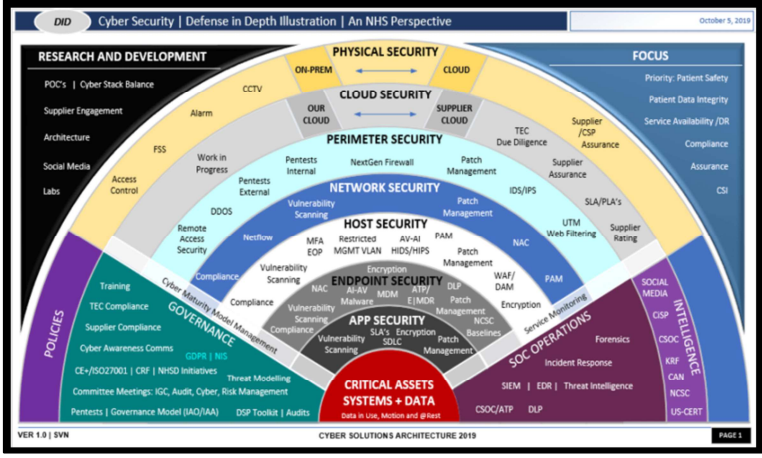
Hello, my name is Dalton Rose and today I'll be proposing a security policy.

OVERVIEW: DEFENSE IN DEPTH

Defense In Depth – a security strategy where multiple layers of security measures are used to secure information systems.



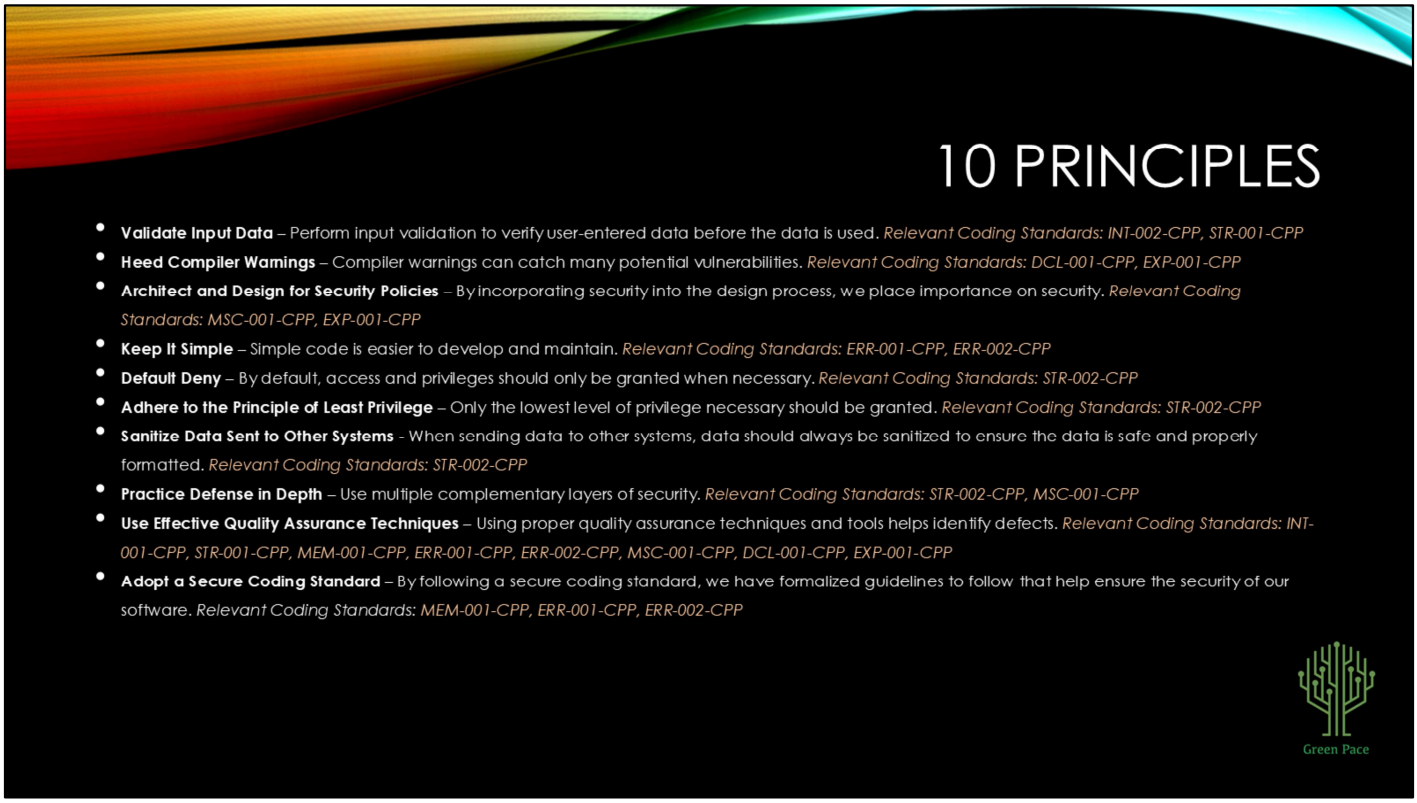
This security policy outlines the security practices at Green Pace including secure coding principles, secure coding standards, encryption policies, automation, and more. By following this policy, we can ensure the security of our software. Each of these elements work together to provide defense in depth.

Green Pace

Defense in depth refers to a security strategy where multiple layers of security measures are used to secure information systems.

This security policy is an example of defense in depth, and we'll cover security practices like secure coding principles, secure coding standards, encryption policies, automation, and more.

In this Threats Matrix, you'll see the coding standards I'll outline divided by their likeliness to occur and how high priority they are. We have coding standards that are likely and unlikely to occur and some that are particularly high or low priority.

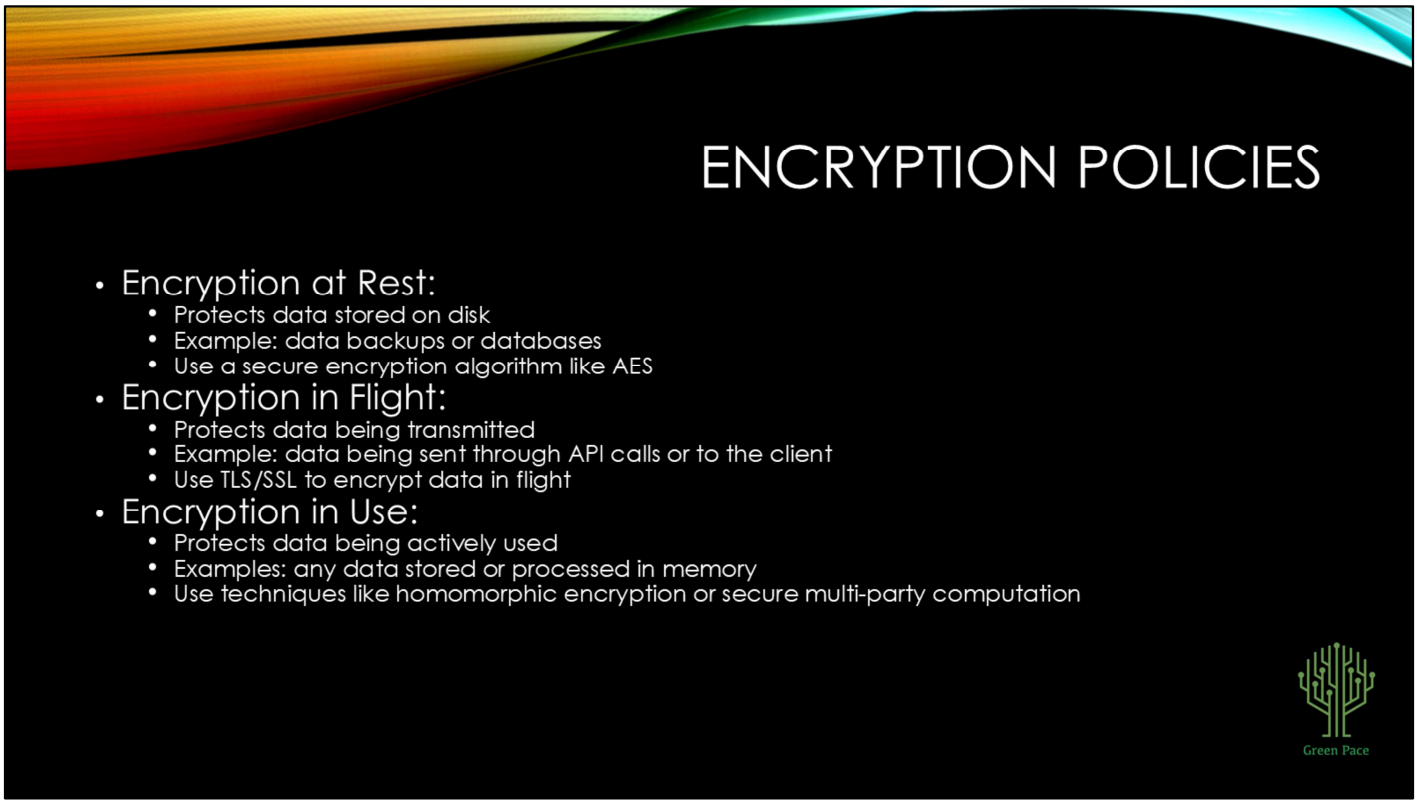


Here are the ten principles of secure coding and their definitions. You'll also see which of the identified coding standards are most relevant to each secure coding principle.

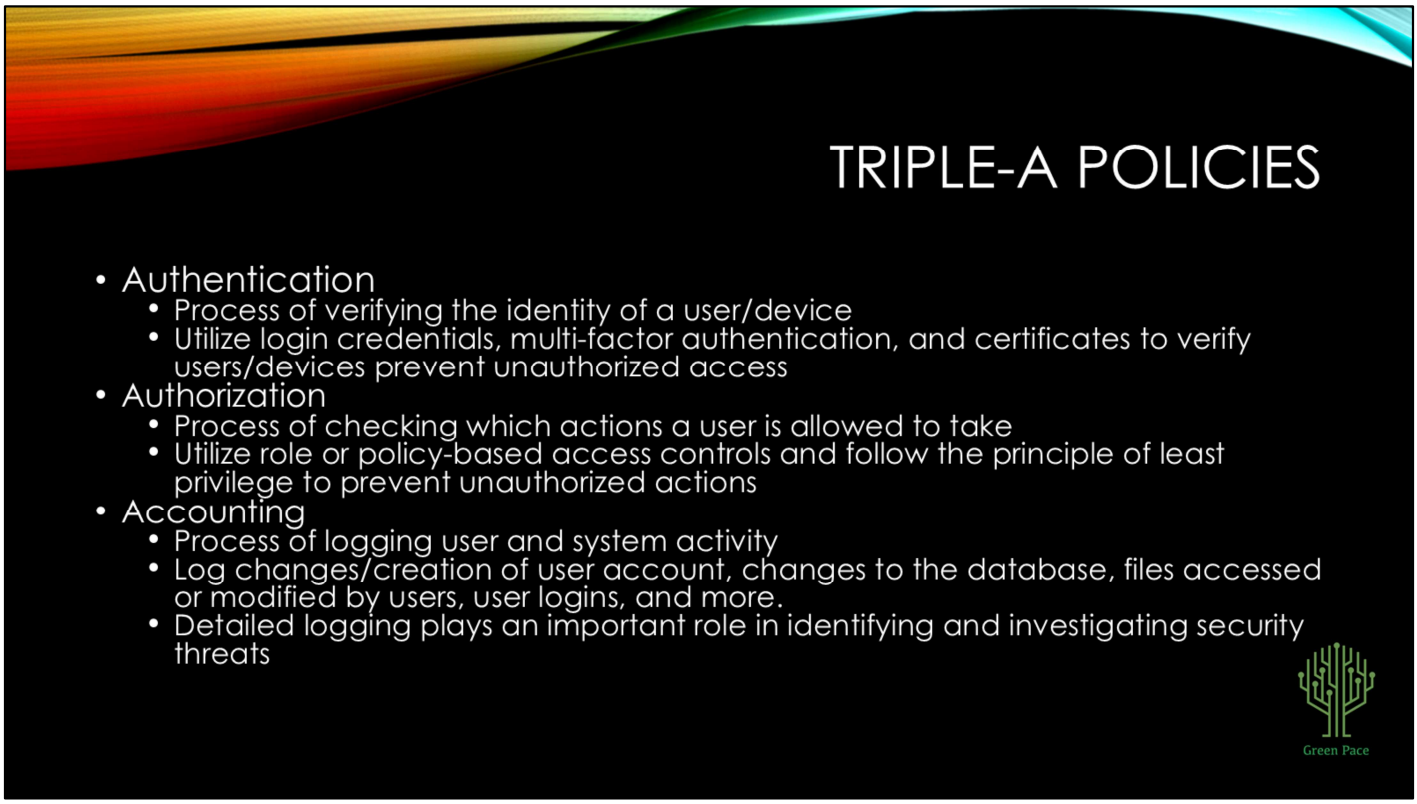
CODING STANDARDS

Standard	Severity	Likelihood	Remediation Cost	Priority	Level
INT-001-CPP	Medium	Unlikely	Low	High	L3
ERR-001-CPP	Medium	Unlikely	Medium	High	L3
MSC-001-CPP	Medium	Unlikely	Low	High	L3
INT-002-CPP	High	Likely	Medium	Medium	L2
STR-001-CPP	High	Likely	Medium	Medium	L2
STR-002-CPP	High	Likely	Medium	Medium	L2
ERR-002-CPP	Low	Likely	Medium	Medium	L2
DCL-001-CPP	High	Unlikely	Medium	Medium	L2
MEM-001-CPP	High	Likely	Low	Low	L1
EXP-001-CPP	High	Likely	Medium	Low	L1

In this table, you'll see the ten coding standards I've identified ordered by their priority. Their priority was determined by considering factors like severity, likelihood to occur, and remediation cost.



There are three types of data encryption we need to focus on.
First, encryption at rest. This refers to encrypting data that is stored on disk. Some examples of this are data backups and databases. To secure this data, we should use a secure encryption algorithm like AES.
Secondly, encryption in flight. This refers to encrypting data that is being transmitted. Some examples of this is any data being rendered to the client or data being sent through API calls. To secure this data, we must ensure we are using TLS/SSL.
Finally, encryption in use. This refers to encrypting data that is actively being used. An example of this is data being stored or processed in memory. We can use techniques like homomorphic encryption or secure multi-party computation to secure this data.

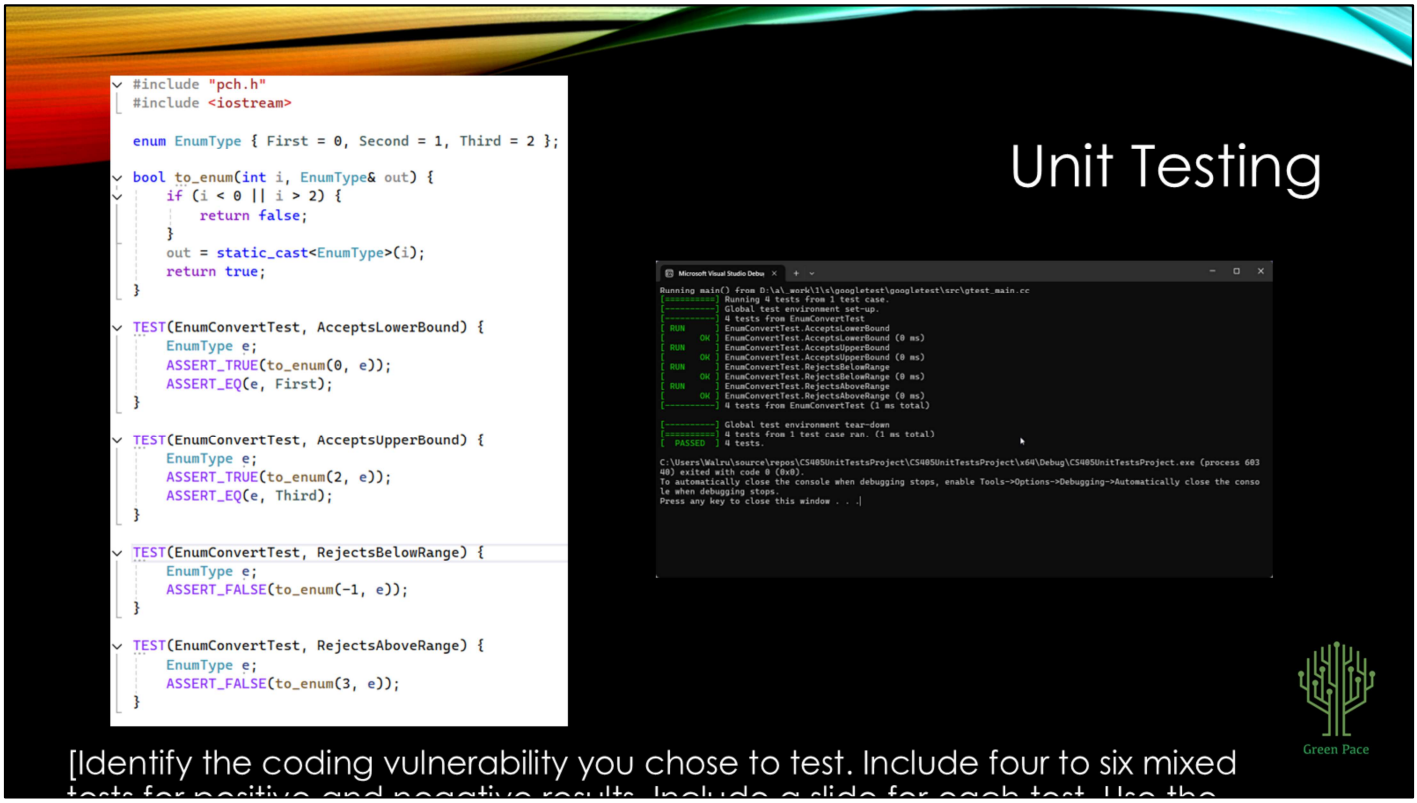


Triple-A refers to Authentication, Authorization, and Accounting.

Authentication refers to the process of verifying the identity of a user/device. We can utilize login credentials, multi-factor authentication, and certificates to verify users and devices and prevent unauthorized access.

Authorization refers to the process of checking which actions a user can perform. We can utilize role-based or policy-based access control and follow the principle of least privilege to limit users to only accessing what is strictly necessary.

Accounting refers to the process of logging user and system activity. We can log activity such as changes to user accounts, database changes, files accessed, user logins, and more. Keeping detailed logs is important for identifying and investigating security threats.

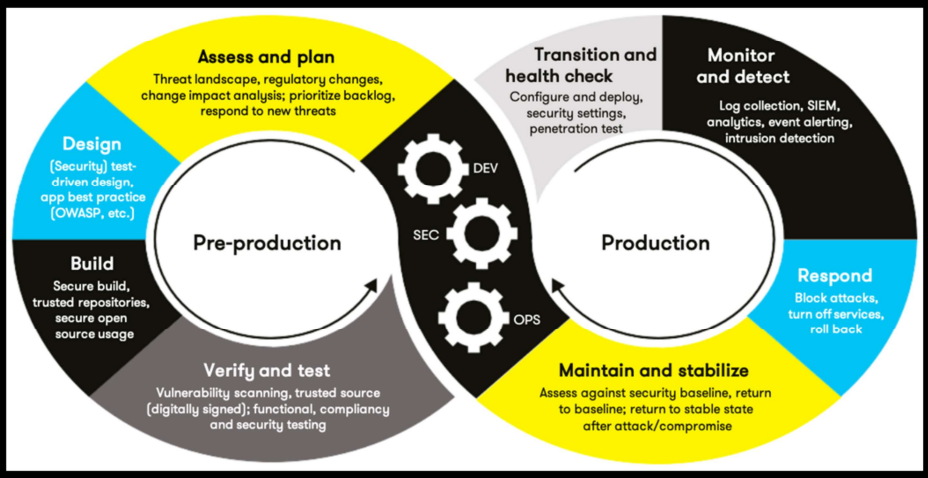


As an example, I created a simple function which converts an integer into an EnumType which includes logic to check whether the enum value is in range.

There are also four unit tests: two positive tests that test that an integer is successfully converted within range and two negative tests that test that when the conversion is performed with an out of range value, the conversion returns false.

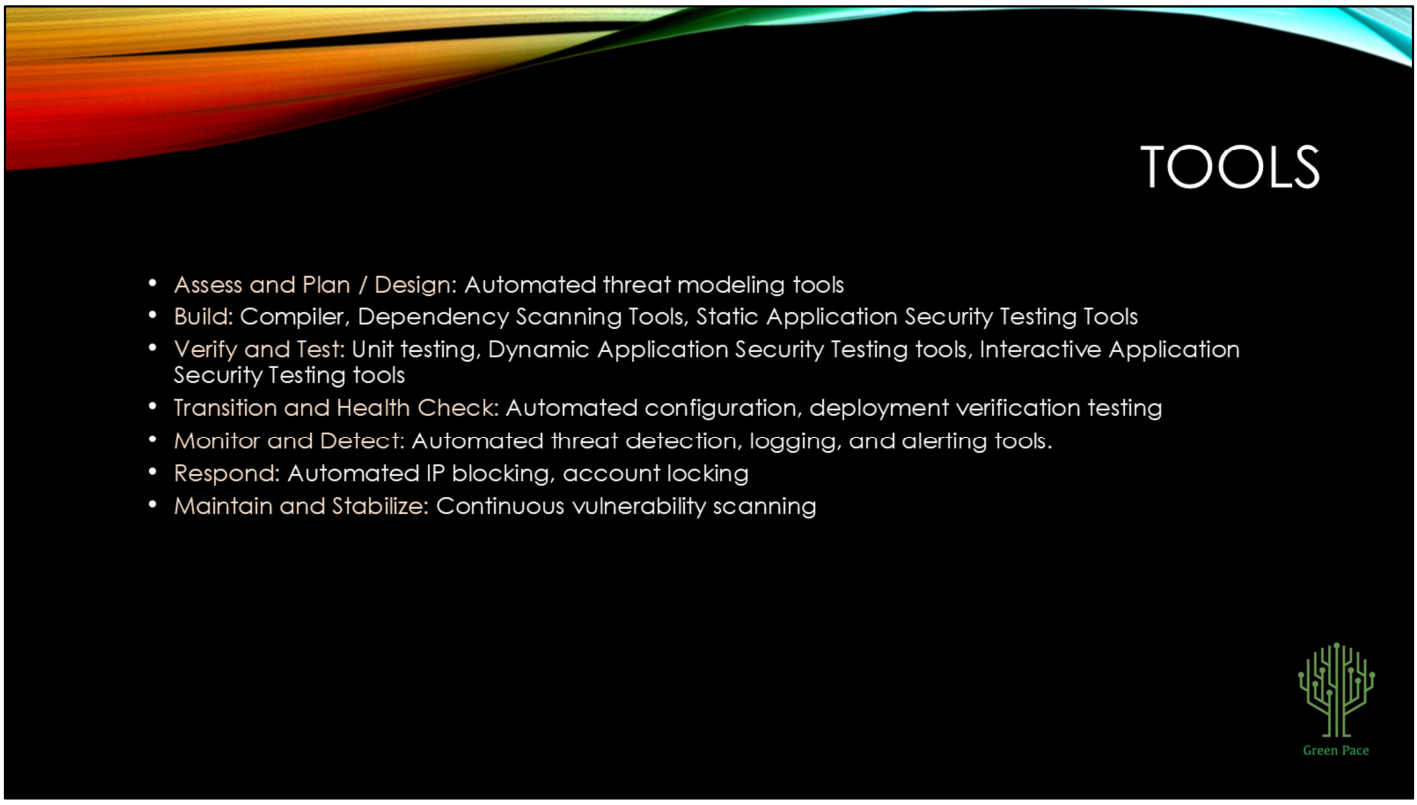
As you can see, all tests pass successfully due to the error handling in the EnumType conversion function.

AUTOMATION SUMMARY



Green Pace

Here is a diagram showing the DevSecOps pipeline. Security automation tools are present throughout each phase of the DevSecOps pipeline. In the next slide, you'll see how automated security tools are used in each phase of the DevSecOps pipeline.



In the Assess and Plan and Design phases, we will use automated threat modeling tools to identify potential threats before code is written.
In the Build phase, we use the compiler and dependency scanning tools to ensure secure code compilation and discover potential vulnerabilities in our dependencies.
In the Verify and Test phase, automation tools will be used to look for violations of the coding standards, verify required security functionality, and scan for other potential vulnerabilities.
In the Transition and Health Check phase, we will utilize automated tools to manage configurations and deploy our code to ensure our production environments are securely configured.
In the Monitor and Detect phase, we will utilize automated threat detection, logging, and alerting tools to monitor our systems for potential threats.
In the Respond phase, we will utilize automated tools to take defensive actions like revoking credentials if unauthorized access is suspected.
In the Maintain and Stabilize phase, we will continue to use automated tools to scan for potential vulnerabilities.

Currently, we have a few problems relating to the security of our software. Firstly, we do not have a formalized security policy outlining our security best practices. Secondly, we have not established secure coding standards. Finally, we are not fully utilizing security automation tools to secure our software.

To solve this, I propose implementing a formalized security policy outlining our security best practices. This security policy will also establish and detail the secure coding standards we will follow going forward. Additionally, we should take full advantage of security automation tools.

Acting quickly is important as leaving potential vulnerabilities open is dangerous. It could result in financial or reputational damages to our company. We may also be putting our user's data at risk. Additionally, security should not be an afterthought but something we practice throughout every phase of development.

Our existing security policy has several limitations. Firstly, the policy is not formalized. As our company expands and new developers are onboarded, it is important that there are standardized security policies and standards we are using in our projects. Secondly, there is insufficient documentation. While the development team may be following security best practices currently, the details of that are not clearly documented. Finally, the current security policy treats security as separate from development. As we move from DevOps to DevSecOps, we must revise our security policy to more deeply integrate security into the development process.

Looking forward, our security policy should not remain static but should continue to evolve to meet new threats and follow security best practices. We also must adapt to fully take advantage of security automation tools. If a lapse in security occurs, we must review exactly what went wrong and reevaluate our policies to prevent the same mistake from occurring. Additionally, we should implement regular security training.