CHAPTER

1

Background on Software Supply Chain Threats

This chapter outlines core topics such as the incentives for attackers, anatomy of a software supply chain attack, and relevant landmark cases. Let's begin by discussing the incentives for attackers to perform a supply chain attack.

Incentives for the Attacker

Supply chain attacks circumvent traditional perimeter defenses in ways that make them very attractive for the attacker. Organizations have invested heavily in firewalls, intrusion prevention, and access controls. These protections are defensive measures against a "push" style of attack that directly targets an organization's infrastructure. Supply chain attacks foster a scenario that is more of a "pull," where legitimate users of information technology (IT) request software updates that are malicious, causing the user to knowingly compromise their organization. Because the request originated from a trusted user and came from inside the corporate perimeter or was sent to a trusted entity already cleared by a third-party risk management process, these updates were trusted. Organizations simply compromise themselves.

When you are exploring the controls necessary to defend against attacks, it is not sufficient to consider a single layer as an effective defensive measure. In much the same way that network infrastructure administrators have realized they need to monitor outbound traffic or implement host-based controls, this

move toward defense in depth, and especially looking beyond the perimeter, is crucial. For a variety of reasons, including cloud, mobile, social media, and modern application infrastructures, the concept of the perimeter must also evolve. It is no longer a static boundary but needs to be thought of as the separation layer between trust zones, whether they reside at your network's edge or as a logical barrier within your application or access control mechanism. As such, our controls and resultant threat modeling must consider the entire attack surface and explore each interaction point and trust relationship.

Supply chain attacks function as a force multiplier as well. By identifying key dependencies or widely used software, an attacker can inject malicious code into any environment that then utilizes that code. This has been seen time and time again and is similar to the concept of a watering-hole-style attack, where an attacker compromises a widely used website used by a target group, such as a programmable logic controller (PLC) web forum used by industrial control systems (ICS) integrators. If an attacker can compromise every user who visits the forum, they can theoretically gain access to any critical infrastructure entity that those integrators do work for. Likewise, any software used by those integrators that has been compromised may introduce unwanted and malicious functionality into the environments this software is installed in, even long after those consultants have moved on to their next project.

All of this means that software supply chain attacks are highly lucrative for the attacker. There is an economic factor to cyber-espionage that greatly benefits from reusable exploits and the low entry cost for supply chain attacks. Also, many recent attacks are seeing a combination of ransomware deployed via software supply chain, which not only makes for ease of compromise but a direct financial gain for ransomware operators and an increasing level of concern for organizations concerned with business interruption.

Threat Models

The topic of threat modeling is one that is frequently cited and, in the authors' experience, rarely performed in practice by most organizations. As an industry, we frequently discuss threats in fairly generic ways without ever exploring the context of the software or systems necessary to properly model those threats. At the core of these activities, however, is a definition of the attack surface—the points of interaction—and an exploration of what could go wrong.

To ensure that we provide maximum clarity, we'll define a few key terms:

Threat A threat is a negative event that creates an undesirable outcome. Threats may be naturally occurring, benign in nature, or overtly malicious. An example might be that the billing system for your business fails to capture transactional entries or the datacenter is destroyed by a tornado.

Threat Agent A threat agent is the entity responsible for causing the threat to occur. Examples include hacktivists, malicious insiders, a disgruntled business partner, a consultant who has been compromised, or even a weather pattern. It has been noted that threat attribution is frequently wrong, and assumptions made about an attack pattern's specific threat agents might lead to wrongful assumptions in protection characteristics. For instance, if you think a nation-state prone to ransomware is a likely threat agent, but you are instead faced with one more focused on espionage, how does this change your security posture? Does the actual country of origin matter, or is it creating undue bias?

Threat Action This is the action taken by the threat agent that ultimately causes the threat to occur. For instance, a threat agent such as a hacktivist might bribe your system administrator to reconfigure the billing system, resulting in a threat that ultimately creates a financial impact.

Threat Model This is the process of documenting systems and threats in such a way that we can model certain types of decisions related to risk management for the system because of a threat. There are different objectives for threat modeling, including general cyber risk management, systems design and analysis, and even information-sharing models.

Threat Modeling Methodologies

There are several types of threat modeling methodologies. Let's begin by discussing STRIDE.

Stride

STRIDE is a very common threat methodology used to help determine what can go wrong, although it is not the only methodology in use. STRIDE is broken up into a mnemonic aligned with the acronym:

Spoofing: Impersonating another user or system component to obtain its access to the system

Tampering: Altering the system or data in some way that makes it less useful to the intended users

Repudiation: Plausibly denying actions taken under a given user or process

Information Disclosure: Releasing information to unauthorized parties (e.g., a data breach)

Denial of Service: Making the system unavailable to the intended users

Elevation of Privilege: Granting a user or process additional access to the system without authorization

Most supply chain attacks are a result of operating a system, and traditional models such as STRIDE are designed for more direct attacks against a system, as opposed to the indirect impacts that occur when an administrator requests an update to their application, previously known as good, which then winds up being malicious.

Stride-LM

Researchers at Lockheed Martin have expanded the STRIDE methodology by adding a seventh dimension known as *lateral movement*. Although STRIDE is useful for systems design, it does not meet the needs of network defenders. STRIDE-LM provides a mechanism for defenders to design controls that allow for greater efficacy in defending beyond the point of initial compromise.

When evaluating threat models for applicability to software supply chain attacks, ask yourself what the model was designed for and how it will apply to the scenario in question. For instance, many supply chain attacks abuse maintenance phases through malicious updates or abuse of trust that circumvents controls designed to detect an initial software entry point. Likewise, due to the single point of failure inherent in software supply chains, the downstream impacts of these actions might not be effectively modeled in the original STRIDE methodology. As we explore the landmark cases in this book, you will begin to see how the use of lateral movement provides additional context to modeling and makes STRIDE-LM far more applicable to supply-chain-oriented attacks.

Open Worldwide Application Security Project (OWASP) Risk-Rating Methodology

The OWASP methodology is used as a quantitative scoring model to evaluate specific risks by leveraging technical threat and business impact criteria to derive the score. At its core, it uses a fairly standard impact × likelihood calculation, but where it gets interesting is how those two sides of the equation are created. The following description attempts to capture the basis of this formula, but it should be noted that in practical usage it is not uncommon for the factors to be changed or for weighting to be applied to specific factors that are very important. For instance, in critical infrastructure, we find many entities desire to add a fifth "safety" factor to the business impacts and will often weight it heavier than the others.

$$\textbf{Likelihood} = \textbf{AVG}\left(\textbf{Threat Agent} + \textbf{Vulnerability}\right)$$

where Threat Agent = skill, motive, opportunity, and size; and Vulnerability = ease of discovery, ease of exploit, awareness, and intrusion detection.

$$\textbf{Impact} = \textbf{AVG}\left(\textbf{Technical Impact} + \textbf{Business Impact}\right)$$

where Technical Impact = loss of confidentiality, integrity, availability, and accountability; and Business Impact = financial damage, reputation damage, noncompliance, and privacy.

$$\textbf{Risk Score} = \textbf{Likelihood} \times \textbf{Impact}$$

OWASP is useful for evaluating an identified risk and prioritizing it for actioning, but it is less useful as a threat modeling technique to identify unknown risks. It might make sense to work OWASP into your process *after* other threat modeling techniques have been applied. In our experience, it is far superior to prioritization mechanisms such as Common Vulnerability Scoring System (CVSS) scoring, but it does require context to be useful.

DREAD

DREAD was a legacy technique created by Microsoft, but we rarely see it used today and it is often considered to be a "dead" methodology. DREAD utilized a mnemonic similar to STRIDE's:

Damage: How bad would an attack be?

Reproducibility: How easy is it to reproduce the attack?

Exploitability: How much work is it to launch the attack?

Affected users: How many people will be impacted?

Discoverability: How easy is it to discover the threat?

Using Attack Trees

Attack trees (see `http://schneier.com/academic/archives/1999/12/attack_trees.html`) are a visual way to work backward from an unintended or undesirable consequence (so that you can understand how that event could occur and the most likely vectors of attack), and a way to create a prioritized list of controls to implement to prevent that consequence. In Figure 1.1, you can see the cheapest path for the adversary is to just cut open the safe. It might make sense to prioritize physical security controls to prevent access to the safe in the first place or to spend more money on a safe resistant to such attacks. While internal threats are a viable attack vector, the economics of the attacker make it far less likely they will choose this approach.

This can be an effective way to begin to understand a supply chain attack and to see how easy it is to carry out as opposed to more conventional attacks. With conventional attacks, half of the battle is gaining physical access to even attempt to bypass access controls. In a supply chain attack, a trusted entity is engaging in the actions necessary to compromise the system. However, there

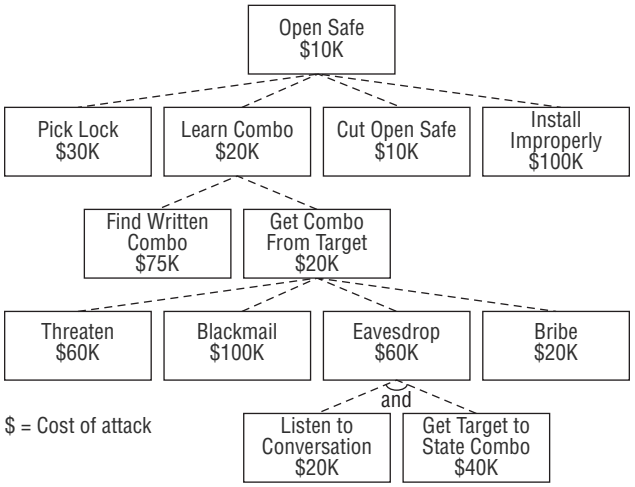

Figure 1.1

are far fewer barriers to pass through. Let's look at a sample attack in Figure 1.2, using a typosquatting attack, which is when attackers offer up malicious packages with names similar to real packages to masquerade a malicious component as a legitimate library on GitHub. This is by no means a complete example.

To put this in perspective, think about the steps required to execute an attack. Lockheed Martin developed a methodology referred to as the Cyber Kill Chain (`http://lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html`), which describes the phases of attack. There are many nuances, but these are the basic steps in the adversary's approach (see Figure 1.3).

What if the adversary can skip most of these steps and go straight to Command & Control? It is extremely attractive to an adversary to reduce the cost and complexity of carrying out an attack, and by understanding the pathways they can accomplish those objectives.

Threat Modeling Process

In this section, we'll describe a typical threat modeling process. First, you must define the system or application you are building or updating. Determine its components and how these components interact with each other. This begins laying the groundwork to identify the attack surface for the system. Perhaps it is an externally consumable application programming interface (API) or Hypertext Transfer Protocol (HTTP) service. Maybe there are other dependencies, such as a middleware or database server that needs to communicate to execute application logic or authentication services that utilize federation, but the need for core architectural understanding is the foundation for starting this process.

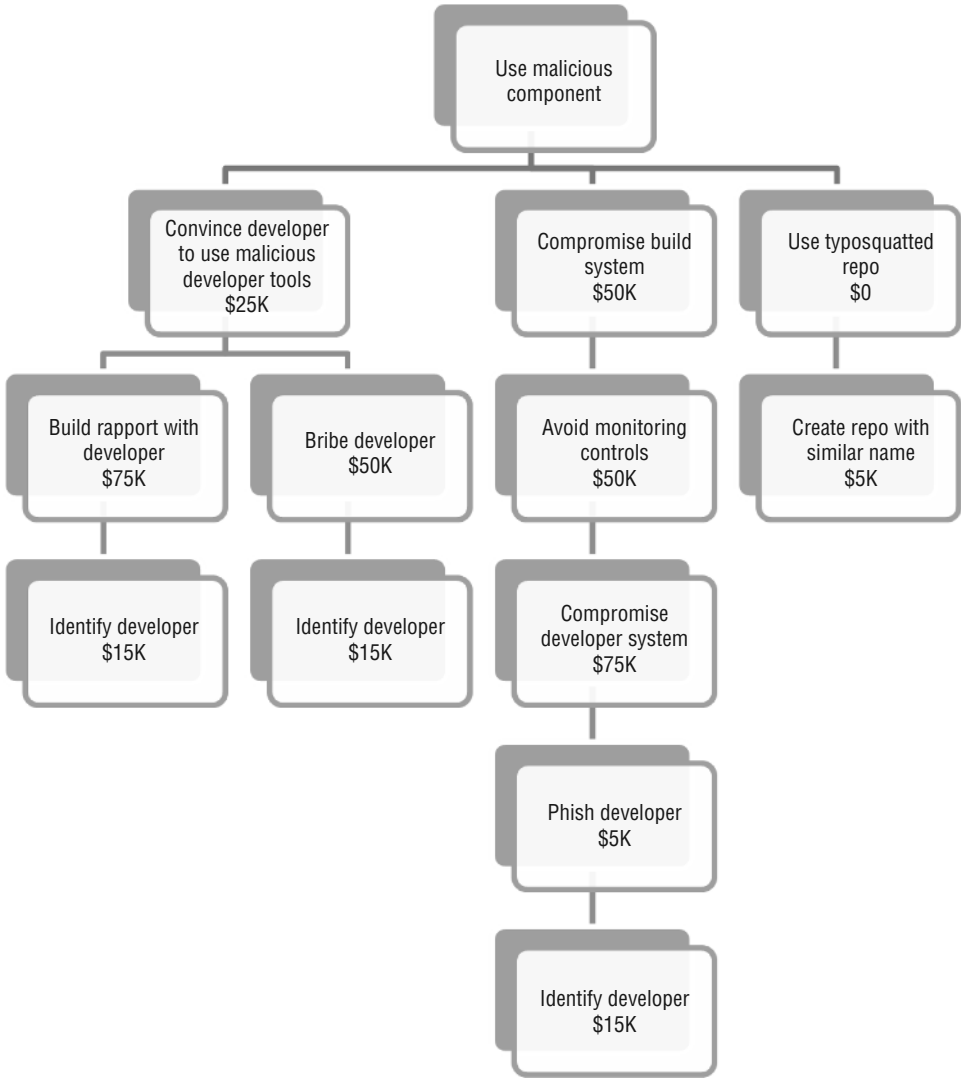

Figure 1.2

Figure 1.3

The OWASP Application Security Verification Standard (ASVS; `http://owasp.org/www-project-application-security-verification-standard`) defines this basic architectural understanding as a foundational requirement, including the need to conduct threat modeling in design and future change cycles.

The OWASP Software Component Verification Standard (SCVS; `http://owasp.org/www-project-software-component-verification-standard`) extends this concept even further by calling for a software bill of materials, which we will explore further later in this book. It is clear, however, that a firm grasp of what needs to be protected is necessary to understand threats and how to defend against them.

Many tools can be used to document your system; one commonly referenced is the Microsoft Threat Modeling Tool. For many years it was the only viable option, but this space has evolved quite a bit over the last few years. The Threat Modeling Tool allows software architects to identify threats early in the development process, before it becomes too costly to remediate them. It uses the STRIDE model to document threats.

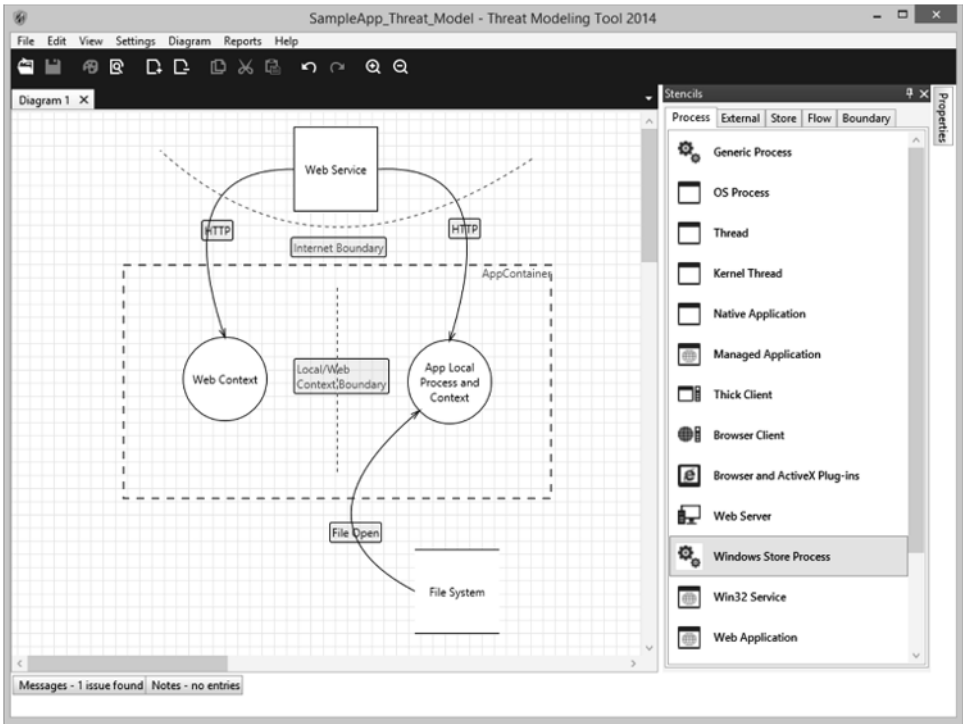

Figure 1.4

In Figure 1.4, you can see a simple application container, which separates the contexts of Internet access to an exposed web service and local access down to a filesystem level that compartmentalizes these application constructs using trust boundaries. By dissecting your system in this way, you can determine how a potential threat agent might interact with your system and can begin

to answer the questions in the second part of this process. There are other free tools, such as pytm from OWASP, which is a Python-based way to automate threat modeling using Common Attack Pattern Enumeration and Classification (CAPEC) definitions, which we will explore later in this chapter, as well as Threat Dragon, which is more similar to the Microsoft tool. There are also very robust commercial tools available, such as Threat Modeler and IriusRisk, which also embrace the previous concepts in ASVS design principles.

The next step in the threat modeling process is to determine what could go wrong. This might be as simple as an error condition or a user who has not been trained to use the application correctly, or it can be as significant as an adversary seeking to cause harm. For our purposes, we will focus on malicious cyber sabotage scenarios to explore this topic.

To illustrate a potential software supply chain threat scenario, consider Figure 1.4, where a malicious actor has injected malicious log entries into the web service, which, when consumed locally and opened by a log viewer, executes a malicious JavaScript in the user's browser. If this web service is consumed by many downstream users, that one log injection attack could result in many downstream users who rely on this service to all become compromised. It then creates a scenario where the adversary has compromised a single web service, uses it to compromise many other organizations, and has now established a beachhead to conduct further attacks against those organizations or their downstream customers who trust them as part of their supply chains. Such a data-tampering attack can have tremendous downstream impacts.

Once you know what can go wrong, you then need to understand what controls can be implemented to minimize the threat. Part of this stage will likely include a prioritization exercise, as it might not be possible to eliminate every potential threat; however, focusing on the most impactful scenarios that also have a reasonable chance of success might help guide the control conversation.

For instance, in the previously described scenario, perhaps sanitizing log inputs or escaping characters might result in additional protections against this attack. Because there are many ways attackers can bypass traditional protections, such as those implemented to prevent cross-site scripting (XSS), it might be beneficial to look at similar protections, such as those documented in the OWASP XSS Prevention Cheat Sheet, which provides guidance to prevent XSS vulnerabilities. The cheat sheet is a list of techniques to prevent or limit the impact of XSS attacks (`http://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html`).

The final stage of threat modeling typically involves evaluating the threat model's completeness and asking yourself if the diagram is complete enough to run through all your scenarios and if it clearly identifies trust boundaries. First, determine if you have covered all the threat actions defined in the model you are using, such as STRIDE, and if all the data flows have been explored. Lastly, determine if all identified threats have had a risk outcome or plan of

action established, even if it has been determined to be non-feasible to worry about whether the mitigations required are outside of your control and cannot be acted on. You should close the loop on the model and revalidate the model every time major architectural changes are introduced into the system.

Additionally, we find that the following concepts play heavily in the topic of software supply chain threats, and as such, will take a moment to define them:

Provenance Provenance is the concept of where an artifact came from. It is commonly confused with pedigree (see the next item). Provenance, as applied to software, might include where a specific software component came from or who contributed code to a specific library. It is uncommon for organizations to track at the individual contributor level, but for high-security scenarios this may be a worthwhile endeavor. We have seen through experience that almost every widely used open source library will have contributions from countries of origin that might be thought of as adversarial, so for organizations that choose to go to this level of detail, additional mitigating controls might be called for or deeper security analysis might be needed to determine if these contributions are problematic.

Pedigree The concept of "track and trace" is not new, and in fact, is a core concept in forensics sciences and closely related to the application of a "chain of custody" log that identifies anyone who touched an artifact from creation through delivery. Tracking indicates the movement of the artifact, and chain of custody attributes the individual or organization who touched the artifact. The concept of secure equipment delivery has likewise emerged to track how physical assets transit the globe and includes the use of unique identifiers such as a Lot ID and Global Trade Identifiers via the GS1 family of standards and protections such as physical unclonable functions (PUFs). As we are primarily describing a digital artifact as it relates to software transparency, the use of cryptography and transparency logs satisfy these requirements. Pedigree, as it relates to software transparency, might indicate the history of a software package or a security patch for that piece of software, and it is a concept that can be documented in some software bill of materials formats. Likewise, pedigree can also track where a point of origin has changed over time, such as a GitHub repository that has changed from one maintainer to a new maintainer, or a supplier who has divested a product line and sold it to a third party. These are concepts we will explore further later in this book.

Many threat modeling frameworks such as DREAD and STRIDE are used today for various purposes. Such frameworks are commonly used to describe security design and analysis. The Consequence-Driven Cyber-Informed Engineering (CCE) framework from Idaho National Labs (INL) focuses on controlling threats aligned to specific business consequences and not necessarily every point in an attack surface. Then there are frameworks like STIX and PRE-ATT&CK, which are designed for information sharing and other preparatory phases.

Frameworks such as CAPEC and MITRE ATT&CK have a specific technology or situational focus. Organization-specific frameworks include the NIPRNet/SIPRNet Cyber Security Architecture Review.

Likewise, one key focus of threat modeling at scale tends to be a focus on attack patterns. The reason organizations do not perform threat modeling correctly is because it's difficult to do at scale. The Microsoft Security Development Lifecycle (SDL) lists threat modeling as activity #4 out of 12—just after defining security requirements and key performance indicators (KPIs) and just before defining design. This makes a lot of sense when designing a new system, but how do we threat-model hundreds or thousands of applications across our environment at scale? Typically, we wind up taking shortcuts. This is why attack patterns are so popular, especially if we can reduce the number of attack patterns based on identified adversaries or identified target groups for those adversaries.

CAPEC is a publicly available catalog of attack descriptions hosted by MITRE and originally created by the Department of Homeland Security. This catalog includes an entry for *supply chain* (`http://capec.mitre.org/data/definitions/437.html`), which helps defenders understand how these attacks are constructed in order to better identify defensive measures. This CAPEC entry for supply chain attacks describes this category of attacks as captured below

> **Attack patterns within this category focus on the disruption of the supply chain lifecycle by manipulating computer system hardware, software, or services for the purpose of espionage, theft of critical data or technology, or the disruption of mission-critical operations or infrastructure. Supply chain operations are usually multi-national with parts, components, assembly, and delivery occurring across multiple countries offering an attacker multiple points for disruption.**

Also included in this family of attacks patterns are the following:

- Excavation
- Software integrity attacks
- Modification during manufacture
- Manipulation during distribution
- Hardware integrity attacks

Additionally, references to MITRE ATT&CK can also be helpful, such as the *Supply Chain Compromise* entry (`http://attack.mitre.org/techniques/T1195`), which also captures three additional sub-techniques as well as some mitigation that might be helpful for defenders:

- Compromise software dependencies and development tools
- Compromise software supply chain
- Compromise hardware supply chain

Unfortunately, their coverage of supply chain is fairly basic at this stage, and it might make more sense to combine this approach with more specialized work, such as that of the Cloud Native Computing Foundation (CNCF), that might provide far more of the specificity needed to defend against these threats.

CNCF's catalog of software supply chain compromises also includes an index (`https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/compromise-definitions.md`) defining several types of attacks. We will discuss those attack types in order to have a shared lexicon of some of the primary attack vectors in the software supply chain. The index of attack types includes:

- Development tooling
- Negligence
- Publishing infrastructure
- Source code
- Trust and signing
- Malicious maintainer
- Attack chaining

It is worth noting that several attack types map to existing frameworks such as MITRE's ATT&CK, which has sections (`http://attack.mitre.org/techniques/T1195`) dedicated to supply chain compromise, including software dependencies, development tooling, and the software supply chain itself.

Developer tooling attacks are a compromise of the tools used to facilitate software development. This might be the developers' end device, software development toolkits (SDKs), and toolchains. MITRE recommends using integrity-checking mechanisms such as validation and scanning downloads for malicious signatures. If a malicious actor can compromise development tooling, they are able to introduce potentially malicious code from the onset of the software development life cycle (SDLC) and tarnish all subsequent application development activities and consumers.

Negligence is a failure to adhere to best practices, which is common, given that there are so many applications security best practices to know and that we live in an increasingly complex digital ecosystem. Something as simple as neglecting to verify the dependency name can have a major impact on an organization. Malicious actors have increasingly been using an attack method known as *typosquatting*, which, as mentioned earlier, takes advantage of lack of attention to detail of dependency names. Attackers typically will target a popular framework or library, add their malicious code under a name similar to the original library, and then wait for unsuspecting victims to download and use it in their applications.

Publishing infrastructure has become increasingly critical as organizations now commonly utilize continuous integration/continuous delivery (CI/CD) pipelines and platforms to deliver software artifacts. One mitigation technique is code signing, which helps ensure the integrity of the published code. However, as you will see in our first landmark case, a compromise of the CI/CD infrastructure itself can allow malicious actors to legitimately sign software artifacts that present them as trusted to downstream consumers. This attack method can be devastating and nefarious, and it emphasizes why the publishing infrastructure must be secured as production environments are and that they must align with emerging frameworks like Supply Chain Levels for Software Artifacts (SLSA), which we will touch on in upcoming chapters.

Source code attacks involve the compromise of a source code repository, either directly from a developer or through the compromise of a developer's credentials. The 2022 Verizon Data Breach Investigations Report (DBIR; `http://verizon.com/business/resources/reports/dbir`) found that credential compromise was involved in over 80 percent of data breaches. This included situations impacting source code or source code repositories. Malicious actors targeting source code and repositories will often try to introduce vulnerabilities or backdoors into the source in order to later exploit them or impact downstream consumers.

Integrity is critical to trust in the software supply chain. This is typically facilitated by activities such as digital signing and attestations. Signing code gives downstream consumers a level of assurance regarding the provenance of code and its integrity. That said, compromising signing can enact software supply chain attacks. Therefore, fundamental security concepts like defense in depth remain key. *Defense in depth* is a long-standing cybersecurity practice of using multilayered defenses so that no single vulnerability or weakness leads to an entire system or organizational compromise. Malicious actors need to exploit several layers of defenses and security measures to achieve their objectives, rather than a single vulnerability or weakness. In the previous example, using only digital signing is insufficient and can be exploited. This is among the attack types listed in the CNCF index and cited by MITRE in their supply chain compromise section of the ATT&CK framework. Potential threats include activities such as theft or private keys, misplaced trust, and abuse of certificate issuance, among others, as cited by the National Institute of Standards and Technology (NIST) in their "Security Considerations for Code Signing" white paper (`http://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/01/26/security-considerations-for-code-signing/final/documents/security-considerations-for-code-signing.pdf`). Mitigation techniques involve such activities as establishing trusted users, separating roles, using strong cryptography, and protecting the signing keys.

Given the complexity of the software supply chain and the often voluntary nature of maintainer activity, not all threats originate from outside a project either. The malicious maintainer attack vector is among the threats listed in

the CNCF index and involves a maintainer, or someone posing as one, deliberately injecting malicious software or vulnerabilities into the supply chain or source code. This can occur due to a maintainer deciding to act maliciously or because their account or credentials has been compromised by an external entity. Motivations for these sorts of attacks range from hacktivists to those posing as involved maintainers who only abuse their permissions and access for nefarious purposes.

Lastly, attacks and vulnerabilities don't occur only in isolation. Malicious actors may chain together several vulnerabilities and attack vectors to carry out their activities and desired intent. Looking at some of the attack types we've mentioned, a hypothetical scenario could involve a malicious actor posing as an interested contributor or maintainer only to abuse their access, insert malicious code, abuse signing, and so on. Stringing together several attack vectors can have a devastating impact and is often referred to as attack chaining, which has been researched and spoken about by researchers such as Dr. Nikki Robinson.

There are also other emerging methodologies and frameworks that can secure the software supply chain and threat-model ways it can be exploited by malicious actors, such as the Supply Chain Levels for Software Artifacts (SLSA), which we will be discussing in depth in upcoming chapters.

Landmark Case 1: SolarWinds

No discussion of modern software supply chain security incidents would be complete without discussing the SolarWinds cyberattack. SolarWinds is one of the largest digital system management tool providers in the market. In 2019, SolarWinds began experiencing cyberattacks that have now been attributed to Russian Foreign Intelligence Services. At the time of the attack, SolarWinds had 300,000 clients, including many U.S. government agencies and most of the Fortune 500 companies as well. It was estimated that of those 300,000 clients, 18,000, including federal government agencies and customers, had received a compromised software update. It is said that the malicious actors then specifically targeted a subset of the compromised clients whom they deemed high-value targets.

While not very much was clear in the initial stages of the attack, there have now been many post-incident reports that have shed light on both the technical sophistication of the malicious actors and the downstream impacts across the ecosystem of affected organizations in both the public and private sectors. One of those reports, created by the U.S. Government Accountability Office (GAO), has called the cybersecurity breach of SolarWinds "one of the most widespread and sophisticated hacking campaigns ever conducted against the federal government and private sector" (`http://gao.gov/blog/solarwinds-cyberattack-demands-significant-federal-and-private-sector-response-infographic`).

The GAO also produced the report shown in Figure 1.5 (`http://gao.gov/products/gao-22-104746`) to help communicate the high-level timeline of activities associated with the SolarWinds attack.

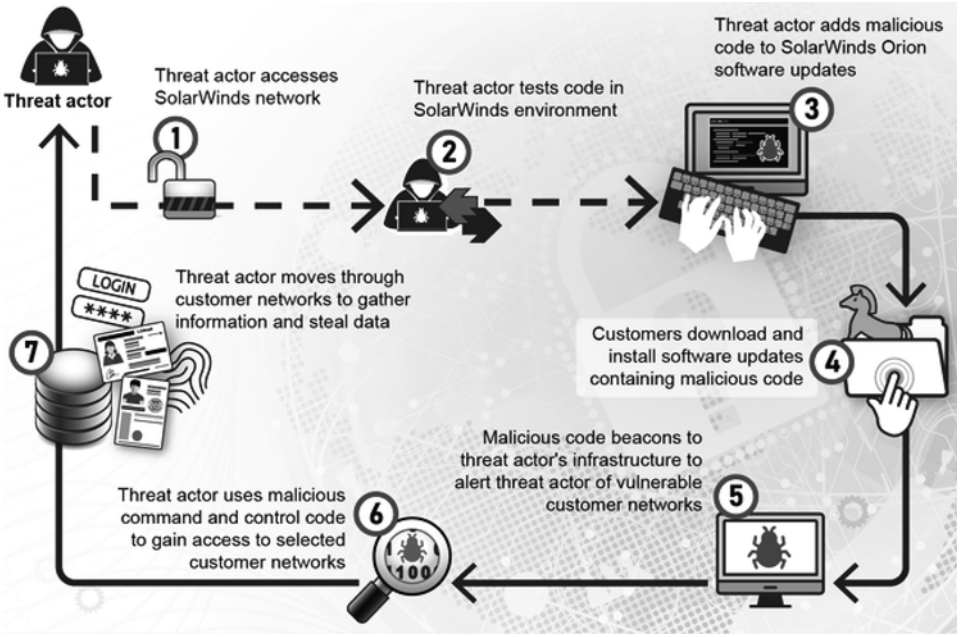

Figure 1.5

SolarWinds made an enticing target and quintessential software supply chain attack because it is unlikely that the malicious actors were interested in SolarWinds as the end target but instead were interested in SolarWinds' customers and downstream consumers. Prior to this cybersecurity attack, SolarWinds boasted their robust and high-profile customer list on their website, a list which has subsequently been removed.

The SolarWinds cybersecurity attack was initially identified not by the organization itself but by one of its customers, FireEye, which just happens to specialize in cybersecurity. It is reported that a FireEye employee was prompted to reset their multifactor authentication settings, which prompted the employee to bring it to their team's attention. This subsequently led to FireEye investigating the incident and tracing it back to malicious software from SolarWinds, specifically their Orion software. The chief technology officer (CTO) of Mandiant, FireEye's incident response firm, is quoted as saying that the organization went through 50,000 lines of code to determine there was a backdoor in SolarWinds. Once FireEye identified the compromised software from SolarWinds, they contacted both the vendor and law enforcement to notify them of their findings.

While some details of the attack timeline vary depending on the source, there are some fundamental facts that have been provided by the same GAO report previously mentioned. The timeline can be viewed from both the private sector activity and public sector engagement once federal law enforcement became aware and federal agencies such as the FBI and Cybersecurity Infrastructure Security Agency (CISA) became involved.

From a timeline perspective, it is said that in September 2019 the SolarWinds internal systems were initially compromised. That said, SolarWinds CEO Sudhakar Ramakrishna stated at the RSA Conference in 2021 (`www.cyberscoop.com/solarwinds-ceo-reveals-much-earlier-hack-timeline-regrets-company-blaming-intern`) that initial reconnaissance activities can now be traced back to as early as January 2019. Around the same time of the initial system compromise in September 2019, the malicious actor also injected test code into the SolarWinds environment to validate they had indeed compromised the systems and could carry out their intended activities.

The malicious actors used malware dubbed "Sunspot" to compromise the SolarWinds software development process and build systems. They then injected a backdoor now called "Sunburst" into the Orion product from SolarWinds around February 2020. In March 2020, a hotfix was made available to customers and subsequently downloaded by 18,000 of the SolarWinds customer base. In June 2020, the threat actor removed the malware it had placed on the SolarWinds build machines. It was not until December 2020 that SolarWinds was notified of Sunburst. SolarWinds then filed an 8-K report (`http://sec.report/Document/0001628280-20-017451`) with the Securities and Exchange Commission (SEC), which provided insight into the situation to the extent that SolarWinds knew at the time.

On December 15, 2020, SolarWinds issued a software fix to address the impacted Orion products. This, of course, presented an interesting conundrum, given that the previous software updates from the vendor are what contained the malicious software and affected customers to begin with. CISA issued an alert (AA20-352A; `http://cisa.gov/uscert/ncas/alerts/aa20-352a`) on December 17, 2020, warning of an advanced persistent threat (APT) compromising government agencies, critical infrastructure, and the private sector as well. This alert identified that the compromise of a dynamic link library (DLL) in five different versions had been affected. The alert also specifically called out the threat actors' patience, complexity, and the overall elevated level of tradecraft associated with the attack. Additional findings related to Sunspot were found in January 2021 as well.

Cybersecurity company CrowdStrike published a detailed technical analysis (`http://crowdstrike.com/blog/sunspot-malware-technical-analysis`) of the SolarWinds cyberattack. It found that the malicious actor used the malware known as Sunspot to compromise the SolarWinds build process and to insert

their Sunburst backdoor into the SolarWinds Orion IT management product. Sunspot worked by monitoring processes involved in compiling the Orion product and replaced one of the original source files with the Sunburst backdoor code. This backdoor then facilitated the creation of a reverse shell for the malicious actor to access the compromised victim's infrastructure who ran the impacted versions of SolarWinds' Orion.

There is much more that could be written about the SolarWinds cyberattack and its associated malware and backdoors, an example of one of the largest and most sophisticated software supply chain attacks to date. Several federal and commercial technology leaders have called the SolarWinds cyberattack a wake-up call and emphasized the need for increased rigor around how the industry secures the software supply chain.

Part of what made the SolarWinds cyberattack so nefarious was that the malicious actors took advantage of a long-standing best practice in cybersecurity, which is to ensure that you patch your software when updates are made available from vendors. In this case, the update was poisoned, so those following that best practice and updating/patching in a timely fashion were potentially impacted. By compromising the build process, the malicious actors were also able to pass the compromised software off as being signed and legitimate.

Since the fallout of the SolarWinds incident, the firm has made substantial investments in its software build processes and capabilities. Some reports and comments, such as by SolarWinds CISO Tim Brown, have claimed that their revised "Secure by Design" initiative will cost as much as $20 million annually (`http://cybersecuritydive.com/news/solarwinds-1-year-later-cyber-attack-orion/610990`). SolarWinds has participated in webcasts titled "Securing the Software Development Build Environment" (`http://cybersecuritydive.com/news/solarwinds-software-build-reproducible-cyberattack-code/596850`), where they discuss their Secure by Design approach, which includes using reproducible builds to weed out disparities in binary code. This involves changes with how the organization manages versioning and dependencies and consists of having multiple environments that a malicious actor would need to compromise to successfully compromise the organization's code. Reproducible builds (`http://reproducible-builds.org`) are cited in frameworks such as SLSA, specifically in their highest level of maturity and rigor, SLSA Level 4 (`http://slsa.dev/spec/faq`). We'll discuss reproducible builds in the following sections, but it is worth stressing that it is a mature implementation likely to be adopted by major software producers initially and, as noted by the SolarWinds CISO, not a cheap or light undertaking.

Landmark Case 2: Log4j

While the SolarWinds cyberattack was specific to a software vendor, the Log4j incident is much different in the sense that it targeted a widely used piece of open source software (OSS). The Log4j incident began receiving public attention on December 9, 2021, when security researchers discovered a flaw in the popular software library Log4j. Log4j is a Java-based logging utility that is part of the Apache Logging Services, a project of the Apache Software Foundation. Log4j, at the time, was primarily used for logging information related to debugging and other activities to help developers. At the time of the incident, Log4j was estimated to be present in over 100 million environments and applications.

On December 10, 2021, NIST's National Vulnerability Database (NVD) categorized the Log4j vulnerability as a 10.0 on their Common Vulnerability Scoring System (CVSS) and associated it with Common Vulnerability and Exposures (CVE) CVE-2021-44228. There were also subsequent CVEs published associated with Log4j that included not just remote code execution vulnerabilities, but denial-of-service CVEs as well.

Soon after the publication of the zero-day vulnerability, the Computer Emergency Response Team (CERT) of New Zealand warned that it was already being exploited in the wild (`http://cert.govt.nz/it-specialists/advisories/log4j-rce-0-day-actively-exploited`). Following this, the CISA issued an emergency directive (`http://cisa.gov/news/2021/12/17/cisa-issues-emergency-directive-requiring-federal-agencies-mitigate-apache-log4j`) requiring federal agencies to mitigate Apache Log4j vulnerabilities. Soon after, on December 22, 2021, the CISA, FBI, NSA, and international partners issued a joint advisory (`http://cisa.gov/news/2021/12/22/cisa-fbi-nsa-and-international-partners-issue-advisory-mitigate-apache-log4j`) to mitigate the Apache Log4j vulnerability. This joint advisory was issued in response to the active worldwide exploitation of the Log4j vulnerabilities. CISA Director Jen Easterly called the Log4j vulnerability a severe and ongoing threat to organizations and governments around the world, and partner nation leaders echoed similar sentiments. You can view the Log4j incident timeline in Figure 1.6 (`http://unit42.paloaltonetworks.com/apache-log4j-vulnerability-cve-2021-44228`).

Unlike SolarWinds, which affected a specific product family from a specific vendor, Log4j's impact was much more diverse and distributed. Log4j was in use across everything from developer tooling, cloud service offerings, and security vendor products. As we will discuss in the cloud-focused chapter of the book, the largest cloud service providers (CSPs), such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, all issued guidance around Log4j because it was used in many of their cloud service offerings at the time of the incident. This, of course, can have a downstream impact on not just customers directly consuming the cloud services, but also on other organizations building on top

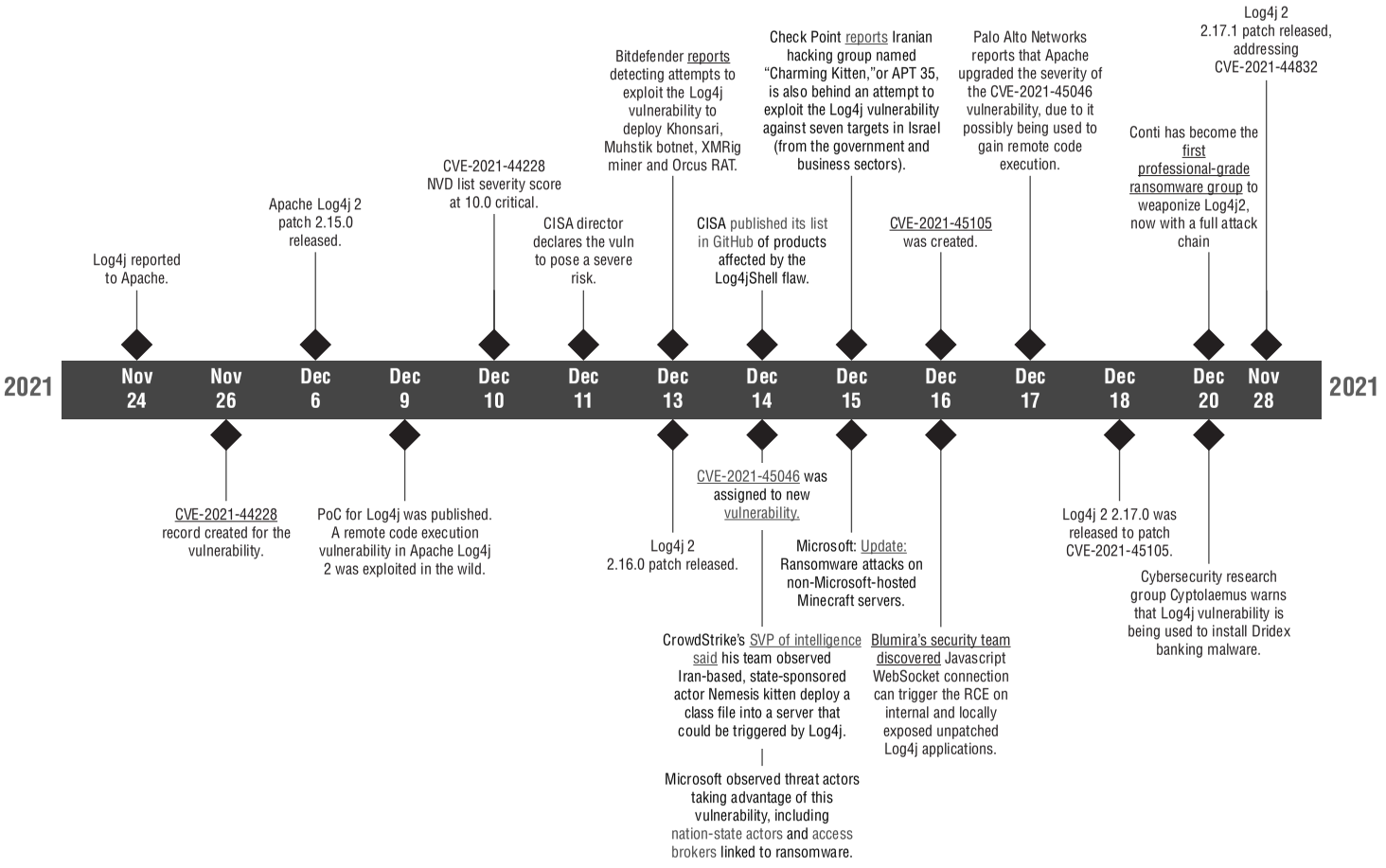
2021
Nov 24
Log4j reported to Apache.
Nov 26
CVE-2021-44228 record created for the vulnerability.
Dec 6
Apache Log4j 2 patch 2.15.0 released.
Dec 9
PoC for Log4j was published. A remote code execution vulnerability in Apache Log4j 2 was exploited in the wild.
Dec 10
CVE-2021-44228 NVD list severity score at 10.0 critical.
Dec 11
CISA director declares the vuln to pose a severe risk.
Dec 13
Bitdefender reports detecting attempts to exploit the Log4j vulnerability to deploy Khonsari, Muhstik botnet, XMRig miner and Orcus RAT.
Log4j 2 2.16.0 patch released.
Dec 14
CISA published its list in GitHub of products affected by the Log4jShell flaw.
CVE-2021-45046 was assigned to new vulnerability.
CrowdStrike's SVP of intelligence said his team observed Iran-based, state-sponsored actor Nemesis kitten deploy a class file into a server that could be triggered by Log4j.
Microsoft observed threat actors taking advantage of this vulnerability, including nation-state actors and access brokers linked to ransomware.
Dec 15
Check Point reports Iranian hacking group named "Charming Kitten,"or APT 35, is also behind an attempt to exploit the Log4j vulnerability against seven targets in Israel (from the government and business sectors).
Microsoft: Update: Ransomware attacks on non-Microsoft-hosted Minecraft servers.
Dec 16
CVE-2021-45105 was created.
Blumira's security team discovered Javascript WebSocket connection can trigger the RCE on internal and locally exposed unpatched Log4j applications.
Dec 17
Palo Alto Networks reports that Apache upgraded the severity of the CVE-2021-45046 vulnerability, due to it possibly being used to gain remote code execution.
Dec 18
Log4j 2 2.17.0 was released to patch CVE-2021-45105.
Dec 20
Conti has become the first professional-grade ransomware group to weaponize Log4j2, now with a full attack chain
Cybersecurity research group Cyptolaemus warns that Log4j vulnerability is being used to install Dridex banking malware.
Nov 28
Log4j 2 2.17.1 patch released, addressing CVE-2021-44832
2021

Figure 1.6

of the CSP's infrastructure and services which emphasized not only the impact of a vulnerable software component but also the cascading impact it can have due to service providers in the software supply chain.

In addition to the insight provided from GAO and others, the Cybersecurity Safety Review Board (CSRB) took aim at Log4j as their first cyber incident to investigate and report on (`http://cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf`). The CSRB was established pursuant to the cybersecurity Executive Order (EO) "Improving the Nation's Cybersecurity" (`http://cisa.gov/sites/default/files/publications/Cyber%20Safety%20Review%20Board%20Charter_508%20Compliant.pdf`). The CSRB was also established to be similar to the National Transportation Safety Board (NTSB) in the sense that its goal is to review major cyber events and make concrete recommendations to drive improvements across both the public and private sectors. The board's makeup consists of both public and private sector leaders of varying unique expertise. Their first report, which, as noted earlier, focused on Log4j, makes significant mention of the need for software transparency, inventory, and governance, with software bills of materials (SBOMs) a core component of that pursuit. The report calls for organizations to make use of SBOMs to improve accurate IT asset and application inventory, and for organizations such as the Office of Management and Budget (OMB), Office of the National Cyber Director (ONCD), and CISA to provide guidance for effectively using SBOMs as the ecosystem matures. The report mentions SBOM 18 times, calling for both adoption and increased investment in the area of SBOM and software transparency for public and private sector organizations.

CSRB's Log4j report is comprehensive and breaks their recommendations into four categories. Those include addressing the continued risks of Log4j, driving existing best practices for security hygiene, building a better software ecosystem, and making future investments. The report acknowledges that organizations will be wrestling with Log4j vulnerabilities for years to come and should continue to report and observe for Log4j exploitation. The report calls for organizations to invest in their capability to identify vulnerable systems, establish vulnerability response programs, and continue to develop accurate IT and application inventories, which is where SBOMs play a part in the context of software components and OSS consumption. Organizations with robust inventories of software components in their enterprise will be better positioned to respond to the next Log4j-type incident knowing if and where their organization is vulnerable. The report calls on OSS developers to participate in community-based security initiatives and to invest in training developers in secure software development, which is a key recommendation in the OpenSSF OSS Security Mobilization Plan, which we will discuss later in this book. It also calls for improvements in SBOM tooling and adoption and investments in OSS maintenance support for critical services. Lastly, the report also calls for

making investments in key areas such as baseline requirements for software transparency for federal government vendors, exploring a cyber safety reporting system (CSRS), and studying incentive structures to build secure software. All these recommendations align with recommendations made by other leading organizations in both the public and private sector, such as NIST, the Linux Foundation, OpenSSF, and many others.

In a nod to the pervasiveness of the threat of vulnerable compromised OSS components, such as Log4j in this case, the FBI and CISA issued a Joint Cybersecurity Advisory as late as November 2022 announcing that a U.S. Federal Civilian Executive Branch (FCEB) agency had experienced an Iranian government-sponsored attack. The malicious actors exploited a Log4Shell vulnerability in an unpatched VMware Horizon server, installing cryptomining software and even moving laterally to a domain controller (DC) and compromising credentials to implement reverse proxies to maintain persistence in the environment (`www.cisa.gov/uscert/sites/default/files/publications/aa22-320a_joint_csa_iranian_government-sponsored_apt_actors_compromise_federal%20network_deploy_crypto%20miner_credential_harvester.pdf`).

Landmark Case 3: Kaseya

Another very prominent software supply chain attack worth discussing is the Kaseya ransomware attack. Kaseya (`http://kaseya.com`) is a unified IT management and security software firm that looks specifically to help managed service providers (MSPs) and IT teams. Their solutions are designed to help teams improve in the areas of IT security, efficiency, and service delivery.

The Kaseya ransomware attack occurred in July 2021 and is estimated to have impacted 2,000 organizations. It has been attributed to the REvil ransomware group, a Russian-speaking group of malicious actors, primarily. Individuals with the REvil group have been tied to attacks such as Kaseya but also other attacks against U.S.-based businesses and even government entities.

Kaseya offers both on-premises and software-as-a-service (SaaS)-based software solutions to customers. On July 2, 2021, the organization's incident response (IR) team detected a potential security incident with their remote management software named Kaseya VSA. The initial investigation warranted enough concern that the organization advised all their on-premises customers to shut down VSA servers until further notice, and they shut down their own SaaS-based offering as well (`http://helpdesk.kaseya.com/hc/en-gb/articles/4403440684689`). Kaseya advised customers that the shutdown was critical because the attackers were removing administrative access from VSA during their initial attacks.

Kaseya then engaged the external authorities and expertise of the FBI and CISA to collaborate on the IR activities. Kaseya quickly provided tools to help

organizations determine if they were impacted by trying to detect the compromise. While organizations like CISA and the FBI were engaging with Kaseya, they also started to provide guidance to potentially impacted customers (`http://cisa.gov/uscert/ncas/current-activity/2021/07/04/cisa-fbi-guidance-msps-and-their-customers-affected-kaseya-vsa`). Their guidance included activities such as using the provided detection tool from Kaseya, bolstering authentication processes, and minimizing networking communications to known trusted IP addresses.

In the meantime, Kaseya began testing patches to resolve the issues and putting other mitigating security controls in place. They also published guidance for customers to try to prepare them for the upcoming patch activities (`http://helpdesk.kaseya.com/hc/en-gb/articles/4403709150993`).

During this process on July 8, 2021, the White House made official statements attributing the attack to Russia. U.S. government leadership echoed statements promising to hold Russia accountable for the attack. While Kaseya was diligent in their efforts to resolve the issue for customers, both on-premises and in the cloud, claims began to surface that the executives at the organization had previously been warned about their software's flaws over a several-year window (`http://krebsonsecurity.com/2021/07/kaseya-left-customer-portal-vulnerable-to-2015-flaw-in-its-own-software`). What made the incident more interesting is that the CVE involved, CVE-2015-2862, belonged to Kaseya themselves.

While the attack itself was said to have only impacted 0.1 percent of the organization's over-40,000 customers, according to comments from the organization's CEO, Fred Voccola, that percentage still equated to between 800 and 1,500 small to mid-sized organizations that were affected through their MSPs who were using Kaseya software. This trickle-down cascading impact is a perfect example of the complexity of the modern software supply chain and how the impact on a single piece of software can affect thousands of downstream consumers. Most of these consumers have no insight into the software their MSP uses to deliver services to them because the software supply chain is opaque, particularly when dealing with external service providers such as MSPs and CSPs. Downstream consumers are only aware of the services or software they directly consume from their providers use in the process of delivery.

It is worth noting, going back to the comments made by the White House regarding holding those responsible accountable, that arrests have been made. The U.S. Department of Justice (DOJ) announced in November 2021 (`http://justice.gov/opa/pr/ukrainian-arrested-and-charged-ransomware-attack-kaseya`) that Russia's domestic security agency arrested 14 individuals associated with the REvil ransomware group. This also included seizing $6.1 million associated with the group's various ransomware exploits. The DOJ leadership also emphasized that they would continue to target individuals who are focusing on victims in the United States.

What Can We Learn from These Cases?

The most important takeaway from these cases is that focus on traditional "last-mile" defenses like hashing and code signing is simply not enough and may project a false sense of security. The most successful attacks happen upstream of these processes, and signing malicious code might create more harm than not signing it at all.

Second, just like traditional network defense, establishing baselines of what "normal" looks like creates a standard reference for anomaly detection. This includes both the production of many software bills of material artifacts, as well as metadata about those artifacts like hashes and code signing. Though not definitive (and this may seem to contradict the previous paragraph), these are still valid techniques, but they must be performed early in the process and be validated and revalidated at every step in the process.

In addition to these static artifacts, understanding code execution flows and network behaviors for software can be valuable in identifying malicious functionality. For instance, in the case of SolarWinds, it was a behavioral change, not necessarily anything identified in a code review, that illuminated the compromise.

Secure software development is a process with many stages and many opportunities for compromise. While the concepts of zero-trust architecture were designed for identity and resource access, such as application access control, the core tenets are no less applicable to secure software development. As a reminder of the core tenets of zero trust, these include:

- Users and identity
- Device
- Network and environment
- Application workloads
- Software supply chain
- Data

Secure software requires a multidisciplinary approach throughout the life cycle. And while a technique like a software bill of materials is one tool in the toolbelt, it would not have prevented sophisticated attacks such as SolarWinds. There is no "one size fits all" security widget that will address these issues, and it requires a transformative change in how we develop software today and how we manage trust at all phases of the software development process and across our software supply chain. Organizations shipping code are responsible for all the code they ship, not just the code they write, and this means that validation of upstream inputs are just as, if not more, important than software produced by their own development teams.

Summary

In this chapter we discussed the background of software supply chain threats and some of the relevant incentives for attackers. We also discussed the potential anatomy of a software supply chain attack. We covered the fundamentals of Threat Modeling and the role it can play in preventing software supply chain attacks. Lastly, we discussed some of the landmark software supply chain attacks that have impacted various entities such as proprietary software vendors, OSS components and managed service providers (MSP)s.