8-2 JOURNAL

8-2 Journal: Portfolio Reflection

CS-405-10861-M01 Secure Coding 2024 C-3 (May - Jun)

Michael Zietz

Southern New Hampshire University

June 30, 2024

Defense in Depth (DiD):

When discussing DiD, Defense in Depth, it can be easy to say that the most amount of protection and security possible is the best solution. When in reality, the correct answer is more of a spectrum, a balance in fact, to discover for each project. The ultimate answer is defense that prevents all unwanted users from accessing private data, but also defense that does not hinder the correct and permitted users from accessing it either. We want our users to be able to prosper from using these applications rather than reducing their efficiency, and when a piece of software is so well defended that required and permitted users are obstructed, than the defense has gone too far.

This is a great question. In fact, I think this is one of the most important questions to ask when discussing secure software development from a business point perspective. Obviously, we want all the programs we create and release to the public and other companies to use to be as secure and protective of its users as possible, but this quickly becomes a game of cat and mouse chase. Too much time or work power spent on developing one product can leave a company with so much overhead expense that they can no longer make profit from what they make. It can cripple a developer studio if not done correctly. Priorities when it comes to security must be outlined so the most powerful development and cost-effective development goes towards securing this data rather than every last piece of data a program might use.

There are many possible aspects of DiD that can make it unique for each situation. The types of secure data that need to be locked down are important to analyze. Not all applications require stored sign ins with save account and password data. Also, the varying threats should also be discussed. Not all applications are going to need to be warry of SQL injections especially if SQL is not utilized by the program, but it should be checked that this security issue is not present rather than disregarded. A program is only as secure as its weakest link, so all dependencies should be analyzed for where obvious threats will be coming from.

Types of Hackers:

In 2016, a self-proclaimed white hat hacker informed the parental control technology company, uKnowKids.com, before it was bought out by Bark.us, that they had breached their private databases and had been able to download its contents. (White hat hacker discovers vulnerability, 2016) The database possessed intellectual property like customer data, business data, trade secrets, and developed algorithms that powered uKnowKid's technology services. (Woda, 2016)

The hacker, Chris Vickery, reported this vulnerability and provided information on how the hack was successfully enacted within twelve minutes of gaining access. Vickery did so to report the found vulnerability in the attempt to held uKnowKids to bolster their security and correct this issue before a malicious entity performs the same steps to gain access for harm. (Woda, 2016)

Following the report, UKnowKids promptly updated their system by patching the acknowledged vulnerability, and additional staff was brought onboard to proactively breach their systems as an ongoing security measure moving forward. (Woda, 2016) It appeared this was the first instance that this company experienced an event like this, and it taught them to take their company's security measures aggressively serious moving forward.

Despite Vickery proclaiming no malicious intention, this hack of uKnowKid's databases was not condoned or approved by any means which defines this incident as a grey-hat case where the risk Vickery could have faced for keeping any private information could be met by legal action. The post-incident staff hired to continually hack their own company would be defined as white-hat hackers as they were approved individuals attempting to break into their systems.

Don’t Leave Security to the End:

The phrase, “Don’t leave security to the end,” is a mantra often used to encourage the thoughtful and constant approach of secure coding throughout a project’s development. This is because when projects are developed with security in mind, it results in less issues occurring later during development where a project’s architecture and design has already been significantly constituted. This can lead to greater costs in time and workers in terms of reworking a design, so it adheres to security policies. Instead, the most efficient use of a company’s resources for development attempt to marry the process of security in all stages of development so no major concerns are overlooked or neglected until they become a mountain instead of a smaller mole hill. (Seacord, 2013)

Steps that can be taken to prevent these threats can be approached from multiple angles. In practice, core security training, establishing security requirements, establishing, design requirements, static analysis, dynamic analysis, and security review are the most important pieces of the security development cycle, and a product is only as strong as its weakest link. Therefore, in my opinion, a more well-rounded approach is more likely to provide the best results rather than spending the majority of a team’s resources on one area. These can be broken down into the important concepts that can deter threats. I would describe these as education, understanding expectations, upholding security standards, and reviewing created work. (Seacord, 2013) Without them, the DID approach to development is likely to faulter as the layered tactic relies on each layer being comprised of well researched, well thought out, and well executed security measures to come together to bolster a system’s security.

I think the use of unit testing is a great example of ensuring security is addressed intrinsically and not left until an issue is discovered later in the development cycle. Thanks to the convenient power of unit testing, it is possible for developers to run tests on actively created code bases while working on them to ensure that time spent developing results in keepable progress towards a project’s timeline. These tests can be run to look for various concerns in newly produced, smaller code bases to catch either early mistakes or missed vulnerabilities that could require more time and energy later down in development to fix or possibly rework. (Moradov, 2022) These unit tests produce helpful data on a codebase and signal if there are known warnings, errors, or vulnerabilities of used code or dependencies being utilized.

Zero Trust:

Adopting a zero-trust approach necessitates a fundamental change in both the organizational culture and its operational practices. It signifies that relying solely on perimeter defenses is inadequate for defending against security breaches. Under a zero-trust framework, every user undergoes the same authentication and authorization processes, without consideration of their physical location or network id. Only authenticated and approved users can interact with the organization’s software, and even those users are restricted to accessing only the resources essential for their tasks. Here, the concept of "no one is safe" also allows overlap with the principle of least privilege. Meaning only each user provided access obtains only access to what is absolutely necessary for their role and nothing more.

As for convincing those who oppose implementing this methodology, I think good communication between the two parties should be able to persuade them. Most likely, the largest opposition to this idea is the fear that it will hinder productivity through tedious steps that could be avoided by users. Examples like authenticating sign ins from workstations that access the same network and share files every day. Of course, they should be approved, and the extra steps can seem superficial and ineffective at first glance. I think showing them cases with instances where this level of security was not enough because malicious parties gained access to this permitted workstation and compromised the whole system's security. Also reiterating that it only takes one instance or slip up to ruin security, so why not take the easier approach and how the method meaning to be implemented would be easier than others previously used.

Motive for the Attack:

Considering the motive for the attack is an important mindset to adopt while reviewing a project’s code base. Looking for the exact places where a malicious party would need to gain access to reach private data can be a clever way of identifying vulnerable areas or ones that need additional attention to increase a program’s security. Also walking through important variables and stored data and how it is stored at rest, in use and in flight can create a mental map or storyline of data in the project and clarify areas that require additional or different security depending on what state the data is in. This is all assuming that the perpetrator’s motive is solely to gain access to important data and steal or modify it. They might also plan to attack whole services, thus shutting them down or using them to affect other programs that are used in congruence. Thinking of the different angles that attacks can come in and what their goals would be can help identify which practices are needed for a given situation rather than spending eternity implementing every security measure no matter how resource heavy or unnecessary it may be.

I think the best way to explain this to a new developer on the team is to first educate them on previous, well documented cases where security compromises occurred, identifying how they gained access, what they wanted, and what the damages were from the incident. Walking them through the perpetrator’s perspective of how they gained access to something they wanted and identifying multiple desires they could have had can start to showcase how important it is for the new member to not only be aware of these malicious parties but to use their own logic against them.

During the same month this journal was being written, June, 2024, there was a cyber-attack on CDK Global, a software provider that many car dealerships in America use. This forced many of these companies to temporarily revert to pen and paper methods of documentation because the attack was specifically aimed at the services the software provided gave to its customers. It is likely this attack was done intentionally for this effect to hinder as many companies as possible. (Granthem, 2024) This is just one example of what a malicious party is capable of. Other incidents aim to steal and sell private data while others seek to maximize damage to individuals or corporations.

Resources

Moradov, O. (2022, May 24). Unit Testing: Definition, Examples, and Critical Best Practices. *Bright Security*. https://brightsec.com/blog/unit-testing/

Seacord, R. C. (2013). Secure Coding in C and C++ (2nd ed.). Pearson Technology

Group. <https://mbsdirect.vitalsource.com/books/9780132981972>

Granthem, W. News, A. B. C. (2024). *Car dealerships revert to pens and paper after cyberattacks on software provider*. ABC News. Retrieved June 30, 2024, from https://abcnews.go.com/US/wireStory/car-dealerships-north-america-revert-pens-paper-after-111364086

Woda, S. (2016). Breaking news... A uKnow database was breached by a hacker, and here are the

facts as we know them right now. Resources.uknowkids.com. Retrieved May 24, 2024,

from <https://resources.uknowkids.com/blog/breaking-news...-a-uknow-database-was->

breached-by-a-hacker-and-here-are-the-facts-as-we-know-them-right-now

Woda, S. (2016). uKnowKids data breach update #2. Resources.uknowkids.com. Retrieved May

24, 2024, from https://resources.uknowkids.com/blog/uknowkids-data-breach-update

‌White hat hacker discovers vulnerability. (2016, February 24). US Official News.