Gracie Ream

8/16/2025

Project Two: Security Policy Presentation

<https://youtu.be/RAJA8HdY6Tw>

Introduction:

Hello everyone, my name is Gracie Ream and I will be presenting the Green Pace Security Policy.

Overview:

The Green Pace Security Policy will implement core security policies like secure coding standards, Authorization, authentication and accounting standards, and encryption standards. This security policy will involve using defense in depth standards and multiple layers of security to ensure that there are no attacks or vulnerabilities later on.

Threats Matrix:

The threats matrix table includes likely, unlikely, priority and low priority. Under likely I have data value standard which says to not access an object outside its lifetime and exceptions standard that describes handling all exceptions. Under the unlikely section I have the containers standard that ensures all container iterators are within valid range, data type standard that says to obey the one definition rule, and the object-oriented programming standard, describing that constructor initializers should be written in canonical order. For priority I have SQL injection to prevent any injections, string correctness that describes not creating a string from a null pointer, and memory protection that explains to not access freed memory. For low priority, I have assertions which is to use a static assertion to test the value of a constant expression, and integers-use correct integer precisions. By using automation, we can detect these vulnerabilities before deployment.

10 Principles:

The first principle is validate input. For this principle I included the standards Data value, string correctness, SQL injection, memory protection, containers and integers. This validates input from any untrusted sources. The second principle is heed complier warnings. My standards for this one included data type, data value, string correctness, memory protection, assertions, containers, object oriented programming, and integers. This principle will compile code using the highest warning level. The third principle is architect and design for security policies, meaning create software that will implement and enforce security policies. The standards that I included in this principle were assertions and exceptions. Next we have keep it simple. This means keeping code clean and simple to reduce errors. I included data type, string correctness, assertions, containers, object oriented programming and integers for this one. The fifth principle is default deny which I did not include any standards in, but it is still a good principle to include as it bases access decisions on permissions, meaning access is denied by default. Next, we have the adhere to the principle of least privilege principle, meaning every process should execute with the least set of privileges needed. I included standards memory protection, integers and containers for this one. The seventh standard is sanitize data sent to other systems, which I included standards SQL injection and containers in. The eighth standard is practice defense in depth. This means to include multiple layers of security so if one fails, another one is already in place. The standards I included for this one were data value, string correctness, SQL injection, memory protections, exceptions, containers and integers. The ninth standard is use effective quality assurance techniques. This standard helps in identifying and getting rid of vulnerabilities. I included standards integers, containers, exceptions, assertions, SQL injection and string correctness. The last standard is adopt a secure coding standard. I included data type and object oriented programming language standards for this principle.

Coding Standards:

The first coding standard is data type, this is important to ensure that variables store the correct information. Wrong data types can cause logic errors or buffer overflows. Next is data value, which validates the values assigned to variables to prevent invalid or malicious input. If ignored, unexpected inputs can crash systems or create vulnerabilities. The next standard is string correctness which ensures strings are properly formatted and safe to use. If not used, SQL injection can occur as well as buffer overflow attacks. The next standard is memory protection which is important to use because it prevents wrong access to memory. Without this, memory corruption can occur. SQL injection is next, and should be used to validate database queries, if ignored attackers can read, modify and delete data. Exceptions allow for proper exception handling without exposing information. Without this, programs can crash or information can be leaked to attackers. Next, containers ensure that collections are correctly managed. Invalid containers can cause crashes and memory corruption. Object Oriented programming ensures maintainable code. Poor design can result in unauthorized access or sensitive data being exposed. Last, integers is important to prevent overflows and underflows and without this standard, memory corruption is possible.

Encryption Policies:

There are three encryption policies, encryption at rest, encryption in flight and encryption in use. Encryption at rest is applied to any data stored on a physical or virtual machine. It is used to prevent breaches that involve any physical theft or inappropriate access to a system. Typically it is used when any long-term data is stored. Encryption in flight is applied to any data in transmission over networks. An example of this is between client and server. Encryption in flight is typically used when sensitive data is being sent over networks. Encryption in use applies to data in memory processing and protects data from exposure during active processing.

Triple A Policies:

The triple A Policies are authentication, authorization and accounting. Authentication is the process of verifying the identity of a user on a system. This is incorporated through usernames and passwords, biometrics or multi factor authentications. Authorization is when access is either granted or denied to certain users or their rights. This can be done through ABAC, Attribute Based Access Control and RBAC, Role Based Access Control. For RBAC, each role is defined and has specific permissions. For ABAC, access is granted based on attributes and not just role. The authorization policy requires permission for any new user and is used to minimize risk. The accounting policy is the act of tracking user activity and recording it. This is implemented through audits and logging.

Unit Testing:

The first unit test that I did was to see if the collection threw an exception while accessing an index beyond its size. This test prevents attackers from reading and writing outside of allocated memory. This test successfully passed. The next test was to see if the collection can store five values correctly and verifies that the collection grew when elements are added. This test also passed. One more test I implemented checked to see if resizing increased the collection’s size. This makes sure that any new elements are accounted for. This test also passed.

Automation summary:

Within the devsecops diagram, the stages are plan, code, build, test, deploy, monitor, respond and maintain. The security automation happens within the code phase in terms of compiler warnings and memory checks, the build stage as compiler and static checking, the test phase as unit tests, the deploy stage as container scanning and security policy enforcement and last in the monitor stage as alerting for any runtime threats.

Tools:

DevSecOps is the practice of implementing security into every stage of development. By doing this, we can catch issues earlier and reduce costs while delivering a safer application. Specifically, we can use tools like Parasoft C/C++ and Clang in the plan and design phase to help catch unsafe code patterns and memory errors. SonarQube, Code Sonar and Helix QAC can be used in the build and test phase to preform static analysis testing and check for vulnerabilities. Parasoft Insure++ is good to use once deployed because it monitors memory use and catches runtime errors. Implementing monitoring and logging tools after deploying keeps track of the system’s behavior and exceptions.

Risks and benefits:

There are benefits and risks to both acting now or waiting to act. The benefits to acting now are greater because it overall reduces vulnerabilities, reduces costs because you are spending less money in the long-run fixing vulnerabilities and attacks, and it builds a security culture among the development team, meaning layers of security will become more important to the team once incorporating this. The only risk to acting now is that there are higher costs upfront to implement the tools and training needed. The benefit of waiting to act is that there are no immediate costs for training and tools needed on security. The risks of waiting to act are possibly exposing the system to more vulnerabilities down the road as well as higher costs to fix them later on.

Recommendations:

Overall, this is a starting point for the Green Pace Security Policy and can be added on and updated further. Coding principles are defined but they are static and attacks and vulnerabilities are always evolving. This means what works now, may not work in the future. A real world example of this is the 2020 SolarWinds supply chain attack, where traditional coding standards did not protect against an attack, and updates were needed to reincorporate a safe system. My recommendation so far is to do continuous review and policy updates to prevent this.

Conclusion:

To summarize the green pace security policy is to approach development with multiple layers of security and secure coding practices. Following a defense in depth framework will protect the system more than traditional methods. Enforcing least privilege rules limits insider threats and limits access. Using testing tools, we can catch vulnerabilities early and act on it. Together, these practices ensure a safe, secure and reliable application to stay ahead of threats.