# CS 405 Project Two Script

Devon Darling

04/18/2021

https://youtu.be/jUxij19Sy78

|  |  |
| --- | --- |
| **Slide Number** | **Narrative** |
| **2** | My security policy is broken down into a few different sections. We have the Core security principles, the Coding Standards, Security Automation, Risk Assessments, Encryption, and the Triple-A Framework that I worked on. This policy provides guidelines in terms of constructing systems for the company that upholds a high degree of security standards. The principles guide the standards which are the backbone of our defense-in-depth best practices. |
| **3** | We have a variety of security risks for our company, but like anything, some risks and vulnerabilities are more important than others. Say for instance the standard of proper naming conventions. That is a likely issue to have, but having it doesn’t pose a detrimental threat to the company, so it’s a low priority risk. For likely risks, we have standards 1,2,3,5,6,9, and 10 and for unlikely we have 4, 7, and 8. The likeliness and priority are somewhat correlated but not directly. Our top priorities are 2,3,4,5,8, and 9. And the low priorities are 1,6,7, and 10. We need to consider the likelihood along with the damage that can be caused if issues aren’t addressed. That was my decision-making process here. |
| **4** | How I wrote my standards, multiple principles aren’t covered by any of the standards currently. This means we need to build in additional standards to account for the additional principles. To save time, I will include a list of the standards and just use numbers when I am speaking about them. For Validate Input Data we have standards 4 and 8. For Heed Compiler Warnings I don’t have any aligning standards, the same for architect and design for security policies until we get to Sanitize Data Sent to Other Systems. For that principle, I have numbers 4 and 8. For Practice defense-in-depth, I have 4 and 8 as well. For Use Effective Quality Assurance Techniques, I have all 10 standards falling under that principle. And for Adopt a Secure Coding Standard, I also have every coding standard. Those were the two that I felt covered the standards the best. |
| **5** | My ordering for the coding standards based on priority was quite simple. Assessing the potential damage and the likelihood of the standard being exploited. Things like SQL Injections are very likely to happen if you are exposed, so that is high up on the list, while proper naming is something that doesn’t have a damaging impact on the company if it isn’t solved today. |
| **6** | Encryption in rest is a way of protecting the local storage of our data through encryption. The purpose of encryption in rest is to protect the data of the application from attackers who can access the storage device but that doesn’t have access to the application itself, making it so they can’t read the data. This policy applies because, to have a truly defense-in-depth strategy, we need to protect our data in all states.  Encryption at flight is essentially just encryption in transit. As the data is being transmitted, the data should be sent through encrypted channels. One example would be using HTTPS. This policy applies because where we already have our data encrypted in rest, why would we decrypt the data to transmit it, opening ourselves up to attack and breaking down our defense in depth strategy? We shouldn’t, which is why we need encryption at flight.  Finally, we have encryption in use which is essentially having the data encrypted until you authenticate a user has access to said data. This policy applies because like before if we open up one of the DiD components, we make the whole system potentially vulnerable. This is one of the most important in my eyes because it could be a user-facing application and if data isn’t properly stored, you could potentially release unencrypted data to a non-malicious user. Encrypting data until it is needed is a fantastic idea and a policy we need to uphold. |
| **7** | The Authentication part of Triple-A is the part that pertains to identifying users. This will be done typically through a User Login like all of us should be familiar with as we use them nearly daily. Each user has their own unique combination of credentials to access the network. The credentials provided are compared with the credentials stored in the database. This policy applies because user authentication is crucial to protect your network from attackers, it is a great first layer of defense.  After a user is authenticated, they must then obtain Authorization to perform tasks on the network. Their level of access to the system depends on the permission level set on their user account. Consumer type users may be able to navigate the network and manage their own account and access files they have permission to, but an administrative user could make modifications to the database, add new users to the system, and change permissions for other users. This policy applies because we need to practice defense-in-depth, after a user gains access, we should only provide them with enough permissions to complete tasks required by their user account, and only escalate permissions when absolutely necessary. That is supported by multiple secure coding principles.  The final part is Accounting. This monitors the resources that each user consumes while on the network. This can be divided up into different pieces of data like the amount of time on the system, the data sent and received, and the files they accessed on the network. This is a great final defense-in-depth measure to help secure our network. If somehow an attacker bypasses all of our other security measures, at least now we will be able to see what they are doing/accessing on our network. It is the last resort, but it is also necessary if we wish to have the most secure system that we can build. |
| **8-11** | For each of my four unit tests, it is easy to see how beneficial they can be to address the different security concerns that we are presented with. I have provided the first, two negative testing examples. The first example verifies that we get an exception thrown when calling an out-of-bounds index, the next proves that adding entries and resing changes the capacity of the collection. The two negative tests are just as useful to us as the positive tests I will be talking about next.  The positive examples are also beneficial for us. I believe the implementation seen here will be a close match to the implementation we would need to construct for our own systems. We simply need to break down our components and modify our testing to ensure that things like injection attacks are mitigated. |
| **12** | Given the current DevOps process, I believe some slight modifications can be made to automate the enforcement of the standards outlined in this policy. I believe that there should be thought put into the planning and design process to adapt the code to the standards and that we should keep a close eye on them as we are building, but crucially, using automation during and after the build is completed to check for any standards being broken should suffice as a way to uphold our system to the secure coding standards. This means we will be using automation in both the build step of DevSecOps and the Verify and test step to ensure that we meet and exceed all of our standards. |
| **13** | Now looking at tools, there are a couple that are used throughout the process and that can be used to identify a majority of our coding standard vulnerabilities. First, we have static analysis tools like C++ check. Then there are a variety of tools that serve the purpose of vulnerability scans that will be very useful as we build new components for the system. This should occur throughout the DevSecOps process. Finally, one tool that I believe will be crucial is Parasoft. This tool has a ridiculous amount of functionality that will be extremely beneficial to us. |
| **14** | There are some large benefits to acting now. For starters, vulnerabilities like injection attacks can cripple a company. That alone is enough for me to say that we need to act now. There are risks with the wide adoption of new standards. If they aren’t adopted properly, we could have a false sense of security, but if adopted correctly, we could start building our system in a way that is beneficial to the company’s security goals. The main problems are our security weak points, and the solution would be to adopt standards to address those weaknesses. We really should act now. |
| **15** | There are some large gaps in the security policy that I have identified. For starters, there are a few principles that I feel we need to develop standards to address. Default deny, Keep it simple, adhere to the principle of least privilege, and heed compiler warnings. We need to look at this logically. We have principles to follow, but no standards to adopt the principles. Without standards to guide engineers, we simply have a rule book. Nothing more than that. Let’s take a look at default deny. This is a principle that I don’t have a standard attached to. It’s a fantastic principle that will be crucial in our company’s policy, but without guidelines on implementation and expectations to uphold this policy, we have simply thrown the principle at a wall and hope it sticks. This is why we need a security policy that is fluid. Allowing it to change over time, to improve and adapt to new risks and challenges. It’s almost like telling someone that a car needs a key when they are building it, but you don’t specify the extent that the key is used. They only implement a locking system, but the push to start works without a key. That is quite problematic. That is why we need to expand our Security Policy. |
| **16** | Specifically, we should add standards to deny permissions and only extend further permissions when necessary. Only escalating permissions when necessary is crucial to upholding our principles. This will begin to relieve us from some risk. Next, I believe that including a standard to address dividing our system into subsystems would be very important. This standard will allow us to almost segment our systems so if we are ever attacked, the attacker must break into multiple systems to find what they are looking for. This combined with our AAA policies should give us a great deal of protection. |