ATTACK SURFACE ANALYSIS
Reduce System and Organizational Risk

Carol Woody, PhD.; Robert Ellison, PhD

June 2020

Much effort is expended implementing security controls and practices to address mandated policy. However, operational experience is showing that these steps are necessary, but not sufficient. The mantra to "think like an attacker" has been widely bandied by experts and contractors in the field. For those who struggle daily to make technology perform as needed, this advice poses a major challenge. Attacker capabilities are increasing continually. How should one determine and address possible system attacks?

System defenders need a systematic way to identify and enumerate potential threats and prioritize the mitigations. Threat modeling is a process that can meet that need. Defenders need to analyze the controls or defenses that should be incorporated into a design based on the attributes of the system, the profile of probable attackers, the most likely attack tactics, and likely attacker objectives. Threat modeling consists of answering four questions.[1]

1. What are you building?

   We can describe what we are building with dataflow diagrams and use-cases which describe the information exchanges. For example, consider the following dataflow and use case:

Figure 1: Dataflow

Use-case: *User submits a 5-digit numeric ID and is shown a web page with the ID, name, and phone number for the entry with that ID number*.

2. What can go wrong?

   An attacker observes a clear-text data stream or reads an unprotected data file.

[1] Shostack, Adam. *Threat Modeling: Designing for Security*. John Wiley and Sons. 2014.

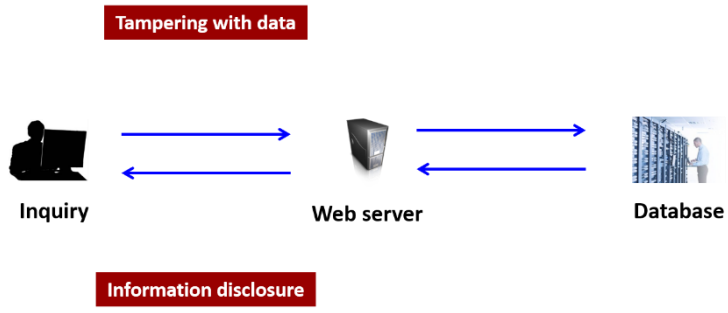

Figure 2: Faults

3. How should you respond to a failure?

   Access controls and data encryption can be effective risk mitigations.

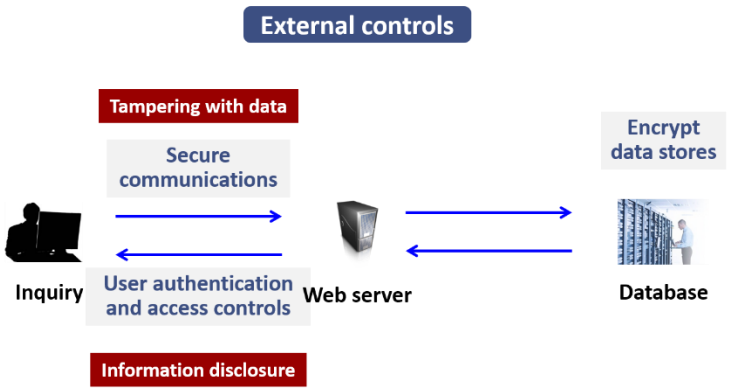

Figure 3: External Controls

But invalid input can also lead to information disclosure. Consider the following implementation of the proposed use-case:

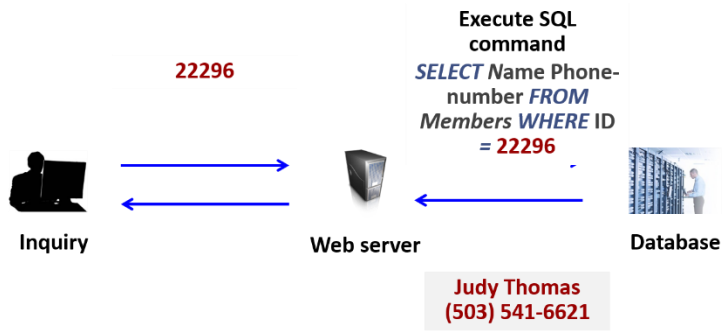

Figure 4:Use-Case Design

This design satisfies the use-case requirement when the submitted input is an integer associated with an entry in the database table Members. But an unverified input value can enable attackers to construct their own queries. For examples, a submitted value of

```
12296 or (1= 1)
```

generates a valid SQL command where the condition is always true and hence returns information on all entries in the table Members.

Successful software security analysis builds on knowledge for how systems were compromised and the mitigations that have been successfully deployed. Such attacks are examples of what can go wrong. The Common Attack Pattern Enumeration and Classification (CAPEC)[2] provides a comprehensive dictionary of known attack patterns employed by adversaries to exploit known weaknesses in cyber-enabled capabilities (https://capec.mitre.org/). CAPEC lists attack patterns by *Mechanisms of Attack* or by *Domains of Attack.* SQL-Injections attacks appear in the *Inject Unexpected Items* mechanisms category. The Common Weakness Enumeration (CWE)[3] entry for SQL-Injections includes recommended mitigations.

4. How well have you done the analysis?

   Provide evidence that increases assurance such as engineering analysis and test results. Demonstrate confidence that the analysis of what can go wrong has been sufficiently complete.

Surprise is likely a response to an unanticipated security failure. Neither the CAPEC nor the CWE easily support a completeness analysis of likely faults as there are more than 500 attack patterns and more than 800 identified software weaknesses. A more systematic approach to identifying potential security failures begins with identifying the system assets that are exposed to an attack. Such a collection is called an attack surface.[4] The attack surface for the use case shown in Figure 4 includes all externally exposed assets such as data stores and networked data flows and all software components that processes externally supplied data.

The CAPEC *Inject Unexpected Items* category is extensive in part as a consequence of the need for commercial suppliers to enable buyers to extend the functionality of a product to meet their specific needs. For example, macros extend the functionality of Office Word and Excel applications. A database vendor provides a general-purpose query capability by incorporating a SQL programming capability.

Extensibility is often a feature for commercial products in order to expand the market for the software. Web pages include HTML commands to display data and JavaScript program segments to enhance

---

2 https://capec.mitre.org/

3 https://cwe.mitre.org/

4 For more information, see https://msdn.microsoft.com/en-us/library/ms972812.aspx

functionality. Extensibility has in each instance also provided opportunities for attackers to execute their own commands.

For example, the *Inject Unexpected Items* category includes CAPEC 182 that describes *Flash Command* injections. For a period of time, third-party software such as Adobe's Flash viewer were required to display graphics on a web page. In a flash command injection, an attacker tricks a victim into reading flash content that executes malicious commands. In 2011, the security vendor RSA was attacked with the objective of obtaining the authorization certificates for a DoD contractor and using those certificates to access sensitive aircraft design data.[5] The attack exploited a vulnerability in Adobe's Flash graphics viewer to compromise an RSA business services employee to create a backdoor into RSA's internal network. Over time, a sequence of account compromises eventually provided the attacker with access to the desired certificates. Most likely the attack surface for RSA's certificate management system would not have included Adobe's Flash viewer which, in this instance, was associated in unrelated activity in the business unit. But the Flash viewer was a member of RSA's IT network's attack surface, and the attack took advantage of the implicit trust associated with employee activities on the RSA IT network. Essentially, internal system development has to consider whether internal systems or networks have been compromised.

Equifax was compromised in 2017.[6] The vulnerability was in widely used third-party software. Widely used software is also targeted by attackers as a new weakness can potentially be applied across a wide spectrum of targets. A patch was readily available, and software attacks reengineered from patches often have appeared within days of a public release.

Members of an attack surface are often third-party add-ons such as the Flash viewer. But increasingly, the attack surface involves infrastructure services and development constructs. The attacker compromised user-facing software at Equifax, but that software was not a commercial product. Instead the vulnerability was in an infrastructure service that provided a framework for easier and more rapid development of Java applications. Such frameworks can improve security by consolidating security analysis into the development of that framework thereby reducing the likelihood of programmer generated vulnerabilities.

Containers are another example of how development technologies can affect the attack surface. Containers are widely used to implement continuous integration and deployment requirements. But there is a tradeoff as that usage also expands the attack surface. Container weaknesses appeared in the CWE in 2018. Kubernetes, a widely deployed open source application programing interface (API) that controls how and where those containers will run, also expands the attack surface.[7]

---

5 For more information, see https://www.theregister.co.uk/2011/04/04/rsa_hack_howdunnit/

6 See https://epic.org/privacy/data-breach/equifax/

7 See https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/

The development of an attack surface during acquisition planning provides a systematic way to anticipate security risks and a leading indicator of the effort and expense that could be associated with resolving the potential problems. An analysis of the anticipated attack surface could be part of an evaluation of responses to a proposal.

The attack surface continues into deployment. Changes in usage after deployment can invalidate the selected mitigations. New vulnerabilities will be identified for commercial software, cloud services such as Infrastructure as a Service (IaaS) and Software as a Service (SaaS), and other development tools such as container and orchestration tools. System connectivity typically increases over the life of a system with the corresponding increases in the attack surface as change creeps into system components and usage. Orchestration software such as Kubernetes will add new features which could be exploited. Can the risks for emerging technologies be incorporated into continuous monitoring?

Historically security for large systems has had to find ways to securely compose what are considered to be secure system components. Kubernetes is an integration mechanism for containers. As was noted earlier, that integration software can itself be compromised by inconsistencies in the specifications. We can apply static analysis to source code in traditional development. Can we do the equivalent analysis to find security weaknesses as we build systems by composing infrastructure services and development frameworks?

Important Considerations of an Attack Surface

Keep in mind that an attacker needs three key elements: availability of a vulnerability, access to it, and the capability to exploit the vulnerability. The first two of these are directly controlled by the decisions made during the acquisition and development lifecycle. Software usage is increasing exponentially and all software, even the highest quality, contains weaknesses that could be exploited by an attacker. The National Vulnerability Database[8] documents an ever-increasing volume of available vulnerabilities. Research has shown that the increased quality of software does reduce vulnerabilities,[9] but quality alone is insufficient.

As shown in the Equifax example, weaknesses which can be exploited may be introduced through the use of third-party software. The choice to implement third-party software must include the discipline to continually monitor the product information and apply updates issued by the vendor as soon as possible to reduce the opportunity for the attacker. The operational environment must be resourced to address the needed level of monitoring and maintenance. These resources need to have knowledge of what third-party software is embedded in the operational systems. Most vendor products also include other third-party software that may be invisible to the acquirer if there is not a well-defined ingredient list. At the

---

8 See https://nvd.nist.gov/

9 Woody, Carol; Ellison, Robert; & Nichols, William. Predicting Software Assurance Using Quality and Reliability Measures. CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589

moment we must trust each vendor to do their own monitoring and maintenance. An effort is underway at the federal policy level, led by the National Telecommunications and Information Administration (NTIA) at the U.S. Department of Commerce, to require a software bill of materials (SBOM) for each product.[10] This effort aims to provide transparency to the lengthy chain of third-party components that go into the creation of software code, which may be open sourced and can contain known vulnerabilities. However, until such time as an SBOM is widely available, each organization must assemble their own information. A key ingredient in this effort is the ongoing relationship with the vendor to share information and respond to vulnerabilities as soon as they are identified.[11] Who establishes these relationships and how are they monitored to ensure they are operating effectively?

From the RSA example, we learn that if the attacker knows the technology used for securing their target, they can apply very creative ways of gaining a form of access, including attacking the company that developed the security capability. Establishing trustworthy credentials is a critical element in limiting the attacker potential access to software vulnerabilities. In the example, the trust was established between two organizations but the monitoring needed to confirm the trust was working properly was not in place. In many cases these boundaries are defined by choices made at system design when decisions of reliance on third-party software are made. How should an organization establish and monitor the trust relationships that are desirable for smooth operations but present new risks, so that this situation is avoided? When the relationship functions across organizational support boundaries, decisionmakers lacking familiarity with the cybersecurity risks can be making choices that increase the attack surface for the organization.

In some cases, the trust is structured across organizational boundaries, as when cloud technology is included as a component in the operational environment. The decisions about how the cloud technology will be monitored are established with the acquisition, which may be external to the participants who are setting up the data exchange. How should those making the decision to use the cloud determine that the trust level is sufficient for the assets that are planned for its use? How should the process be structured to ensure the trust relationship is working as expected?

Another type of trust relationship involves the automated sharing of data which is frequently established during system design. Systems are designed and built to connect with existing systems and to reuse existing software components. These reused components and external interfaces were based on a different set of requirements and may not have been maintained to the expected quality level. What provisions are established to determine if sharing is appropriate and to monitor and manage this type of the trust relationship, which may be needed to meet business/mission needs but carries potential risks for attack. Will these reused components be maintained to the needed level based on the expanded use as attacker capabilities grow overtime?

---

10 See https://docs.house.gov/meetings/AS/AS26/20190910/109894/HHRG-116-AS26-Wstate-RinaldoD-20190910.pdf

11 See *Lessons in External Dependency Management* https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=427486

Third-party software is also introduced into a system through development frameworks and code libraries that support software builders with ready routines in their coding language available to boost productivity as they create new software components. For code libraries, where the code is sourced and the care with which these are administered and maintained will impact the vulnerabilities that are introduced into new software modules. Almost every application today contains code from open source libraries, which allow developers to quickly add basic functionality. A recent report examined 351,000 external libraries in 85,000 applications and found that open source libraries are in wide use and contain vulnerable code.[12] Integrated development frameworks, another timesaver for developers, need to be managed and maintained as third party software.

In addition, the products of the software development must be effectively shepherded through the development process to reduce opportunities where attackers could introduce vulnerable code. How are effective configuration management processes and practices established and maintained to ensure developed code is protected? What are the mechanisms that track the analysis and testing performed on the code and the steps for verification and validation? How is the code transitioned to the operational environment to ensure integrity is maintained? As more of these steps are taken over by automation, the integrity of the processes and control of how they can be changed become increasingly important.

Establishing Appropriate Risk Tolerances

Each type of software integrated into a system or connected in some manner indirectly is usually also a composition since all builders use available components, which can include open source, code libraries, third party products and reused code. Each piece may be well written, but that provides no indication of the security of the composition. There are methodologies that propose individual enumeration of every potential problem, but the data quickly mushrooms into an intractable problem.[13] Instead, an initial acquisition focus needs to consider the processes a contractor has in place to support the identification and remediation of potential risks. What capabilities have been put in place by the contractor to address their supply chains that provide evidence that will allow the acquirer to ensure that their integration of software is sufficient for the security requirements of the context in which it will operate? How is the contractor managing their development pipeline to ensure the infrastructure and tools that touch the code are uncompromised? How will an acquirer test software to ensure vulnerabilities have been removed? There are many tools that can be used to analyze source code, but each tool has limitations. It is important that a mix of tools are applied that explore a broad range of potential vulnerabilities.

The acquirer has responsibilities as well. Are there sufficient requirements in place that ensure that identified vulnerabilities are fixed? If code is reused, how will it be evaluated for potential risk? If source code is not available, what are the options for binary analysis? If the components cannot be

---

12 See https://www.veracode.com/blog/research/announcing-our-state-software-security-open-source-edition-report

13 Banga, G. Why is Cybersecurity Not a Human-Scale Problem Anymore? *Communications of the ACM*. Volume 63. Number 4. April 2020.

effectively evaluated, how might they be isolated and monitored so that potential attackers cannot use them to gain access to other parts of the system?

There is a growing body of security practices that can be applied.[14] Each selected practice requires effective installation and ongoing monitoring as technology is changed to meet new organizational needs. Automation to support continuous monitoring[15] is growing in popularity, but this requires an organizational commitment of resources to support the selection and implementation of the automation mechanisms, and structuring of monitoring mechanisms to identify when something happens that falls outside of acceptable limits.

Many systems must operate in bandwidth constrained environments where additional monitoring can impact performance. Tradeoff decisions need to be addressed during design to implement the best options for both operational success and attack response. Since prevention is not always an option, designers need to incorporate mechanisms that support recognition of an attacker, resistance mechanisms to minimize damage, and effective recovery capabilities. These may rely on human monitoring and response. Will the resources be there to recognize and respond?

Reduce the Attack Surface

The designers of the system should consider ways to minimize the attack surface to reduce the ways in which an attacker could reach vulnerabilities. This requires determining the how a system can be attacked and the potential impact a compromise will have on the operational mission. Can trust relationships be strengthened? Should shared data be encrypted? Should vulnerable legacy code be scrubbed to remove vulnerabilities or rewritten? Are there capabilities in third-party software that are not needed and should be disabled? What kinds of attacks should be considered?

Attack patterns such as those described in CAPEC show that the examples provided in an earlier section of this paper are not uncommon. An attack under pattern number 152 “inject unexpected items” would include the SQL injections (pattern 248) described in Figure 4 as well as the RSA attack that was an injection into the Adobe Flash product (pattern 242). The architect would need to select the patterns that are possible for a design and determine if adjustments can be made to prevent such an attack.[16] These patterns are tied closely to the technology and can be grouped by outcome similarities.

Each potential acquisition source and type of technology may require a different level of analysis. Assuming one has direct access to the technology, there are tools to help identify vulnerabilities but these will be different for each coding language and technology platform. For services such as Cloud, third-

---

14 See NIST 800-53 Rev 4. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf. Rev 5, which adds controls for engineering and supply chain risk management, is currently in review (https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/draft).

15 NIST 800-137 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-137.pdf

16 As of May 27, 2020 CAPEC has 517 patterns

party software, and other types of acquisition, the acquirer would need to establish requirements for others to select the tools and do the analysis.

Few program managers and system engineers, even with the attack patterns, have sufficient knowledge to effectively identify the ways a system could be attacked and match these to appropriate mitigations. In many cases the technology has not been established when requirements for the acquisition must be identified. Too frequently we see acquirers providing only a list of controls that are required with no linkage to the possible ways the system could be attacked. Instead of attempting to specify tools and mitigations, more effective requirements should be developed using mission related attack scenarios that describe how a compromise would impact the mission of a system. From these, a determination can be made as to the controls needed or other constraints that could achieve a similar outcome. Some examples of these scenarios for the supply chain are provided in the appendix. These scenarios could be tailored for each specific system to provide as more detail context as more detail technology information is known.

Appendix

Supply Chain Attack Scenario

1. Subcontractor employee inserts malicious code into a software update for the system of interest.

Scenario: A technically savvy employee at a subcontractor in the program's supply chain becomes upset at a perceived slight from the prime contractor's staff during a technical exchange. The subcontractor employee decides to execute a cyber attack on the system of interest. The employee performs reconnaissance to obtain subcontractor and prime contractor artifacts (for example, requirements specifications, architecture and design documents, and source code) that describe the system of interest. The employee develops malicious code and then inserts it into a software update for the system of interest. The prime contractor integrates the software update with the malicious code into a software package for the subsystem of interest. Acceptance testing by the prime contractor and the program does not detect the malicious code, and the updated software is deployed in the system of interest.

2. A trusted supply chain partner accidentally transmits malicious code to the system of interest.

Scenario: A computer used by an employee of a trusted supply chain partner is infected with malicious code. The malicious code spreads to the organization via the trusted path. The malicious code eventually propagates to the system of interest.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-0486