8-2 Journal: Portfolio Reflection

Janera Dobson

Southern New Hampshire University

CS-405-H4736

Table of Contents

[Introduction 3](#_Toc409783205)

[Types](#_Toc409783206) of Hackers………………………………………………………………………..3-4

[Adoption](#_Toc409783207) of secure coding standard, and not leaving security to the end………………...4

[Evaluation](#_Toc409783208) and assessment of risk and cost benefit of mitigation………………………4-5

[Zero](#_Toc409783209) Trust……………………………………………………………………………….5-7

Implementation and recommendations of security policies…………………………….7-8

[References………………………………………………………………………………..9](#_Toc409783211)

8-2 Journal: Portfolio Reflection

This is my final course reflection as well as my portfolio submission. I will reflect on a discussion about the types of hackers. Adoption of a secure coding standard, and not leaving security to the end. I’ll evaluate and assess the risk and cost benefit of mitigation. Along with zero trust, and implementation and recommendations of security follows.

There are three types of hackers called white-hat, grey-hat (CIA), and black-hat hackers, who pose ethical questions for developers in the field. White-hat hackers tend to use their abilities to do good rather than evil. Companies sometimes compensate them for being security specialist that attempts to find security flaws. Back-hat hackers use their extensive knowledge about breaking computer networks and bypassing security protocols to cause damage. They are known for writing malware to gain access to systems with the motive for personal or financial gain. By seeking financial information, personal information, login credentials and then selling them leads to criminal offense and jail time. Subsequently, grey-hat hackers (CIA) are a mixture of both white- and black-hat hackers. Grey-hat hackers look for vulnerabilities within a system without the owner's permission or knowledge. They'll even find these vulnerabilities and reach out to the owner with a way to fix them for a fee, but if they don't comply, they'll exploit the vulnerabilities to everyone. Their intentions are not malicious, but more so to gain something out of their discoveries yet can be viewed as still illegal.

What is meant by the statement "Don't leave security to the end" as a best practice in secure coding is to think of hackers throughout the process of developing your application, not just when it is done. Software vulnerabilities are an ever-present risk; therefore, securing your codes is fundamental. Therefore, having secure and protected codes from the beginning of your application ensures that you have done developing your application with integrity. Nevertheless, you will not have to spend countless hours going back in within your codes to redesign your application. It will save time, money, frustration and ensure your application's safety when done from the beginning.

The steps you can take to prevent threats are thinking of one's motive for an attack and understanding no one is safe. According to the "Top 10 Secure Coding Practices," you should validate input to eliminate vulnerabilities. Heed compiler warnings to detect and eliminate security flaws. Use architect and design for security policies to enforce security policies. Keeping your application simple to decrease the likelihood of errors. Use default deny to give access only where permitted and adhere to the principle of least privilege to reduce the chances of attackers executing arbitrary code. Sanitize data sent to other systems to sanitize data before invoking subsystems and practice defense in depth to add layers of protection to prevent a security flaw and accidental exploiting vulnerable information. Quality assurance technique to effectively identify and eliminate vulnerabilities, and lastly apply secure coding standards. (Schiela, R. 2018, May 02). Another step in preventing threats is by conducting unit testing to instantiate a small portion of your application that will verify its behavior independently from other parts of your application. Taking it an step further would be to compiling your program and submitting it within CppCheck to flag any errors or warnings. Following these steps is essential to prevent threats to ensure you have developed checks within your codes to strengthen your software and get rid of any security issues and vulnerabilities.

Security testing has also historically added significant time to a schedule because it is traditionally a manual effort. Adding security tasks into a DevOps culture would then seem to be working against the goal of and reducing the effort it takes to release updates and new features. Operations teams moving to a DevOps culture faced a similar challenge when so many servers and applications were manually deployed. The solution for operations teams was to work toward automating these tasks to reduce the friction of software deployment. This automation eventually helped reduce cycle times.

Some types of security testing are also easily automated and much of it can be done in parallel with other project tasks. Static analysis security testing (SAST) can be run while code is being developed. You can also run automated dynamic analysis security testing (DAST) scans against applications in test or staging environments. You can even write automated tests for the security configuration of the infrastructure itself. While these automated scans will not replace all types of security testing that you may want to do, the effort does two things: 1) it improves the timeliness of feedback from security, and 2) it frees security staff up to tackle the harder problems. Furthermore, by automating these tasks, you also avoid inflating the schedule, continuing the focus on reducing cycle times.

Organizations can see a variety of benefits when adopting DevSecOps culture, practices, and integrating security early and throughout the project lifecycle. Businesses that successfully transform from DevOps to DevSecOps will see advantages such as a reduction of security-related rework, as well as find an increase in employee engagement with an improved security posture without impacting team agility and flexibility.

The zero-trust concept is based on authentication, authorization, and validation of security configurations and posture before being granted to data or applications. The zero-trust policy is based on real-time visibility in users attribute like:

* User identification
* Firmware versions
* Endpoint hardware type
* Operating system version
* Patch level
* Vulnerabilities
* User logins
* Application installed
* Security or incident detections

**The model of zero trust policy includes the following:**

* The identities are secured and validated by using multifactor authentication.
* The device is validated and managed as healthy
* The least privileges are provided

If we incorporate a zero-trust policy, it will help restrict access to data and reduce attack surfaces via segmentation, data breaches, interception of data, and data leaks. If you are not using hashed data, end-to-end encryption, automated backups, you can surely use zero trust in your security plan. The mobile devices, biotech, drive, cloud, and edge can be protected using zero-trust security.

The developers should get on board, the policy of zero trusts because of the following reasons, and that is as follows:

* 1. **Reduce business and organizational risk**
* The zero trust assumes the applications and services; if they are malicious, they would be disallowed from communicating.
* If the software and services are meeting their predefined authentication and authorization requirements.
  1. **Help to reduce the risk of a data breach**
* Because the zero-trust policy is based on workload, it is easy for the security team to identify malicious activity and stop that malicious data-based activity.
  1. **Provide access control on cloud and container environment**
* The zero-trust policy is based on the identification of communicating workloads. In this, the security stays close to assets that require protection, and basically, it is not affected by network constructs like IP address, protocols, and ports.

**4. Support compliance initiative**

* Zero trusts can reduce the number of places, and network communication can be exploited in different ways. The zero trust segmentation can be implemented. The organization can create perimeters on sensitive data like -- credit card or PCI data and data backup. Using the fine-grained controls, which helps to keep regulated data separate from other and nonregulated data.

Implementations and recommendations of security policies includes the developer knowledge gap, lack of AppSec tool integration, and pipeline friction and developer overload. Developer knowledge gap is a barrier to any attempt to shift security left, so your DevSecOps strategy needs to address it using a combination of training. With the ack of AppSec tool integration developers need a consolidated view of issues, but combining and reconciling findings from multiple vendors’ tools can be difficult. This challenge is the motivation behind the design of [Code Sight](https://www.synopsys.com/blogs/software-security/code-sight-ide-plugin-sca/)™, an IDE plugin that brings results from SAST and SCA together directly at the developer desktop. Furthermore, the pipeline friction and developer overload is a lengthy, human-intensive model is incompatible with the high-velocity, integrated, and automated model of DevOps. And it makes clear that it’s not enough to build security into DevOps. You need to leverage AppSec tools that have DevOps built into them.

References

Techquickie. (February 21, 2017). Types of Hackers (Hats) Explained. Retrieved from. <https://www.youtube.com/watch?v=Q0F-pv-n_lc>.

Exposing the Deep-Secure elements Of smartcards: Christopher Tarnovsky: Media center. (2019, June 20). Retrieved March 17, 2021, from <https://media.hardwear.io/exposing-the-deep-secure-elements-of-smartcards-christopher-tarnovsky/>.

Schiela, R. (2018, May 02). Top 10 Secure Coding Practices. Retrieved April 06, 2021, from <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>.

Schiela, R. (2020, May 20). SEI CERT C++ Coding Standard. Retrieved 2021, from <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>.

Vijayan, J. (2019, December 31). 6 DevSecOps best practices: Automate early and often. Retrieved 2021, from <https://techbeacon.com/security/6-devsecops-best-practices-automate-early-often>.

Devsecops: A complete guide to what, why, and how. (2021, March 30). Retrieved April 18, 2021, from <https://www.plutora.com/blog/devsecops-guide>.