# CS 405 Project Two Script

Craig O'Loughlin 02/19/2023

recorded presentation available at:

<https://youtu.be/nicoQyRhT-4>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Heelo and welcome to my presentation. My name is Craig O’Loughlin. In this presentation we will be going over a security policy proposal, and focusing on how we can implement the best security practices in software development. |
| **2** | To start, Our security policy is founded on the concept of Defense In Depth, as the illustration here shows. defense in depth means approaching security as a set of layers that all work to support each other. It includes perimeter security concepts such as securing the physical server location and providing firewalls. But we don’t stop there and assume we are safe. The security concepts continue all the way to the core of the application, involving security all the way down, involving implementations even on the underlying business data itself. |
| **3** | There are many layers and topics to consider, so The first thing our policy needs does is define a system for prioritizing security risks. Some risks are large, some are small, some are difficult to detect and can cause major problems down the road. It is impractical to try and focus on them all at once.  We have presented a matrix here for prioritization. This system is based on something you will see, for example, from the SEI CERT standards, and is useful for allocating security resources appropriately. Vulnerabilities are either more or less likely to occur, and will have a higher or lower priority based on, among other things, how much damage they can do to the system. |
| **4** | The next thing our policy does is break the major points of secure coding and architecture design into a set of principles. This makes our policy easier to define and enforce, and we can build a set of standards around these principles.  Validate input data  Heed compiler warnings  Architect and design for security policies  Keep it simple – we want to prioritize clear solutions that have less moving parts, which is less places to go wrong  Default Deny – limiting system access and data flow to what is absolutely critical for business operation  Adhere to the principle of least privilege – that is, no system user has more control of the system than what is needed  sanitize data sent to other systems  practice defense in depth  use effective quality assurance techniques  Adopt a secure coding standard – you will see we base a lot of this policy on the SEI CERT C/C++ standards. OWASP is another great resource for web application security. |
| **5** | Based on the principles we discussed, we now present a series of direct coding standards that can be applied during the development stage and onwards. They are listed here by priority.  At the top are larger vulnerabilities that can cause major system damage if left alone, such as SQL injection vulnerabilities, memory vulnerabilities (especially applicable to C and C++) At the bottom are smaller risks that are less likely to occur or cause long term damage, such as avoiding program termination from uncaught exceptions. still a risk but lower priority |
| **6** | Moving beyond that, this policy also continues applying defense in depth by looking at the security of the data itself.  In short, encryption of data is one of the strongest protection layers available, as even if data is breached or stolen, it is still a long way away from being usable. The policy touches on data in three states: rest, flight, and in use, which covers any scenario. Our policy enforces encryption at each state, using secure algorithms and proper key management. The nice thing here is that many quality solutions for secure data encryption already exist, and are relatively easy to implement in most frameworks. There is no excuse not to be doing performing proper encryption. |
| **7** | Next, our policy looks at system authorization. The ‘triple-a’ framework involves authentication, authorization, and accounting.  Authentication is who are you: are you a public user, admin, developer. The system must include a strong way of verifying who is who.  Authorization is what each user is allowed to do. So, as a public user for example, what resources am I allowed to access? This should, by default, be as few as necessary.  Accounting means keeping a record of system activity. This is important in the triple a scheme as it can alert you to potential attacks. |
| **8** | Another thing the policy covers is unit testing. So, after we have developed a new component or module using secure coding standards, does that alone make it safe? No. we would need to see the module in action, or, at runtime, first  Unit testing is a widely adopted practice and can contribute a lot to system security. We will briefly go over a few examples. In these examples we are using C++ along with the Google Test framework.  This below test is a positive test, that is, we expect it to pass given the input parameters. We use an ASSERT macro to perform a direct test, in this case, whether a newly created pointer is not null. On the right, we see the results when the test is run. |
| **9** | Another test example, this is again a positive test. This one is just slightly more complex, and shows how we can test a certain function with multiple different parameters in one go. We repeat the test four times with a different size, failing if any of the steps do not work as expected. |
| **10** | This is an example of a *negative* test. We feed this test bad data and expect the code to fail in some defined way. In this case, we expect it to throw an exception, which, as we see on the output, it does. |
| **11** | Finally, one more test. This is again positive, and involves setting some parameters, evoking the function in to be tested, in this case the erase function, and then checking to see if the parameters have changed as expected.  In summary, unit testing can be used to assure that what we have created is actually working as expected, not in theory, but in actual practice. |
| **12** | OK, then, we have all these standard and principles, are we on our own to implement them? No. We would like to use some tools to help us here, often referred to as automation. The illustration presented here shows a system lifecycle start to finish, and describes the specific steps where automation can be used. As we can see, it can be applied in just about every area of development and deployment. |
| **13** | Here we present some examples tools that can be used in 5 different major stages of the development cycle. We can start even in the planning stage, making sure we have an issue tracker in place, along with some way to help model threats effectively so we may design with them in mind.  In the building stage, it is highly recommended to use some sort of static analysis tool on the development code. These are great for making sure that code is compliant with security standards automatically.  Further, testing, as discussed, by using a unit testing framework is very effective. Integration testing frameworks are also useful for tying new components into an existing architecture. Finally, even into deployment and observation, tools exist that can help alert us to bugs, or we can use things like Docker to ensure consistency between runtime environments. |
| **14** | OK. This covers the main points of our security policy. Briefly we can discuss risks involved with shorting security measures. In the long run, it is not profitable or a good idea to do so. Any professional source you can find on the subject will tell you the same thing, that applying security early and throughout development provides measurable returns on investment throughout the system lifetime. |
| **15** | In summary, the main points of our policy are listed here.  Implement security early, and enforce it throughout the development lifecycle.  Use automation wherever possible to assist.  Use multiple layers of security. again, this is Defense in Depth, and is what this policy is based on. It is ineffective to rely on single sources of security alone  Finally, security training. This begins, of course, with defining a policy such as this one. Along with the principal of defense in depth, operator training is yet another security layer to add to a system. If you have some time, you can look into the Marriot International Starwood data breach, one of the largest in history, and that was made possible in part because some operator unknowingly installed malware onto a production system. |
| **16** | In conclusion  [ read slide as is ] |
| **17** | Thank you for watching this presentation. |