Southern New Hampshire University

CS 405 Project Two Script Template

Tabitha Tallent

Project Two: Security Policy Presentation

February 23, 2025

Narrated presentation - https://youtu.be/KGuWSSJoVpY

Slide Number	Narrative
1	Hello everyone, my name is Tabitha, and this presentation outlines the basics of a security policy for Green Pace.
2	Defense in depth utilizes layers of defense methods built from the research and development stage and forward throughout an application's lifecycle. This policy uses DiD to define acceptable coding standards, in accordance with C++ security best practices for maintaining the safety of data being handled, and the overall security of the software.
3	Here we have a threat matrix with ten categorized coding standards, don't worry about understanding them right now, we'll go more in depth very shortly. This matrix displays four priorities of threats. First, likely threats which are most common, but also only have low to medium severity. Secondly, high priority threats which are also likely to happen, but have a higher repercussion severity. Next is low priority threats which are both unlikely to occur and have a low severity. Finally, there are a couple of unlikely threats, but these have a medium to high severity should they occur. I'll give you a better look at what these threat IDs mean in a few minutes, but this matrix can be referred to at a glance if necessary.
4	These are ten security principles that can, and should, be applied to any application, and many of them line up with SEI CERT secure coding standards for C/C++ which is used for this project. Validate input data, all data should be checked to ensure it meets the expected parameters, and does not include any unexpected input that could equate to arbitrary code being executed. Compilers do are not as useful as many testing tools, but they do provide important warnings and errors, and anything flagged by the compiler should not be dismissed lightly as security vulnerabilities can be introduced this way. The next several principles, architect and design for security policies, keep it simple, default deny, and adhere to the principle of least privilege all provide easy

Slide Number	Narrative
	guidelines for creating a secure application. Simple code, designed with security in mind from the start, is also not only less likely to be vulnerable, but is also easier to maintain overall. Meanwhile, not every user needs access to everything, will go more into access controls and authorization in a bit, but the short version is deny it all, then write the policies to allow what's necessary. Data in transit is subject to man in the middle attacks and it should be sanitized the same as input data. We covered defense in depth a bit already, but layering security can include applying all these principals, regular security audits, firewalls, and logging. Finally, use effective quality assurance techniques and adopt a secure coding standard, testing is vital not only for checking basic application functionality, but also for checking the possibility of vulnerabilities such as SQL injection or denial of service opportunities, these are good examples of adopting a secure coding standard.
5	Remember that threat matrix earlier? Here is what those ID strings referred to. Now these are ordered by priority, which was determined by combining the likelihood of occurrence weighted by the severity. The first four here, guarantee that storage for strings has sufficient space for character data and the null terminator, exclude user input from format strings, do not access freed memory, and do not dereference null pointers are all likely to happen and have a high severity ranking. The next one, ensure that operations on signed integers do not result in overflow is unlikely to happen, but has a high severity should it occur. The next three are all medium severity in terms of repercussions, but have varying likelihood of occurring. Ensure your random number generator is properly seeded is important because this threat is likely to occur. Free dynamically allocated memory when no longer needed is probable, and handling memory allocation appropriately will show up later. Understand the termination behavior of assert() and abort() especially since assertions are not usually included in production builds, this mishap is unlikely to happen however. Finally, the last two are low priority even though the possibility of not handling all exceptions is probable, however creating incompatible declarations of the same function or object is unlikely.
6	Encryption is the modification of data, generally using a key or algorithm of some sort to ensure it cannot be read by anyone other than the intended party. While many encryptions can be cracked, it takes time, and makes the data less easy prey. Three simple encryption policies should be applied, encryption at rest emphasizes that any data being stored, even long term storage data, should be encrypted. Data in flight, or data be transferred from one place to another should also remain encrypted, this is another opportunity for man in the middle

Slide Number	Narrative
	attacks if not adhered to. Finally data in use should remain encrypted the entire time, even when being updated.
7	Moving on is what is referred to as Triple-A, and it outlines security expectations for handling users and their activity, and monitoring the system appropriately. Authentication is used to verify that user is who they say they are, this is often handled as a set of login credentials, and may include multiple methods of authentication before access is granted. Authorization controls what a user is allowed to do within the system, again remember to follow the principle of least privilege, there is no reason a standard user should have access to anything other than their personal information and valid transactions. Don't want to have to worry about attempted access to other users information, or to system controls such as deleting databases or taking the system offline. Finally, implement a logging system, this should be mostly automated, to record any activities performed or requests made when they happen and from where they were initialized. Combining this with a system that flags certain activities, say an attempt to access an unauthorized part of the application, can help create an early warning system for potential malicious activity. Furthermore, logs can be reviewed in the event of a breach to help narrow down possible causes.
8	Now let's look at a few unit test examples. This first one handles memory allocation, and it tests to verify that calling erase with the start and end of a collection. Ensuring that erase works appropriately here is an important step in making sure memory is deallocated appropriately.
9	This next test regards memory safety and it checks to verify that reserve() increases a collection's capacity, but not the size of the collection itself. The system needs to be able handle resizing to prevent possible memory leaks and accidental buffer overflows.
10	This test deals with resource handling alongside memory by ensuring that resize() appropriately increases the size of a collection, not to be confused with capacity which we already looked at. Adding entries and making sure the system responds as expected demonstrates that resources are being handled correctly.
11	This last test is a bit different, it checks to make sure that the entire process will exit gracefully if access to a removed element is attempted. This ensures that the system can handle terminal errors, which prevents unexpected behavior from accessing unallocated memory within the application.

Slide Number	Narrative
12	Moving on, this is the DevSecOps pipeline, and I will soon share a few places to insert automated security protocols. First note that this pipeline starts all the way at the planning stage for implementing security centric practices.
13	Automation is helpful for encouraging security practices, set it and review as necessary often times once a year is sufficient, unless something occurs to designate otherwise. Static testing can begin at the build stage, here code has been created, with security in mind, and tests for common vulnerabilities can be run, OWASP is a good resource for this stage. As the application starts to come together and the issues discovered by static testing have been mitigated more dynamic testing can come in, these tests should check program functionality alongside buffer overflow and memory and resource handling as previously demonstrated. The transition and health stage is a good place to setup firewalls and perform penetration testing. Finally the monitor and detect should include extensive logging and automated alerts. Alerts should be reviewed, but they can be false positives, such as a user mistyping something in the URL, or clicking on something that they do not have access to (remember that this attempt should fail anyway based on the principle of least privilege).
14	Do not leave security until the end! Security is a critical aspect of any application, especially one handling user data. Let's cover a few additional benefits of prioritizing security. It is much easier to implement security from the start, practicing secure coding can lead to better codebases for easier maintenance, and improved coding skills for developers. On the other hand, the longer you wait to implement security, the more difficult it can be to add into an already functioning codebase and there is the very real possibility of a breach occurring. Once again, it is just better to code securely from the start.
15	Just a few more things, training your personnel is vital to the success of this security policy, people cannot adhere to a policy they are unfamiliar with. Maintain clearly established, and enforced, repercussions (this often includes up to termination) if personnel is found to have violated the security policy. Security vulnerabilities and possible attacks are always changing, perform regular security audits and updates to the policy. Once again, implement access control and authorization protocols that align with default deny or the principle of least privilege, and don't forget to log all activity for easy reference and potentially early warning. Finally, at the end of the day there is still the possibility an unknown, or new vulnerability, will be exploited within your system, be

Slide Number	Narrative
	prepared for handling a security breach.
16	Finally, I've listed some simple, important principles and methodologies from this presentation for you to leave with fresh on your mind, perhaps a touch repetitive, but the security concerns cannot be emphasized enough. Practice secure coding from the start, do not wait until the end! Save and adhere to the then coding principles provided at the beginning. I have also provided the SEI CERT website which is where all the coding standards in this policy originate from, and many more. Bookmark it, reference it, and don't forget that to find language appropriate coding standards from other sources if necessary. Train your people. Developers. Management. Associates. Everyone on every level should be familiar with the security policy, the expectations of their role in adhering to it, and any potential penalties. The DevSecOps pipeline has become a popular method for implementing security from the start, don't overwork yourself trying to reinvent the wheel, use it, that is what it is for. I hope this presentation provided a good outline of the Green Pace security policy as well as encouraging secure programming, thanks for your time today!