# CS 405 Project Two Script Brennan Reed

<https://youtu.be/MHsh6Eg8iZM>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Brennan Reed and today I’m presenting Green Pace’s security policy, complete with revisions and recommendations for future adoption. |
| **2** | First, we start with a diagram depicting a model of the many aspects of Defense in Depth. Our security policy will help create overlapping layers in a variety of these different facets that you can see on-screen, such as app security and network security, particularly those where we identified potential vulnerabilities and weaknesses. |
| **3** | Here we have a threat matrix where I’ve mapped our 10 coding standards into one of four categories each. We used likelihood of the threat and the severity of the threat to create this 2x2 matrix, which ends up consisting of: likely threats (which are likely but not as severe as other threats), low priority threats (which are both unlikely and low severity), unlikely threats (not likely, though they do have potential to be severe), and then the most dangerous – high priority – which are both fairly likely and fairly severe, thus potentially exposing us to major risk. |
| **4** | Those aforementioned coding standards are tied to our 10 major principles, which you can see here followed by the standards that are related to each. We have:  Validate Input Data – Which relates to Standards 1, 3, 8  Heed Compiler Warnings – Which relates to Standards 2, 3  Architect and Design for Security Policies – Which relates to Standards 1, 2, 5, 7, 9, 10  Keep It Simple – Which relates to Standard 5, 10  Default Deny – Which relates to Standard 4  Adhere to the Principle of Least Privilege – Which relates to Standard 4  Sanitize Data Sent to Other Systems – Which relates to Standard 4  Practice Defense in Depth – Which relates to Standard 6  Use Effective Quality Assurance Techniques – Which relates to Standards 6, 8, 9  Adopt a Secure Coding Standard – Which relates to Standards 1, 7 |
| **5** | The 10 coding standards we’ve been discussing are seen here in their entirety.  1. Do not define a C-style variadic function  2. Ensure that integer conversions do not result in lost or misinterpreted data.  3. Use the bounds-checking interfaces for string manipulation.  4. Do not store an already-owned pointer value in an unrelated smart ptr.  5. Sanitize string data passed to complex subsystems  6. Free dynamically allocated memory when no longer needed  7. A value-returning function must return a value from all code paths  8. Do not abruptly terminate the program  9. Handle all exceptions  10. Use a static assertion to test the value of a constant expression.  These are prioritized in accordance with the threat matrix, looking at a combination of likelihood and potential damage/cost – those are at the top, working down to less likely or less dangerous (or both) at the bottom |
| **6** | Our encryption policies can be seen here. We have three important pieces here, one for each of the major data states: at rest, in flight, and in use.  Encryption at rest involves using encryption on data that will be stored (hence at rest), thus requiring decryption to read the sensitive data and making it significantly harder to access/use. We will encrypt any data that is to be stored/retained by using symmetric cryptography. This will keep any sensitive, confidential, and/or proprietary information safer from malicious actors.  Encryption in flight refers to encrypting data as it is transmitted. We will avoid untrustworthy file transfer mechanisms, use TLS on our web connections, and use a VPN for remote access to our network. This will help ensure that our data is handled/transmitted safely throughout our network, avoiding intentional/malicious loss of data during transfers.  Encryption in use refers to protecting data that is in use – such as actively being handled in memory/code/databases. We will protect data in use by using Multi-Factor Authentication and the Triple A framework to avoid malicious actors accessing data, as the ability to monitor for any intentional/accidental mishandling of data in use. |
| **7** | As previously mentioned, Triple-A policies are an important part of our security policy and our Defense in Depth strategy. AAA refers to Authentication, Authorization, and Accounting.  Authentication is the first step in the AAA security process – how do we identify a user and ensure they are whom they claim to be. For our organization, each user must have a unique and valid password and user account to login.  Authorization refers to determining what activities, resources, or services a user is permitted to use/access. Once a user has been authenticated, they are assigned the appropriate user properties/permissions that describe what they are authorized to perform. This can be seen in the various access roles for our organization, such as base access will being able to make changes to the database only, whereas other roles have additional functions (like adding users or changing permissions).  Accounting refers to a variety of ways that users/resources are monitored in the system. Our employees’ access to files is monitored to ensure the safety of our resources – we can monitor when files were accessed, by who, and changes they made to each file. This helps ensure that those with authorization are using their roles appropriately and not mishandling data intentionally or accidentally. |
| **8** | As we’ll talk about more in the next few slides, unit testing is a very important tool in our secure coding arsenal. Unit tests help us isolate our code so that we can very modularly ensure that each piece is working as intended. So, I’ve chosen to demonstrate unit tests for one of our coding standards, STD-002-CCP, which is “Ensure that integer conversions do not result in lost or misinterpreted data.” So, we’ll have four tests to follow that show both passing and failing tests of how to adhere to this standard, which will help reinforce the value of using unit testing.  We’re not going to go too far into the weeds as far as the specific code for each test, as we simply want to see how they’ll fit our framework. Test 1 shows that the time\_t now variable is set to time(NULL). We then expect this to return true that now and (time\_t)-1 (casting it to a *real* type) are comparable – they are. |
| **9** | Onto the second slide for unit testing, we see an example showing that now does NOT EQUAL -1 however, because it was not cast properly. The equal/lt/gt/ne tests allow for binary comparisons as seen here. |
| **10** | Onto the third slide, we see the inverse of the last slide – if we expect now to equal -1, then our test will show a failing test because as we’ve already indicated – they do not. This is how negative tests can help inform our code. |
| **11** | And finally, the fourth example shows that though test 1 showed that now and (time\_t)-1 evaluate to true when compared, when using binary comparison they are not equal since this EQ test failed. So, this shows the issues with this coding standard, how Boolean Condition tests differ from the Binary Comparison tests, and a second example of a negative test. |
| **12** | For our discussion of automation, take a brief look at this diagram of the DevSecOps pipeline and the various stages of this process. |
| **13** | DevSecOps secures code and the overall SDLC process by adding security at multiple/all stages of the process. Specifically, there are a few tools that we identified as noteworthy. This includes using a Static Code Analysis tool, such as Cppcheck or SonarQube in conjunction with automation so that the “Verify and test” phase always is seeking these vulnerabilities so that engineers can adjust code as needed. It also includes adopting Test-Driven Development, which means setting up these tests in the Design phase – adopting an automated unit testing tool like Parasoft would help us ensure that we follow this methodology and implement unit tests for our code every time. |
| **14** | There are risks and benefits to either adopting these amendments to the policy and making immediate changes, or choosing to wait to implement changes/fix threats. However, I strongly advise that we act now, because I believe those pros outweigh the potential cons. Let’s look at how the two strategies break down in a bit more detail.  If we act now, we have costs with getting everything up and running – ranging from implementing automation software to staff training time, but it helps us avoid potentially costly issues with bad publicity and reputation, malicious actors accessing our data, and more.  If we choose to wait to implement changes, we continue business as usual which is more profitable in the short-term as there’s no need for changes, training, new software, new models to follow, etc. However, we run the risk of incidents particularly related to the aforementioned standards on the threat matrix in the interim. A single threat could result in a massive leak or hack that could cost the company tremendously, whether that is in reputation or financials.  So, I believe it’s better to be proactive and get ahead of these potential risks, which is why I advise us to act now and make the changes which will shore up potential weaknesses, as seen earlier in the presentation. However, this strategy does also lack in long-term coverage – we’re addressing immediate concerns, but how quickly will we address those unlikely issues or will they be swept under the rug? We’ll need to coordinate follow up to see that the lesser threats are addressed (not just the high priority ones) and that new issues are routinely added to a new threat matrix to keep us ahead of the curve of new potential threats. |
| **15** | In addition to my recommendation to act now, I also have several other important recommendations.  The biggest gap in this policy is consideration for our training and processes for updates in our security policy. How and when do we train staff? How often? Do we provide support for developers to learn the newest tips for writing secure code? The answer to many of those is not as clear as it should be… So, we should commit to offering routine training for staff, create a multi-pronged approach for disseminating new information and training when new policies are adopted, and promote self-discovery and continued learning opportunities for staff. I believe this focus on training staff in regards to security policies and future changes will pay massive dividends both now, and going forward.  Similarly, we do not have a concrete disaster plan laid out. If all these efforts unfortunately fail – what do we do when something bad happens? Most of our efforts should be spent on prevention, but we do need to adopt and train on a mitigation plan for any issues that may arise. This will allow us to be prepared and rehearsed for roles in such a situation. |
| **16** | In addition to the previous slide’s recommendations for a better security training plan/procedure and a disaster mitigation plan, I want to highlight a few of the coding standards that we should be taking action on before I wrap up the presentation. Our highest priority threats are:  STD-001-CCP or Do not define a C-style variadic function  STD-002-CCP or Ensure that integer conversions do not result in lost or misinterpreted data  STD-003-C or Use the bounds-checking interfaces for string manipulation  STD-004-C or Sanitize string data passed to complex subsystems  STD-009-CCP or Do not store an already-owned pointer value in an unrelated smart pointer  As these standards carry the most risk, including these in our policy and ensuring staff are appropriately handling/avoiding these types of vulnerabilities will allow us to continue operating safely and avoiding those disaster-type scenarios I alluded to previously. Coding securely keeps us safe, which keeps us operating and keeps our clients happy. As the old adage states, an ounce of prevention is worth a pound of cure! |
| **17** | You can see our references on this slide, thanks to those sites for providing context and information to help support this policy.  Thank you all for listening to my recommendations for Green Pace’s Security Policy! |