions form to the customer, who signs the request. After this is confirmed by the teller, the customer pays the deposit fee to the teller. The teller logs this transaction into their log book and inserts the payment as per the steps in "making a deposit", except that the remittance goes to TrashPanda's safety deposit box fund (listed in the teller's handbook) instead of the user's account.

The manager is then called by the teller, who re-checks the client's identification and verifies the remittance to TrashPanda's safety deposit box was processed by checking the logbook of the teller. The manager now accesses the safety deposit usage map to find an unused safety deposit box, considering customer requests for a specific lucky number or an accessible box. The manager then provides the customer a copy of the key for the box. The teller and the manager use their keys to provide the customer access to the vault, where the safety deposit boxes are kept. The manager and teller leave the vault to provide the customer privacy. Once the customer is finished, they exit the vault and the manager locks the vault again.

Note that increased complexity of actions does tend to correlate with insecurity, at least if you ignore the complexity added by security steps. A system which does a few simple things is easier to secure in most cases.

Please don't mistake this for saying that fewer API calls or system calls means better security. If that were true, we could just have one API call that takes an argument telling it what action to actually perform! This would be a case where the complexity of the API isn't well reflected by the number of API calls.

Goals and Non-Goals

System Goals

One of the most important things to do in threat modeling is to understand what an attacker can and cannot do based upon the access they have. In our concept of a "game" this is like the conditions by which the attacker gains points (by violating the goals you have for your system) and the legal moves that the attacker can make toward that end.

Assuming that you are being realistic in your attacker model, the stronger the set of moves the attacker can make, the more secure your system is. To understand why, let's say that TrashPanda Bank made the assumption that all of its employees were trustworthy and did their job flawlessly. If it turns out that one of the employees is malicious or makes a mistake, then you are now outside the bounds of what you have considered in your assessment. It is as though a player of the game you set up made a move that you thought was not legal, when you did your analysis! This means you don't have a way of understanding what the impact of an attack would be or whether your security will hold.

To make it simpler, there are a set of standard assumptions that most systems make in the current era (early 2023). A note for any future reader, these assumptions tend to evolve over time and so may not be reasonable while you are reading this document.

Compartmentalization of Actors

- *Common assumption:* The government, company management, or a similar agency will not compel the organization to perform actions that violate the security goals of the system. While this may seem a fanciful attack to some readers, this is a legitimate attack risk that many companies have faced and in fact do (often silently) face today. For a real world example, consider the pressure on Apple to create a malicious update and unlock the San Bernardino shooter's phone [Wikipedia: Apple–FBI encryption dispute] However, most security systems are designed so they will fail in such a case and allow the government, company leadership, or a sufficiently large set of malicious insiders to violate its security goals

- *Common assumption:* Cryptographic algorithms that are widely thought to be secure, are secure. This includes public/private cryptography, symmetric key algorithms, cryptographically secure hash algorithms, etc. In practice, contests like the ones that NIST holds to choose cryptographic algorithms tend to have produced excellent results. Even when algorithms fail, it tends to be a slow breaking of the algorithm. The breaking of the algorithm is often possible first by parties with a large quantity of computational resources instead of a sudden moment where anyone can trivially break the algorithm. Other standards bodies have a much more mixed record, in particular if their security systems are effectively designed by committee. Look carefully for broad peer review of cryptographic algorithms and security designs, as NIST performs, as an indicator of quality.

- *Common assumption:* Hardware memory protection mechanisms work as designed. After SPECTRE and MELTDOWN, people in the community realized that there are some ways to use rare cache / memory error behaviors to bypass security protections. For example, a program could read memory in the operating system kernel or in another program. A series of defensive code changes now makes these attacks infeasible on modern hardware (as we understand it). The assumption that memory protections work is common not because it is universally thought that memory protection will absolutely hold in all cases, but largely because not having this assumption makes it too challenging to design security systems. It essentially makes it infeasible to do compartmentalization on a single piece of computing hardware and may make it feasible to cause information disclosure from any component on the same physical hardware. As this is currently an area of active research by hardware security researchers and chip makers, the protections and our understanding of the risks in this domain are likely to evolve over time.

Justin Cappos's commentary: Note that today, these assumptions are being relaxed by some modern security systems like TUF. For example, TUF supports multiple cryptographic algorithms and has a built-in way to add and remove cryptographic algorithm support while maintaining security properties. This enables secure migration to new algorithms either proactively, or as the need arises.

For example, while there is support in TUF for post-quantum cryptographic algorithms, many adopters may not have enabled it. A TUF repository can enable post-quantum crypto and re-sign its metadata using both algorithms, thus allowing current users

to securely transition to the new algorithm and protecting all users versus post-quantum attackers.

Here are some assumptions that are common but are not necessarily good ones in this day and age.

- *Misconception:* An attacker cannot hack a specific component or system. Modern systems tend to have so much code and tend to use so many libraries, that this just isn't a reasonable expectation. Even a "proven to be secure" microkernel like SeL4 has had security bugs found in it [1, 2, 3]. It is important to assume code could have bugs, especially large components, and to design your system to have different isolated compartments so that your system's security will degrade gracefully when components are successfully breached. This assumption seems to be on the way out, but some systems being created today do still use this assumption. You should assume that such compromises are a matter of when, not if [Catalog of Supply Chain Compromises].

- *Misconception:* A key or other secret will never be leaked, compromised, misgenerated, etc. Incidents violating this assumption are common [TAG Security Catalog of Supply Chain Compromises]. Modern systems should design revocation mechanisms that retain trust even when an attacker knows a secret and is a man-in-the-middle. Ideally, one should also design the system to prevent substantial harm while you work to address a secret disclosure.

- *Misconception:* Multifactor authentication (MFA) using SMS is a sufficient barrier. This assumption isn't actually a bad one for MFA that does not use SMS. An organization using authenticator apps or hardware tokens seems to do quite well from a security standpoint (barring a few minor hiccups [Wired Magazine: The Full Story of the Stunning RSA Hack Can Finally Be Told], which do not seem to be indicative of a trend). However, the same is not true of SMS based MFA systems, which have been shown to be vulnerable to attack. So, do try to have your organization not only mandate MFA, but choose a means of performing it which provides a level of security appropriate for what you are protecting.

- *Misconception:* The complexity of parsing code for a complex format is not particularly relevant when considering security. This is a common mistake that organizations make, where the code to parse data formats or keys becomes a major liability. The number of X.509 certificate parsing errors alone that have led to security vulnerabilities is astonishing [MatrixSSL: Security Vulnerabilities]. A related problem in this space is that even just getting a format serialized into a consistent format is a more difficult challenge than many developers initially realize. So the complexity of the data communication and storage format should be a major concern, especially for sensitive API calls and components.

Note that the following assumptions were thought reasonable at one time, but have been shown not to hold well in practice:

- *Misconception:* Operating system user access control protections like file permissions are an impassable barrier. It turns out that it is often not that difficult to escalate privilege when gaining access to an account on a system. The reason is

that the operating system's system call boundary is massive and hard to employ effective controls on. You should not willingly let attackers into a system and rely on user permission bits, file ACLs, etc. as your only means of protection. Rather think of these as a barrier that may slow or trip up an attacker, but are not reliable as a line of defense.

- *Misconception:* The network cannot be tampered with. It turns out that becoming a man-in-the-middle is possible in many scenarios, including wireless attacks in a coffee shop, BGP route hijacking, DNS cache poisoning, etc. While it isn't trivial for any person to become a man-in-the-middle for a network path between two randomly selected computers, it certainly isn't unobtainable for a large and important class of attackers.

- *Misconception:* Software provided by dependencies are secure so long as we take minimal care when adding them. Attackers in some ecosystems have begun attacking software projects by taking over a dependency and adding malicious code. In other cases, a dependency is simply neglected for a long time and does not receive security patches. In yet other cases, an organization simply forgets or neglects to update dependencies to a later version so that a vulnerable version remains in use. Like the software your organization writes itself, dependencies need care, examination, and attention so that they do not become liabilities.

- *Misconception:* Firewalls keep out bad guys. Firewalls are an important tool for helping to provide compartmentalization of networked components. However, experience shows that they are insufficient on their own. In practice, many attacks involve an attacker bypassing firewalls and network monitoring systems to to access things that should have been restricted. This is not surprising given how difficult it is to write a policy that stops exactly all of the "bad things" and allows exactly all of the "good things". So, it may be helpful to think about this as a way to increase the difficulty for an attacker rather than as a means for stopping them outright.

- *Misconception:* Antivirus stops malware on end hosts. In much the same way, antivirus software on client machines largely just helps to make certain compromises less likely, but comes with its own risks and concerns. Today many experts recommend only using the antivirus software that comes with your operating system (if applicable). However, purchasing commercial antivirus software gives questionable benefits and does come with some added risk. [Project Zero: How to Compromise the Enterprise Endpoint, The Register: Avast antivirus hole patched after public Project Zero slap, CSO: Google researcher reveals more Kaspersky bugs, calls out the irony of antivirus].

- *Misconception:* Users can be trusted to choose and manage sufficiently secure passwords. This is patently false, which is one reason why multi-factor authentication is an option or even a requirement for many systems. Strong password guidelines for users are important. Users should also be incentivized to use tools like password managers.

Note that this does not mean you should not employ the controls in the above area! It just means that long held assumptions on the efficacy of these measures should be restated and that they are better used as part of a layered approach to make things harder, instead of infallible controls. They should not be relied on alone to stop a skilled attacker.

System Non-goals

In addition to goals, another key aspect to consider are things that you consider to be non-goals of the system. These are "illegal moves" in the game. They tend to come in two types, the first being things that you simply do not care about if they occur.

For example, TrashPanda Bank is likely well aware that people off the street may wander into the bank. Some of those people may steal a pen or the deposit sheets that are left out on the desks. They may use the bathroom and enjoy the heat / air conditioning without being a customer. However, TrashPanda Bank may also just assume that those costs are minimal and any effort to deter such actions would have a negative impact on the experience of other customers. So, solving these types of issues may be a non-goal.

The second type of common non-goal is one that seems too fanciful for the attacker to carry out. For example, let's say in order to break into TrashPanda Bank, the attacker will become president of the country and launch a nuclear strike on the vault. Whether or not the vault resists such an attack, any surviving members of the company are likely to be focused on things other than the vault. So, TrashPanda Bank could consider worrying about such an attack a non-goal.

Attacker Goals

Another way to frame the system goals is to talk about what an attacker may want to accomplish. This is sometimes (mis-)used to say that these goals are the only things an attacker would want to do, and so the system's goals should just be to prevent those. Unfortunately this line of reasoning will often miss cases because it is assumed the attacker will simply not care to perform them. In the movie The Dark Knight, there is a famous (and long) story told by Alfred, which concludes with the statement "Some men just wanna watch the world burn".

You should assume that someone will have the temptation to do a bad thing if it is possible to do so without a massive amount of skill and resources.

Using STRIDE To Enumerate Attacks and Goals

Fortunately, security researchers have long understood that it is too easy to miss computer security concerns when threat modeling. To aid in going through different cases, there is a model called STRIDE. STRIDE stands for the following properties:

- **Spoofing:** The act of using another's credentials. This can be for many purposes, such as gaining access to a resource they should not have access to, or masking the source of an attack. Commonly, this is done by authenticating as a different user when performing an activity.

- **Tampering:** The act of modifying information in a malicious way. This depends a lot on the project, but can involve things like replacing a user's data with something else, manipulating account balances, or changing log information.

- **Repudiation:** The act of performing an action but asserting you did not in situations where others cannot prove otherwise. This involves situations where the attacker makes tracing the cause of a problem infeasible.

- **Information Disclosure:** This is when you make private information public. Situations where

this occurs typically involve data leaks of user account data, private messages, financial details, etc.

- **Denial of Service**: This is where an attack prevents legitimate users from accessing information or services they are supposed to have access to. This can be very localized, such as locking a user out of their account, or very broad, such as bringing down an entire website. Attacks of this type are sometimes done using a set of computers that work together to attack a system. An attack of this type by a distributed set of computers is called a DDoS attack (Distributed Denial of Service attack), which you likely have heard mentioned before.

- **Escalation of Privilege:** This is the act of gaining more authorization to perform actions in a system that should not be granted. Note that while Spoofing focuses on appearing to be someone else, Escalation of Privilege focuses on using an identity you have to do things which should not be authorized. For example, consider the administrative assistant for TrashPanda's CFO. For withdrawals over a certain amount, the CFO may be required to place her signature on the transaction confirmation. However, if the administrative assistant for the CFO states that the CFO authorized it, the teller may (incorrectly) still complete the transaction. This is a case where the administrative assistant has escalated his privilege to do an action he was not authorized to perform.

Attack	Property violated	Impact
Spoofing	Authentication	Misdirected identity
Tampering	Integrity	Unreliable data
Repudiation	Non-repudiation	Lack of ownership for actions
Information disclosure	Confidentiality, Privacy	Lack of confidentiality
Denial of service	Availability	Unreliable service
Escalation of privilege	Authorization	Grants Unauthorized access

Justin Cappos's commentary: STRIDE has been used for a long period of time, but unfortunately has portions that don't apply as well to modern distributed systems. So the notion of escalating privilege could be thought better as the ability to move laterally (break the boundaries between actors) in a system. In other words, once an attacker gains access to X, are they able to find a way to get access to Y? This involves a failure to sufficiently compartmentalize X and Y from each other.

Also, the notions of spoofing and escalation should be thought of in an additional way that a reader may not initially consider. Distributed systems often use a concept called a token (also called a capability in some literature), where an API request contains information to authorize the transaction. In these cases, authentication is not needed. The API request token is sufficient to authorize access. This is much like a movie ticket being sufficient to grant access to a movie. There is no need to check the attendee's identification, so long as they possess a valid ticket. So, for Eve to gain access to Bob's data, it doesn't necessarily mean that she must know Bob's password. She may have just gained access to a token that some service uses to perform actions on behalf of Bob. She may even confuse the service into doing the actions she wants using Bob's token. Of course, if tokens are not used and service X is just always trusted to do a set of actions, spoofing and escalation become trivial once you compromise a service!

Ragashree Shekar's commentary: In the current landscape, with more and more data generated from each of us through the surplus connected devices we use, it also gives an opportunity to gather more and more data about us and utilize it to enhance their business.

It is about time privacy is engineered into each project we build that collects personal, health or protected user information not just to comply with the regulations, but also to protect user's right to privacy. LINDDUN is a privacy engineering framework that helps model the system, find and manage the threats associated with this system. LINDDUN categorizes the threats into 7 categories such as Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, and Non-compliance. Let's look at each one of them:

Linkability tries to find if an attacker is able to link two items of interest without knowing the data subject [1] corresponding to these items.

Identifiability tries to find if the attacker is able to identify a data subject from a set of data objects through items of interest

Non-repudiation is when a data subject cannot deny an action.

Detectability: An attacker is able to distinguish whether an item of interest about a data subject exists or not, regardless of being able to read the contents itself.

Disclosure of Information: An attacker is able to learn the content of an item of interest about a data subject

Unawareness: The data subject is unaware of the collection, processing, storage or sharing activities (and the corresponding purposes) of the data subject's personal data.

Non-compliance: The processing, storage, or handling of personal data is not compliant with legislation, regulation, and/or policy.

A few notes to consider. First, Identifiability and Linkability are closely associated with each other as lack of anonymization affects results in identifying 2 data subjects or linking two different data objects.

Second, Awareness is a big part of the privacy laws, regulations and policies and failing to inform the data subject of what data about them are being collected, how it would be processed/used, who else would it be shared/sold to, and to let the data subjects decide if they want to opt-in. Thus unawareness is a subset of non-compliance.

References

[1] LINDDUN | LINDDUN

[2] Data subject is an identified or identifiable natural person.

Art. 4 GDPR - Definitions - GDPR.eu

Attack Graphs: A Useful Technique

Once you understand the potential attacker(s) and a goal, it is helpful to think through the ways in which they could achieve this. While you can just sit and do this in whatever way you want, it is often useful to reason about this by brainstorming using a tool called an Attack Graph. (Note, this is also called an Attack Tree or Threat Tree / Graph in some literature.)

An attack tree has at the top (which is called the root node), the goal of the attacker. For example, the attack tree in the figure, has "Open Safe" as the root node, so this is the attacker's goal. The nodes in the tree (i.e., the square boxes) are connected by one or more edges (the lines between boxes). For two nodes that have an edge, the higher node is called the parent and the lower node is the child. The child node or nodes are more details about how to achieve the parent node.

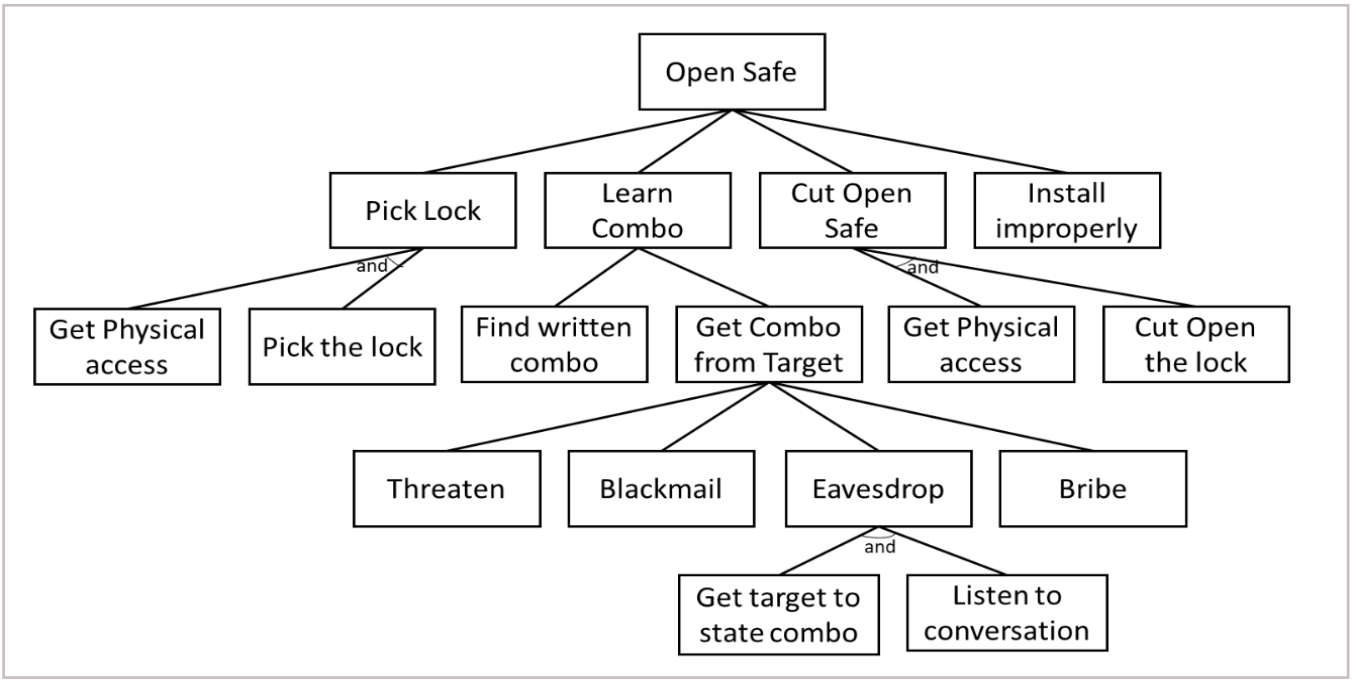
In the next stage, we can see that the goal of learning the combination can be achieved in two ways - finding the written combination and getting the combination from the target (who is an authorized individual possessing the combination to the safe) which can further be done in 4 ways. These represent the OR nodes. Success in any one of these attacks leads to success of the ultimate goal of opening the lock. One attack to retrieve the combination from the target includes eavesdropping, which needs the success of two attacks where the victim states the combination and the attacker listening to the conversation. Failure of either results in an unsuccessful attempt to break the lock open.

Justin Cappos's commentary: Attack graphs were really helpful for me when I was first starting to threat model large systems and also are really helpful now when I don't understand a system well. Today, I often can intuitively go through and enumerate the cases here because I've had enough practice. So, I rarely write out an attack graph. (I usually jump straight to attack matrices, which will be described later.)

You can think of the exercise of writing out an attack graph like writing out your multiplication tables by hand before you have them memorized. Eventually it may become second nature, but it will be an immense help at first. If you're starting out, I strongly encourage you to start with attack trees though and get practice with them. This will help you build the foundation you need to do more accurate threat assessments.

One problem with attack graphs is you don't necessarily know how complete they are. There are a wide array of things that you haven't thought of. Be sure to think back to your system goals carefully and focus on them. When you reason about the situations where those goals hold, think about what those situations mean for an attacker. How is the attacker constrained? What can the attacker do? You may need to update the goals and other parts of the writeup as you go through this process.

SAMPLE ATTACK TREE



Reference: [Academic: Attack Trees - Schneier on Security]

There is a depth of material on attack trees that focuses on adding parameters of different types to them. They can do things like help you reason about what attackers with different skill sets / access / constraints might do in a system or how much an attack might cost an attacker. As you are working through examples, you may find it useful to refer to the following reference: Schneier, B. "Attack Trees." Schneier on Security, Dr. Dobb's Journal, December 1999 .

Jack Kelly's commentary: For some clients or colleagues Attack Graphs and Trees are a valued deliverable. They are most valued by visual learners and non-technical persons as a tangible representation of what is elaborated on in a Threat Matrix. An Attack Graph helps a reader easily follow from initial breach to the attacker's goal, and identify which nodes on the graph may be a hotspot either for traversal to other goals, or is used in many possible routes to the same point of impact. This provides a quantifiable justification for the controls used to remediate the threat of attack.

Attack Graphs can be intensive to build out and maintain, so it is recommended to use a solution that can generate Attack Graphs from code.

Not all attacks are the same

Note that not all avenues of attack will have the same properties. Some attacks an attacker can only do once and if it fails they will be caught, while others an attacker can do repeatedly. Some require specialized skills, while others can be done by anyone.

Justin Cappos's commentary: It is helpful when thinking about attacks to really think outside the box. One exercise I like to do is to "prove" why an attack couldn't happen. As I'm reasoning through it, I usually come up with the way in which the attack could occur.

For example, consider TrashPanda Bank. If I'm thinking of how to get into the vault, I might think "It's not possible because there is a guard during the day and an alarm system (which automatically triggers a lockdown) at night. Even if you get past those, you need to have the manager key and a teller key to open the vault." I would turn thought into "In order to break into the vault, you need to somehow bypass a guard during the day or the alarm system at night. The attacker needs a manager key and teller key..." and then proceed from there to devise under what circumstances this would be possible.

It is also important to question your assumptions a bit when doing this process. So, you should also consider that this all assumes that the alarm system functions properly, the locking mechanism in the vault operates as designed, the vault was correctly installed and so drilling in, etc. are impractical.

Different attacks can have different impact

Not every compromise is the same in a system. In some cases an attacker gains only limited access to a system. In others, they may have total control. We talk about these differences by talking about the "impact" of an attack.

You can think of impact as the monetary cost, reputational cost, etc. to an action having occurred. However, it is hard to know an exact value for this. What is the cost of having leaked a large amount of private customer data? Unfortunately, this seems to happen

fairly regularly for some large companies and very little actually occurs. In other cases, a company may face lawsuits from investors and customers or fines from regulators after a security breach. This makes the impact easier to quantify.

Using DREAD to estimate the expected impact of a threat

DREAD stands for impact categories for Damage, Reproducibility, Exploitability, Affected users, Discoverability. The idea is to provide a measurable means to quantify the impact of an attack by rating the attack between 0-10 in the impact categories, 0 being no impact and 10 being the highest impact. The final impact is the average of the impact across these categories.

> Impact score = (Damage + Reproducibility + Exploitability + Affected users + Discoverability)/5

Let's dive into what each of these impact category means:

1. **Damage:** The potential destruction the attack is capable of causing for the assets in the scope. In this context of information security, information disclosure is the damage. 0 stands for no damage, 10 stands for destruction of the information or information system serving this data causing denial of service.
2. **Reproducibility:** Reproducibility stands for how easy is it to reproduce this attack? Is it just very juvenile or does it need experience to find the vulnerability and cause this attack? Zero refers to Difficult or impossible and 10 refers to very easy to reproduce.
3. **Exploitability:** Complimentary to Reproducibility, exploitability refers to what is needed to ensure the attack is successful? Does it take advanced scripting or tools to exploit the vulnerability or is it as simple as adding the string " OR 1=1?[1] 0 refers to practically infeasible computational power or sophisticated tools & techniques, whereas 10 refers to just an availability of interface to interact with the target application such as browser or command line etc.
4. **Affected users:** Affected users refers to how many users are impacted by this attack, ranging from no users (0) to all non-administrator users to all-users and administrators alike (10).
5. **Discoverability:** Discoverability refers to how easy it is to find the attack in the first place. Is it evident in the plain sight (for example use of components with publicly disclosed vulnerabilities, authentication in the URL, or directory traversal) or is it hard to disclose? The scores range from 0 (very hard to discover) to 10 (very easy to discover).

Impact category	Description	Ratings Range
Damage	How bad is the damage?	No Damage = 0 Complete destruction = 10
Reproducibility	How easy is to reproduce this attack?	Difficult to reproduce = 0 Easy to reproduce = 10
Exploitability	How easy is it to cause this attack?	Difficult or practically infeasible = 0 Easy to exploit = 10
Effected users	Which users does this attack impact?	No users = 0 All users across privilege levels = 10
Discoverability	How easy is it to discover?	Difficult to discover = 0 Easy to discover = 10

The DREAD framework provides a rough estimate of the expected impact of a threat, thus limiting the barrier to entry. While the framework looks seamingly simple,

---

1 For the reader unfamiliar with SQL injection, when " OR 1=1 is inserted in a form field it will cause many SQL queries to evaluate to true. This can cause things that are supposed to be conditional, always to be executed. So it can do things like cause all the user data to be displayed instead of only the information for the current use.

While the framework looks seamingly simple, the accurate analysis in complex ecosystems needs extensive information security expertise with up to date knowledge in the domain.

In practice, many security experts argue that discoverability is both hard to quantify and so often gotten wrong. As a result, it is suggested to use DREAD without trying to estimate D (Discoverability). To do this, you would always mark Discoverability as a 10.

For more information on the DREAD model, refer to DREAD (risk assessment model) - Wikipedia.

Likelihood / risk

A very useful concept when thinking about security assessments is the concept of risk. Rather than simply categorize things as possible and impossible, risk lets us try to understand how likely they are. If you have two equally negative outcomes which could be addressed with the same amount of effort, the more likely one is the one to focus on first.

Unfortunately, there really isn't a solid way to know how likely certain events are in computer systems. These are uncommon events and advances that attackers make lead to huge advances in attack capabilities. However, in general, most people underestimate unlikely events. To be blunt, the state of the field is commonly that one tries to list existing anecdotal examples and once that occurs in a sufficiently public way, everyone seems to agree that this is now something to be concerned about.

Realistically, you get the most value out of understanding roughly how likely things are 1-in-100 vs 1-in-a-million vs 1-in-a-trillion, etc. versus trying to put an exact number.

Justin Cappos's commentary: I worked with Evan Gilman, Matt Moyer, and Enrico Schiattarella from the SPIFFE / SPIRE team on a threat assessment and as part of it we tried to quantify risk. We each did this independently for aspects of the system; our answers often varied by more than 10. In fact, in one case it varied by more than a factor of 1000! After discussing these differences, we began to better understand ways in which our mental models differed about how the system could be deployed. This was a really useful exercise for us even though I don't think any of us put a lot of faith that the values we ended up with are close to the real value.

Expected damage

So if one understands the likelihood of things happening, how does that help if the impact of those things differs? Well, fortunately, there is a simple formula to compute the expected damage from an attack:

Expected damage ~= likelihood * impact

For example, if something has a 1-in-100 chance of occurring on a specific day, and costs you $1000 when it occurs, you expect that the amount you'll have to pay over a long period is about $10 per day.

When addressing risks, you can look at how much your protection would cost (in terms of effort, money, etc.) and how this changes the expected damage. This would be an ideal way to prioritize how to work on things. So why don't we do this? Because the actual values for likelihood and impact aren't really known in practice. So understanding "that things that are likely and high impact are really bad and need to be addressed" is going to be more useful in practice than the actual formula will be.

Andrew Martin's commentary: We have found it

helpful to list the remediations and controls from a threat model in precedence order. The recipient of a threat model is likely to be a risk owner such as a CISO or equivalent holder of funds, and the model should inspire them to remediate immediate existential threats, or threats with unacceptable impacts on business functionality, and consider which of the other scoped threats are worth investing in. Expected damage is a useful metric for risk management at the executive level, and it can be modulated with the secondary data point of likelihood — existential risks should be addressed in some manner, but it's also acceptable to mitigate them in other ways (such as transferral with disclaimers or insurance policies, or acceptance of low likelihood). Each mitigation is a complex tree of possibly catastrophic permutations and so should be explicitly addressed by the risk owner.

How do we make sure we didn't miss anything?

One common problem is that it is easy to miss one or more cases when doing threat modeling. With distributed systems that have many components, this problem becomes much more common. The reason is that there are many different combinations of components that could be compromised by an attacker and used collectively to do nefarious things.

For example, suppose that in TrashPanda Bank suppose that the vault is locked and may only be unlocked by the manager's key and a key from any one of the tellers. People going into the vault are also checked by a security guard to ensure they are escorted in by the manager. All vault entry and exit times are logged by the security guard. The security guard notifies the manager when the customer leaves so that the manager and teller may retrieve their keys and re-lock the vault, which the guard confirms to the manager.

If you threat model this situation, you also need to consider cases where a malicious security guard can work with a malicious customer to do something bad, by not logging their entry. Suppose the customer enters the vault and starts a fire or does some similar action. If the customer isn't logged, it will not be possible to know who to blame. Even worse, the guard could potentially add a log entry to indicate that a different customer entered, blaming them for the incident. The system has lost forensic traceability (the ability to know what happened) of these events due to having insufficient protections over the security guard being malicious and working in coordination with a malicious customer.

Similarly, if a teller and guard work together, they could simply fail to re-lock the vault after a customer leaves. The teller could fail to perform the action and the guard could simply say to the manager that the vault was re-locked.

Attack Matrices

As the simple example above shows, there can be a number of fairly complex interactions between actors when they could be malicious and act in unison. We could just write one big massive block of text to describe all of the interactions in the system and how different malicious actors can cause harm. This would be really unwieldy to read and to ensure we didn't miss any cases, so instead we recommend you write it in a way that a reader can more easily reference.

To do so we use a representation called an attack matrix. An attack matrix is typically written so that the rows of the matrix correspond to a set of actors that are under the control of the attacker. The columns

often represent different security designs that you may want to evaluate or things like different capabilities the attacker may have. What you are effectively doing is putting the text for what an attacker can do in the part of the matrix that corresponds to that set of capabilities.

Let's look at a few example attack matrix entries for the previous section's example vault at TrashPanda Bank.

Malicious actor(s)	Impact
Customer + guard	Loss of forensic trace-ability from customer malicious actions. Able to falsely blame other customers for malicious actions
Teller + guard	Vault may remain unlocked after a customer visits the vault, when this teller and guard are working

Reducing the number of rows (actors)

Note that if we continue to fill out the matrix above, there will be quite a few rows due to the fact that there are $2^{\text{number of actors}}$ different combinations of malicious actors. When the number of actors is even moderately large (like 5 or 6), this can be overly burdensome. Fortunately for us, in most cases the number of interesting sets of actors is actually quite small. For example, if the teller, guard, and manager work together, they can really do as they please and so a customer also being malicious really doesn't add any further impact to the attacks that can be performed.

A few useful rules to consider:

- A superset of a set of malicious actors can do at least the union of all subsets of those actors. In other words, if a teller+manager can have can have impact X, a manager+customer can have impact Y, and a teller+customer can have impact Z. The manager+customer+teller can have any impact from X, Y, and Z. In fact, the impact may be greater than this because X, Y, and Z may be limited by checks the non-malicious party performs.

- It is common for many rows to subsume other rows. This is for two reasons. First, once a certain level of compromise is reached, usually the attacker effectively has full control of the system. In this case, additional compromises do not change the security impact of the attack. Second, some parties are so limited that their ability to harm a system has minimal added impact. So, whether they are malicious or not is inconsequential.

- Many capabilities are quite easy to get in practice. So, if this is the case, it may be better to assume that an attacker already has those capabilities in all cases in the matrix. For example, it is common to assume a man-in-the-middle attacker who can intercept and modify network traffic. Breaking the table down into attackers that can and cannot do this may make the table unnecessarily long.

A question arises, if you have different ways to get the same impact, how do you label the row? In an attack matrix, what you do is to take the minimal set of actors that will cause a certain impact and label the row with it. This indicates that any attacker with at least these parties compromised, can perform this action.

Notice also that in some cases the impact of a compromise of different (disjoint) parties could be the same. For example, suppose that teller+guard and manager+guard have the same impact. In this

case, it is sensible to write the row as teller+guard OR manager+guard to save space instead of having two duplicate rows.

These space saving tips do not fully solve the problem though. Consider that the matrix we wrote before has the customer+guard row (such as above) as well as the potential for us to add a teller+manager+guard row. How do you know which row of the matrix to use? To make this clear to the reader you should sort the attack matrix so that the most impactful attacks are lower in the matrix. When reading an attack matrix and reasoning about a scenario, move down the matrix to find the lowest row that you match and then use this cell to determine the impact.

For more information about threat matrices, here are some references for further reading:

G. Almashaqbeh, A. Bishop and J. Cappos, "ABC: A Cryptocurrency-Focused Threat Modeling Framework," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 859-864, doi: 10.1109/INFCOMW.2019.8845101. https://arxiv.org/abs/1903.03422

Matt Tatam, Bharanidharan Shanmugam, Sami Azam, Krishnan Kannoorpatti, "A review of threat modelling approaches for APT-style attacks", Heliyon, Volume 7, Issue 1, 2021, ISSN 2405-8440, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7814160/ https://www.sciencedirect.com/science/article/pii/S2405844021000748)

Rajesh Gupta, Sudeep Tanwar, Sudhanshu Tyagi, Neeraj Kumar, "Machine Learning Models for Secure Data Analytics: A taxonomy and threat model", Computer Communications, Volume 153, 2020, Pages 406-440, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2020.02.008 https://www.sciencedirect.com/science/article/pii/S0140366419318493

Zhang, L., Taal, A., Cushing, R. et al. "A risk-level assessment system based on the STRIDE/DREAD model for digital data marketplaces." Int. J. Inf. Secur. 21, 509–525 (2022). https://doi.org/10.1007/s10207-021-00566-3

FINAL THOUGHTS ON THREAT MODELING

Threat modeling is not an exact science, but is necessary to reason about what you are protecting a system against and in what scenarios those protection holds. It helps you understand weaknesses in your design and, as we will discuss in a moment, how prospective security designs will impact that set of weaknesses. With a little practice, the art of threat modeling can become second nature.

Justin Cappos's commentary: I find it useful to talk through my threat models with other people. The act of explaining something aloud can really force you to go through and rethink it.

When you do this, it doesn't have to be a technical security person actually! You can talk with a friend you've known since primary school, your pet, or even a stuffed animal. The important thing is to make yourself reason carefully through what you are claiming in a way that you reconsider your assumptions and conclusions.

I find it most useful to discuss threat models with people that are skeptical of the value of a system. They will certainly push back on any claims you make that are overbroad. Them poking holes in your statement help you to be more precise about the value and benefits of the system you are describing. Just remember to keep the statements falsifiable and you can have a productive, factual argument.

MITIGATING RISK WITH EXPERT INSIGHT

SO THERE ARE RISKS... HOW DO WE DEAL WITH THEM?

The core concept of defensive security is to take things that are damaging and either make them less likely or less impactful.

To better reason about this, we will look at several capabilities that a defender often retains even when attacked. Note that this is not an exhaustive list, but these are the most common properties that exist today, so deserve emphasis. The capabilities we will discuss in detail are detection, non-repudiation, recovery, and prevention.

Detection

Another important aspect is what is done when an attack occurs. In the worst case, the attacker could try repeatedly and the defender would never realize an attack is occurring. This is very common if the attacker can download and run the defender's software locally on their own infrastructure because then the attacker can experiment with a running copy of the system. This is basically the norm for open source software and is also common for proprietary software.

If a system is attacked, ideally you'd like to know it. This is where detection comes in. Detection is any means by which you can know you've been attacked. Common ways to know this involve logging API calls, examining network traffic, and looking for anomalous events by collecting measurements of anything deemed as deviation from standard system behavior.

It is common for many systems to be constantly under attack. What is more important is detecting successful attacks and determining their severity. A person who steals a pen from TrashPanda Bank is less of a concern than one who steals the vault keys from the manager!

Non-repudiation / Forensic Traceability

Once you've been attacked successfully, you may have an intruder in your systems. You may need to look through what has occurred to understand which actions an intruder performed and which were legitimate actions by the normal system.

If in TrashPanda Bank, Bob the teller says that Alice the manager asked him to give her the contents of his cash drawer, but Alice denies this, how do we know who to believe? Well, if Bob got a receipt or there exists a video recording then there may be a way to prove who is lying and who is honest.

In computer security, this is usually done by having something called **non-repudiation**. This is where a statement is made such that it later can be proven that a specific party actually made it. This is usually done by the party (Alice, let's say) signing it with a private key that only she owns. Then any party with Alice's public key can verify that Alice (or a party who compromised her private key) made that statement. So, as you can see non-repudiation is essential for post-attack forensics and should be a goal for any systems with multiple actors.

Note, it is possible to have differing amounts of non-repudiation and detection in a system. If TrashPanda Bank counts money for the whole bank at the end of each day, the bank may be able to quickly detect

if something does not add up. However, this does not mean that they will know who is responsible. Conversely, if TrashPanda Bank has video recordings for all time, but never checks them, then they will not detect problems well, but when they do can figure out exactly what occurred.

Recovery

Once you know an attack has occurred, a major goal is to get the attacker out of your system. In some cases, this is very difficult. If an attacker gained the ability to install software as root on your devices, for example, then they could have installed basically any software (rootkits, firmware, etc.) and so you may need to start over.

Fortunately well designed systems usually have the ability to securely recover from a compromise. Note that it is common to assume that an attacker could act as a man-in-the-middle for your users. So, if you have a compromise of a system, you can't securely revoke or restore trust using the same key that was compromised. [compartmentalization needed again]

Recovery is a very important property to have. However, in general, it isn't possible to recover in every case. After all, if every actor in a system is compromised, it doesn't seem possible to ever move back to a state with a trustworthy root of trust without starting anew.

Prevention

Another means to deal with an attack is simply to prevent it from being effective. The previous sentence, using the word "simply" is a bit misleading because this is often one of the most difficult things to do. Well designed systems have this property for most types of attacks.

Note that you need to carefully be able to argue why you protect against a set of attacks. This includes in what scenarios an attacker is prevented from doing an action. Once again, being rigorous and clear about limitations are absolutely key.

Justin Cappos's commentary: Some modern systems provide prevention of only certain attacker actions, in only certain scenarios. They may prevent information from being valuable after a certain point of time or from a key from being exfiltrated after a successful attack. (See HSMs, the concept of perfect forward secrecy, and ephemeral keys, as examples.) These properties are certainly nice to have, but ideally you want full prevention as a goal.

Which of the prior is the best?

A natural next thing to consider once you understand the different means by which you can handle a compromise, is whether there is an implicit order so that prevention is always better than detection, for example. It turns out that this is not always the case. For example, suppose that TrashPanda Bank could detect Eve embezzling a small amount of money. Alternatively, they could have a means to prevent Eve from doing so, but not detect her attempt. In this case, TrashPanda Bank's management may feel it is worth the small financial loss to know Eve is unreliable and fire / prosecute her.

To consider another example, let's say that TrashPanda Bank has a super alarm system that can detect when the view of any sensor is blocked momentarily. Unfortunately, TrashPanda Bank is set near a set of cherry blossom trees and when the blossoms fall, they block the sensors, leading to a ton of false alarms. Suppose that TrashPanda

set the alarm to automatically ring the police when it was triggered. After being summoned several times, the police are unlikely to respond to alarms for TrashPanda in the future, leading to the police ignoring an alarm on the vault. So, in this case, the security system's drawbacks may actually degrade security.

Justin Cappos's commentary: While the example above is a bit silly, adding a security mechanism does sometimes degrade security in practice. It used to be thought that changing passwords frequently was an important security practice. It was later shown that this made users choose weaker passwords, reuse passwords more often, and led to companies providing more vulnerable means to recover lost passwords.

However, this all being said, there is actually a practical hierarchy of what defender's capabilities are usually preferable. Usually prevention is the best because it actually stops the negative outcome from occurring at all. Recovery is really, really important for all but the most unlikely of events. Note that manual effort for recovery is common which is often reasonable. However, this implies that this also should be a rare act to avoid overburdening the poor person who does the recovery. Detection is important, but can be overwhelming if it is overly broad. If you can detect problems, but cannot forensically trace the cause, it can lead to a lot of extra work.

So, while it is not always true, in general:

Prevention
>
Recovery
>
Detection w/ Forensic Traceability
>
Detection
>
Forensic Traceability

Mitigation quality is important, but where you apply them is more critical

Applying mitigations is usually not as simple as just choosing a set of mitigations and applying them to parts of your system. A common mistake that I see novice system designers make is to focus more on the quantity and type of security mechanisms added than focusing on where and why. You need to reason about the goals your system has and then figure out how to intelligently apply mechanisms and controls to meet those goals.

To understand why, let's go back to TrashPanda Bank and think about their security. If they buy and deploy the latest alarm system, but apply it to the manager's snack drawer instead of the bank vault, they will not get the desired security benefits!

This also helps to explain why it is so important to design security into a system from the start instead of trying to bolt it on afterwards. If you don't design things well from the start, it is often impractical or even impossible to get the security properties you want later... at least without starting over.

A few more tips about applying mitigations

It is important to have a system that degrades gracefully under attack. This means that an attacker must compromise many parts of the system that are well protected and compartmentalized from each other in order to do substantial harm. So, think of how to make a system that slowly loses security properties as compromises occur, rather than one that has only "secure" and "insecure" states.

Note that you need to consider lateral movement in a system very carefully when thinking about a system degrading gracefully. If the ability to do X gives one the ability to do Y, then security does not degrade gracefully with respect to these two. If you can only get Y by obtaining the capability for X and Z (which are compartmentalized), then you have actually made the attacker's life harder than compromising X if their goal is Y.

Another really key thing to do is to protect all access to something sensitive. (This concept is called complete mediation.) If TrashPanda Bank has a well fortified vault entrance with guards, etc. but has an unlocked, unmonitored window in the vault, the attacker will likely just use that. Violations of complete mediation are extremely common in systems where security was not designed in from the start. The reason is that the defenders may be unaware of an inappropriately secured action or be unable to secure some set of actions due to design flaws.

Marco De Benedictis's commentary: Attack Graphs capture the defenders' mindset and working process, and so are time-consuming and require significant effort to generate to ensure correctness and the completeness of the paths that an attacker could exploit to achieve a potential goal.

We can understand if tactical security controls are addressing the most relevant threats by cross-referencing the attack graphs back to the proposed mitigations. This can be practically achieved by overlaying security countermeasures at each individual step, and visually inspecting the branches that aren't properly covered by remeditations.

This visualization allows us to evaluate the effectiveness of our security assessment, and to surface the residual risks by identifying the branches with insufficient security controls and suggesting remediations that satisfy the greatest number of branches at once, taking into account their ease of maintenance, business requirements, and budget implications.

TAG-SECURITY ASSESSMENTS

TAG-Security Assessments

Here we discuss TAG-Security in the CNCF and the way that the TAG-Security collectively decided to do security assessments. This draws on experience from many people in the community who do security audits and assessments professionally. While the original framework for this is based upon prior work by the community, especially threat modeling in the Secure Systems Lab at NYU and the SPIFFE / SPIRE projects community assessment, it should be noted that this document and process has benefited from additions and improvements from dozens of community members.

What is a TAG-Security Security Assessment?

A TAG-Security Security Assessment (TSSA), is as the name might indicate, a security assessment. However, it is specially tailored to be concerned primarily with cloud native technologies, as that is the purview of the CNCF. This includes distributed systems that are designed to be highly available and that can be deployed dynamically and elastically at web scale. As such, it has an opinionated take that is specific to this domain. For example, since there tend to be many actors in cloud native projects, this assessment process often uses threat matrices. Whereas, threat matrices are less useful for something with few actors like threat modeling an Android application or a website.

There is a basic self assessment which is done by projects to give a means for community feedback. The self assessment, while valuable in its own right, is built upon by a joint security assessment that is done with TAG-Security and project members.

There is also a process to help a project complete a self assessment called the Security Pals process. One or more Security Pals will effectively draft a self assessment and then work with the project to validate and refine it.

What is the right time in a project's lifecycle to do a security assessment?

In an ideal world everyone would do a security assessment of their project while forming the design in order to ensure that the design will meet the security goals. Most importantly, if you are designing a security focused system, you need to understand what you are trying to protect against. If you haven't threat modeled the system ahead of time, your design is very unlikely to match your threat model well. This will lead to insecurity as well as bad user experience in many cases. So, at least some lightweight threat modeling in the design phase is standard practice for organizations that write security focused code.

In reference to a more formal assessment like a TSSA, this can be done whenever it makes sense for the project. As of the writing of this book, the TOC and other CNCF guidelines currently have most of that done around the time the project is accepted into incubation, however these guidelines are being revised currently.

In general, the earlier an assessment is done, the more secure the project will be and the easier it will be to adapt to any design or other changes that are uncovered by the assessment. So, do your best to start early!

What is the process for getting a security assessment?

The current detailed guidance for this process is at TAG Security Assessments Guide available on GitHub. This website is more rapidly refined based upon community feedback and experiences than this book. However, for completeness, we will describe this process at a high level.

First, the process will differ a little bit whether the project is an early stage (likely sandbox) project or a later stage project with adoption already. An early stage project will just do a self assessment and then provide this to TAG-Security, either by creating an issue or by asking to schedule a presentation. This is meant to be lightweight for all parties and give a means to get rapid feedback.

For projects that do not have the time or skills to do a self assessment on their own, the Security Pals process described in the next section helps with this. In this case, an external party helps to move your security assessment to a point where a project gets a solid self assessment.

A later stage project, for example one looking to advance in the CNCF, will go through a joint process involving review both from TAG-Security and the project. This TSSA is also set up as a joint assessment between a project (who are experts in the specific project technology) and reviewers from TAG-Security (many of whom are security experts). So the parties need to work together to share information as part of this process.

With the self assessment as groundwork, the TAG-Security team is now ready to help guide the assessment. The basic idea here is to flesh out the threat model and the attack matrices, looking for missing cases, hidden assumptions, and the like. This process benefits greatly from continued input from the project. The core reason is that with things like hidden assumptions, it can be challenging to understand exactly when they do not hold if you do not know the area.

Justin Cappos's commentary: When doing a security audit or assessment, I have often said something like "why do you think that this property is always true?" and then a project member has said "well except for case X, it is always handled." Then they realize that case X is really important and could occur. I've even had people then be very impressed and say "I was impressed how Justin realized that case X was a problem" where really that wasn't the case. In fact, I may not have known anything about case X. I just knew it was important to question non-obvious assumptions and see what breaks. Asking good questions is an important security superpower that anyone can quickly learn.

How does the Security Pals process work?

The role of the Security Pals is to act as short-term security-minded aid for projects that are not primarily security-focused or do not currently have the capacity, skillset, or available effort to complete a self assessment document.

The focus of the Security Pal's involvement will be on guiding the project in completing a Self Assessment that evaluates their security posture. This can be done with minimal work from the project.

The Security Pals will also provide guidance and insights to assist these projects in jump-starting

their security considerations and improve this self assessment across various channels, including Github, Slack, and synchronous TAG Security meetings.

The overall commitment from the Security Pal, as a TAG member, will be approximately two weeks per project. However, this time will vary by the complexity of the project being examined. A Security Pal may work alone or work on a team with other Security Pals.

Security Pals Stage 1: Preparation (<1 day effort)

- One or more Security Pals are identified and a GitHub issue is created / updated.
- The Security Pal should review existing information and documentation about the project in the form of prior KubeCon talks, webpages, project documentation, etc.
    - This will not require maintainer interaction.
    - If the project does want to interact at the earliest stages, they may elect to prepare a comprehensive 30-40min presentation describing the function and characteristics of the project undertaking a self assessment. This presentation should include an overview of the project and its architecture, existing security practices and concerns, and suggested security focus areas.
- A draft document for the self assessment can be created in a fork of the repository. (This will be submitted as a PR later.) This document should have the Metadata portion at the top completed, and placeholders for all of the sections.

Security Pals Stage 2: Understand the Project Landscape (~1-2 days effort)

A first step is for the Security Pals to understand the overall project at a sufficient level of detail. In essence, before doing more detailed security work, one should understand:

- What the project does / how it is roughly used
- What parties perform actions (e.g., a sidecar, central server, core project maintainer, etc.)
- What sort of actions are performed (e.g., collect telemetry data, provide a query language for users, release a new version of software, etc.)
- What is the project trying to achieve, mostly related to security (e.g., only the organization deploying the software should be able to access PII, all versions of software that are loaded must come from the core developer team)
- What the project is not trying to achieve, again with security as a focus (e.g., stopping a malicious insider at the organization from posting PII on social media).

Note, that these items correspond to the Overview section of the self assessment document.

If multiple Security Pals are working on a self assessment, it is recommended they do this step independently and then compare notes. This is the most important part for all Security Pals to internalize and understand for a project.

If the project is interested in examining this portion of the self assessment, then it would also be helpful for the Security Pals to make the project aware of it at this point. However, the Security Pals need not wait for a response, and can safely continue to the next stage.

Security Pals Stage 3: First complete draft of the Self Assessment (3-5 days effort)

At this point, the Security Pals should have a rough idea of the security goals, non-goals, actors, and actions. Now it is time to make a pass over the remaining sections with the existing context.

- Self assessment use
- Security functions and features
- Project compliance
- Secure development practices
- Security issue resolution
- Appendix

Note that it is a good practice to link back to the documentation when describing why a certain item is believed to be true. This is especially important so when the project does a later examination of this step, if they disagree or need to clarify something, they know where to do so. So, repeatedly link back to project documentation.

Another key item to do here is to indicate when information is not known. For example, it is fine to write lines like this:

The Flibble project's development process protects the key used to sign a new version of the Flibble sidecar. (How?)

Those will be resolved in the next stage.

It may be useful for the Security Pals to perform a Lightweight Threat Model based on this template. While this will not be checked in as part of the self assessment, this can help to point out areas where the self assessment needs to be further refined. Other models like STRIDE, etc. may also be useful here.

Security Pals Stage 4: Iteration with the project (~2-3 days effort)

At this point, the Security Pals need interactions with the project to further refine the document and resolve points which need clarification. This will consist of a few rounds of iteration where project maintainers provide further information which makes its way into the self assessment.

The real goal of this process is to accurately document the project's state. Ideally the project will also fix documentation issues that arose during the self assessment process, but the focus on the Security Pals is on getting this clarity, instead of pushing for security changes. (Those changes and recommendations are handled in the joint assessment which comes after this process.)

Note, this process usually will go in multiple rounds. For example, suppose we have the following line in the self assessment:

The Flibble project's development process protects the key used to sign a new version of the Flibble sidecar. (How?)

One of the project personnel may clarify that the key is stored in an HSM on the build server which is only accessible to maintainers. A revised statement might look like this:

The Flibble project's development process uses an HSM on the build server to protect the key used to sign a new version of the Flibble sidecar. Only maintainers have access to the build server (want to link to updated docs). (How does the build server authenticate maintainers? Is there logging to see when the key is used / who logs in? How do the systems that check the build server's key know they have the correct one?)

And after more clarification, the self assessment document may look like this:

The Flibble project's development process uses an HSM on the build server to protect the key used to sign a new version of the Flibble sidecar. Only maintainers have access to the build server, which is enforced manually by a system administrator, which is currently the maintainer Bob. MFA required for all maintainers, following NIST SP800-63B password guidelines, including requiring either a hardware token or authenticator app. Logins by maintainers and uses of the HSM key are not logged in any way currently. It is not possible to tell which maintainer's account used the key after this occurs.
Systems performing a software update have the public key of the build server added to the image at creation time, which serves as the root of trust for this verification. It is assumed that the build server always provides the latest version of a software image, but this is not verified.

Security Pals Stage 5: Finalization (1 day)

At this stage the Security Pals are ready to finalize this effort. There are two main things that need to be done.

The first is to present their findings. There are several appropriate audiences for this and the Security Assessment Facilitator can help to guide which outcome is most desirable.

- Present to the project maintainers (likely 20-40 mins). The self assessment is a good topic for a project's community meeting or similar. Developers from the project should be able to help to find and fix errors in the document. This may also help the project become better aware of which issues they should prioritize moving forward.
- Present to the Security TAG in the CNCF (likely ~20-25 mins). This will help get more review from security experts and likely lead to parties who are interested in the joint assessment being recruited.

The second and final step involves the self assessment PR being submitted for approval to the Security Assessment Facilitators for approval. At this point, a facilitator can merge the PR (adding the self assessment to the repository) and close the Security Pals Issue. At this point the self assessment is now finished, approved by the project, and the project is ready for the joint assessment to begin!

How to work with TAG-Security on a joint assessment

The most important thing is to be open minded and set aside some time to work with us. This is not the kind of process where you can simply sit and write a certain number of words or know you need to put in X hours and then it will be done. Even the self assessment phase is not predictable as it would be to fill out a questionnaire. While the process is designed not to be onerous for either us or you, depending on what is found in the process, the number of hours can vary dramatically. I have seen some assessments take only 5 or so hours from the project team, while others have taken more than 40 hours of work.

The most important tips are as follows:

First, do a thorough job on your self assessment. This is not the place to do the bare minimum as doing so will certainly create extra work later. Being sloppy will usually require some portions that looked like they were completed, to be revisited. This is demoralizing because it seems then like the goalposts are getting further away, rather than closer. So, starting with a solid self assessment really helps everyone scope the work and have the right problems / thoughts in mind when working.

Second, chat informally with folks on the security assessment team as things come up. Most of us, while having other full time jobs, are happy to answer questions and guide the work in the right direction from the earliest stages. So, don't feel like you need to only check in or ask things when you have a huge deliverable to send. Ask focused questions if you need to and feel free to show intermediate work products.

Third, plan for the assessment to take a small to medium effort over a longer time period. A TSSA, like all security assessments, requires thought. It just isn't possible to cram one in over a two day sprint and feel confident in the result. (Just like writing a term paper at 4AM isn't going to be your best work.) So, start the process early, take breaks for feedback, and expect to iterate a bit as you and the team assessing your project gain greater understanding.

Finally, have some patience with the TAG-Security team doing the assessment. We are volunteering our time to help to try to make your project as secure as possible. This will lead to more adoption and ultimately improve it overall. So, please bear with us when we ask a "dumb question" which implies we don't understand part of your project or need a few days to read the 15 page or longer document you sent. We're all on the same team here!

Also, one more ask of you once you finish the assessment process is to help us assess another project by participating in the TSSA process as a reviewer, if possible. It usually takes 4-5 folks from TAG-Security to do an assessment. Having some participation from project members of projects we have assessed helps to scale this. Their past experience going through the process means they are often some of the best assessors as well!

Ash Narkar's commentary: The TAG-Security assessment process really helped the OPA team understand the overall health of the project from a security perspective. The assessment identified areas of the project that could be improved for example better documentation around secure deployment practices, enhancing OPA's toolchain usability to reduce policy authoring related errors. The OPA project benefited

from the recommendations and advice provided by the security experts at TAG-Security and our on-going relationship with TAG-Security helps us gain insights into the latest security best practices thereby allowing us to continuously improve OPA's security posture.

and make new friends with security counterparts, this is accomplished by the knowledge and wisdom exchanged in threat modeling. This a good reminder for establishing feedback loops to validate outside perspectives in a world where it's easy to be insular by being hyper-focused on the work at hand.

Andrés Vega's commentary: I was skeptical at first about what the assessment could provide in value. We knew our system well; the team I was part of had designed and built it. I perceived it as a nuisance but still went with it. Receiving an assessment required the project to progress through the motions of perceived maturity. I was quickly shown how mistaken I was.

An assessment will challenge your assumptions. What you think you know, what you take for granted, and prove opposite to everything you think can't go wrong. It will likely have you reassess aspects of system design and what to weigh when considering tradeoffs; it may even change how you build, test and deploy, particularly how your team is organized around those tasks, but ultimately will expand your own understanding leaving you better off and raising the quality, security, and safety of the work you produce.

The output from the analysis the SPIFFE and SPIRE projects underwent twice helped identify practical improvements to the security of SPIRE and other SPIFFE implementations. The publishing also acts as an educational resource for end users and implementers seeking to better understand from an architectural perspective and profile it from a defensive and offensive lens.

Assessments are a great way to team build, network,

The figure hows the high level process of TSSA, which can be described as follows:

1. The project should open an issue for a joint review by creating an issue in the TAG-Security GitHub
2. Upon receiving the joint review request, TAG-Security's members expressing their interest to participate as reviewers and makes the conflict of interest statement
3. The lead security reviewer will now perform an initial clarifying pass (also called the naive questions phase) of the project.
4. All reviewers perform a review at this point and try to collect their thoughts in three categories: clarifying questions, feedback for the project, and feedback for the TOC. The lead security reviewer or their designee, with the assistance of the security reviewers will create a draft summary document to capture existing comments, feedback, and recommendations.
5. The reviewers may optionally perform a hands-on review.
6. The reviewers and project team meet to discuss the joint assessment and clear up any disagreements. Once the review is complete, the project team presents the joint review to the TAG-Security community.

For a detailed view of the process, the key roles and the outcome, please refer to the official TAG Security Assessments Guide.

What if I disagree with part of a joint TSSA?

First and foremost, discuss disagreements with the assessment team. In almost all cases, disagreements can be resolved this way. If not, then adding the TSSA Facilitator may be useful.

If it cannot be resolved even then consider that intentionally, the TSSA is set up to have different "final writers" for different parts of the document. The self assessment document is the purview of the project. While someone from TAG-Security may disagree, you control that part of the document and can write as you please.

Conversely, TAG-Security controls the README.md of the document. So, in this place they can put their recommendations, thoughts, and counterpoint to a disagreement between the groups.

The best way to handle this is to stick to factual statements. Stating that "project X loses security property Y when Z happens" is a factual statement that hopefully both parties can agree on whereas statements like "project X is secure." or "Z is a huge problem" are more subjective and likely to inflame.

What happens when your TSSA becomes out of date?

Projects are rarely static. New features are added, new interfaces grow, subsystems diverge or converge, and on occasion technical debt is paid. When changes happen, it would be ideal to update the TSSA at the same time. Fortunately, unlike a security audit, security assessments persist in validity over a long period of time. Unfortunately, this means they are often relied on when portions are stale.

To update a TSSA, simply open an issue on TAG-Security and suggest the changes that should be made. The follow on process will be more lightweight because the new assessment can focus on the changed items.

How to volunteer in TAG-Security to do a TSSA

The process is designed to help people of different

levels of skill gain experience and perform an assessment together. Over time as people level up and get more experience and comfort, they are ready to take on added responsibility as part of the assessment process.

Justin Cappos's commentary: A natural question is what domain-specific technical skills you need to have to perform a joint assessment. Honestly, so long as you have a basic technical understanding in the domain (similar to someone getting started with the technology), that may be enough to be an observer on an assessment. Security acumen is much more important than domain knowledge when doing an assessment. In my experience teaching security to over a thousand students, security acumen is something anyone can learn. It is also something that some students just intuitively understand from the start. Without trying, you will have a hard time knowing if you have a natural knack for security or whether it will take a little work to learn, but you can definitely get there!

There are four levels of participation in a TSSA:

- Observer: This is someone who will attend the meetings, lurk in the slack channel for the assessment, and read the document. However, this person has no actual responsibilities as it relates to the assessment. Hence, they just watch the process without being asked to intervene. However, if they have a question or observation, they should voice it. The intent of providing no responsibilities isn't to silence the observer. It is to enable them to not feel pressured that they must speak or act for the assessment to be successful.
- Reviewer: A reviewer is an active party in the security assessment process. They will give feedback on documents, chat with the project members doing the assessment, attend meetings, and the like. They do the heavy lifting from the TAG-Security side in order to make sure that the assessment is rigorous, timely, and accurate.
- Lead reviewer: The lead for an TSSA is the person directly responsible from the TAG-Security side. They recruit reviewers and observers, will divide up the work amongst the reviewers, and act as an active reviewer themselves. In practice, this party is almost always a party that has been a reviewer on one or more assessments before.
- TSSA Facilitator: This is the party at TAG-Security who helps to prioritize the order of assessments, recruit a lead reviewer and other reviewers, ensure uniform quality of TSSA assessments, and similar tasks to manage the overall TSSA queue. The TSSA Facilitator is also the POC for interactions with the CNCF TAG and other external organizations.

So, someone who does not have a security background, may be most comfortable as an observer role at first. For one with some security experience, acting as a reviewer makes the most sense. Leads are usually chosen from top performing reviewers by the TSSA Facilitator.

How to use an assessment (from TAG-Security and anywhere else)

Suppose you are deciding whether to trust a security project, how does a security assessment help you do this?

Look carefully at the goals of the project and the scenario they evaluated the project in. Do the goals and scenarios match your use case? If not, you may have a serious problem because little of the rest of the analysis may be relevant in your case, circumstances,

or be palatable to your risk tolerance.

Read through the threat matrix and think how likely the compromise cases are in your scenario. Is there strong isolation between actors or is there a lot of lateral movement potential? Many distributed systems have a single point of failure (or many single points of failure) because they fail to compartmentalize trust adequately. If this is true in your case, the risk is very high.

Look at the security practices the project uses for bug disclosures, code review, testing, etc. This can give you an idea of the security emphasis and expertise of the group. Some things to look for are: the amount of test code coverage, the way in which code is admitted into the process, the way in which dependencies are vetted and kept up-to-date, and the amount of security expertise of the group. Also looking at past security audits (if they exist) can be very illuminating.

Justin Cappos's commentary: For test code coverage, a natural initial thought is that the closer to 100% you are, the better off the project is. I have even heard managers say that people should not ship code if it is below a specific threshold (95%, 99%, etc.).

My experience is that shooting for a fixed target across projects is too rigid. Some code is really unreasonable to test. For example, a project with a lot of error handling code that aborts during hardware failures, may be difficult to test in a good way. Different languages and frameworks can also make getting a high degree of test code coverage very challenging. In these cases you get almost all of the benefit from the first 9X% or so of test case coverage. Often the last bit of testing just isn't worth it compared to the other things you could be spending your time fixing and improving.

The code admission process for dependencies should at a minimum look at the sorts of aspects you would look at in any software project. How clean and well tested is the code? How quickly do the maintainers respond to issues? What sorts of issues are raised? Is security a priority? What is the release cadence? Is disclosure and documentation around fixes provided in a clear manner? How big and diverse is the contributor base? Do they seem to often take upstream contributions? Are those contributions vetted appropriately?

Does the project group demonstrate strong security expertise? Do they have a clearly stated threat model? Are the actors and actions for the project clear? Are actors appropriately compartmentalized? Have they described why their security mechanisms are in place and what they are supposed to protect against?

Has the project had a security audit? How did it go? Note that just counting the vulnerabilities discovered isn't necessarily a great metric as different auditors put different weight on the same issue. However, you should be concerned if a lot of serious issues were found, even if those were later addressed. Remember, since this is just one perspective from one moment in time, a project that fixes issues from an audit does not give a strong indication that further bugs do not exist now. So, be sure to check whether they have also improved their security practices.

Justin Cappos's commentary: There are a lot of red flags to look for when looking at a project:

A lack of threat modeling: I've spoken with engineers at major tech companies that were making security technology and asked them what the technology was

supposed to protect against. Multiple times, across different companies and products, I've had them say that they will figure the threat model out after they finish building it. This is a huge red flag because if you don't understand what you are trying to protect and from which attacks, how can you hope to do so? Fortunately, this is a small minority of people at these companies, but still it is a telling problem.

Viewing security mechanisms as features: Another major mistake I've seen made is that some developers seem to feel that adding a security mechanism is just a piece of functionality you can bolt on at the end. In other words, you could just add AES encryption to a system and then just say "I use AES" and then your system would be secure because AES (with reasonable key sizes) is thought to be secure today. Unfortunately, like putting a bike lock on a bicycle, it really matters where you put the lock. If you just attach it to the seat, the lock really isn't doing anything. So, security isn't something you can just defer until later. You need to design for security as early as possible.

A general lack of understanding of security by the security team. Occasionally, even in major companies, you will encounter folks that clearly misunderstand very basic concepts about security. While no one was born with deep security knowledge, a lack of understanding needs to be coupled with a desire to learn. Someone on the team needs a generally strong understanding of security. I've done security designs with domain experts in fields where I am not an expert (like automotive). As long as folks are willing to learn and discuss problems openly, this can work out well. Someone with a security skillset needs to be involved with design. If you are not overly comfortable with security and don't know how to judge the group's design(s), then ask other experts to take a look. If they have published the work in a reputable academic venue, then the academic peer review process is another way to ensure that review from other experts has happened.

Over-reliance on standards. Except for the excellent cryptographic standards from NIST which were done by a competitive process, most of the security designs through other organizations (ISO, IETF, IEEE), are historically very hit and miss. (Note, I am saying this as someone with projects that have been standardized through different organizations.) Anyone accepting standardization as a strong indicator of quality, does so at substantial risk. Many protocols and projects with a very poor security history (e.g., RFC 7519) were standardized through these organizations. Looking at the security analysis of a technology is a much stronger indicator.

Over-reliance on branding. Similar to over-reliance on standards, some people seem to feel that if it comes from company X, it will be great. While I've seen many excellent things come from top 5 tech companies, I've also seen them release some horribly insecure technologies. Every company has hits and misses. Don't be too quick to jump on the bandwagon. As before, peer review by experts is a much stronger indicator than a brand name.

Over-reliance on outdated technology that has known weaknesses. As security technologies age, the community gets more experience attacking them. It is important to understand the known limitations of technologies when designing new systems. For a first example, X.509 certificates parsing errors are commonly used by attackers to bypass security protections in a system. Using a format like X.509 or JWT leads to a lot of security issues due to the complexity of parsing [HackerNews: Do not use JWTs, Medium: Hacking JWT : Exploiting the "none"

algorithm, Exploit Database: wolfSSL 3.10.2 - x509 Certificate Text Parsing Off-by-One, SuSE: Security vulnerability: openssl 3 certificate parsing buffer overflow CVE-2022-3602]. As another example, technologies like revocation via OCSP and CRL also have huge drawbacks in practice [DarkReading: Solving The SSL Certificate-Revocation Checking Shortfall]. These technologies are not used in modern browsers [Chromium: CRLSets, Mozilla Firefox: Remove CRL User-Interface, Certificate Revocation in Microsoft Edge], despite being designed specifically for this use case! If a project doesn't seem to understand the security risks and limitations of outdated technology, then this should be a major concern.

Over-reliance on reinventing the wheel. Some developers just like to invent things and don't want to use tools or techniques from elsewhere. This is particularly problematic in security. You should be using the designs that have held up to stringent review unless there is a clear reason to create something new. Reinvention raises new opportunities to make mistakes and introduce vulnerabilities.

A lack of awareness of other technologies. A major indicator that a project doesn't understand security is a lack of awareness of the pros and cons of other solutions in the space. If someone does not seem to understand what the differences are between competing approaches, they likely do not understand the threat landscape or problem domain either. While closed source, commercial products can be difficult for outsiders to understand, for well documented, public systems, you should be able to get a clear description of the differences.

How to use a TAG-Security Assessment

TAG-Security Assessments are structured in a way that is intended to make them easier for an adopter to read. Most importantly, strongly consider the recommendations in the README listed by TAG-Security. These will point out the points that TAG-Security's reviewers believe are most concerning from a security standpoint.

The project's self assessment is also valuable because it describes how the project sees itself and its security posture. Consider carefully the scope of their threat model. Is it better for a logging system to have a flawed system which sometimes fails to sanitize personal information or to never try to sanitize it at all? The answer could go either way. Counterintuitively to some people, if users of the logging system expect its sanitization to be perfect and then use it in situations where they otherwise wouldn't, its existence can certainly harm security.

Here are some example assessments and some quick notes a reader would likely take from them. **Note, that these reflect the assessment at the time it was completed. The project may have addressed the described issues in the interim.**

Harbor: Users need to still do a review of the resulting security properties given the way they deploy it. The mechanisms are all there for strong security, but there are concerns that it is not being deployed in that manner by some adopters

In-toto: In-toto provides software supply chain security by validating cryptographically protected metadata about the process. At the time of assessment, the documentation around getting started, usability, and best practices were thought to be inadequate for some adopters.

SPIFFE / SPIRE: This project handles secure provisioning and management of key materials in heterogenous cloud and on prem environments. The README is very positive and only lists some minor TODOs for the team in different areas. Their assessment indicates they have a single (trusted) server and that there are some risks with their agent implementation. However, the project seems to have a good understanding of this and has worked to minimize the risk from these components.

CONCLUDING THOUGHTS

CONCLUDING THOUGHTS

Congratulations on making it through this book. So now, dear reader, you are equipped to understand and assess the security of complex software projects. We hope that you will use your skills to improve the security of the cloud, software projects, or even brick-and-mortar institutions like TrashPanda Bank.

Everyone is relying on your efforts to make the world a safer place!