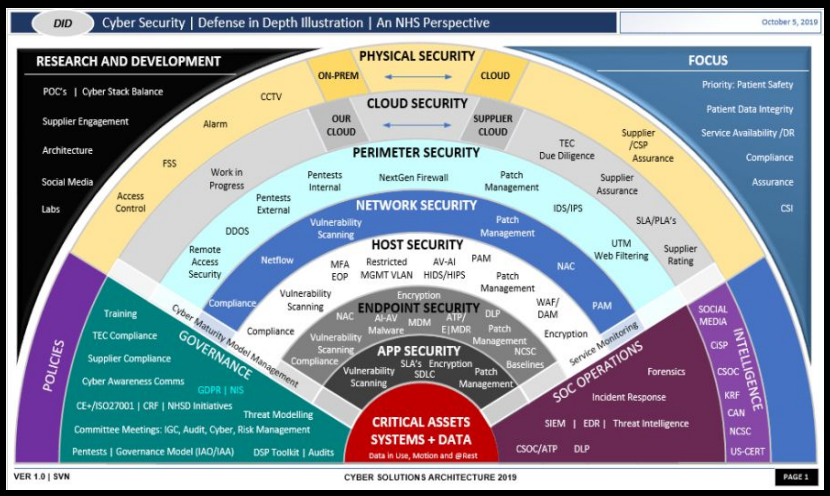
[Title Slide:] Hello and welcome, my name is Kyle Dale and today I will be going over the new Security Policy. The new policy will be implemented to prepare us for the upcoming audit, and smoothen the transition from DevOps to DevSecOps. We will cover what this transition will entail and how it will affect procedure and practices here at Green Pace.

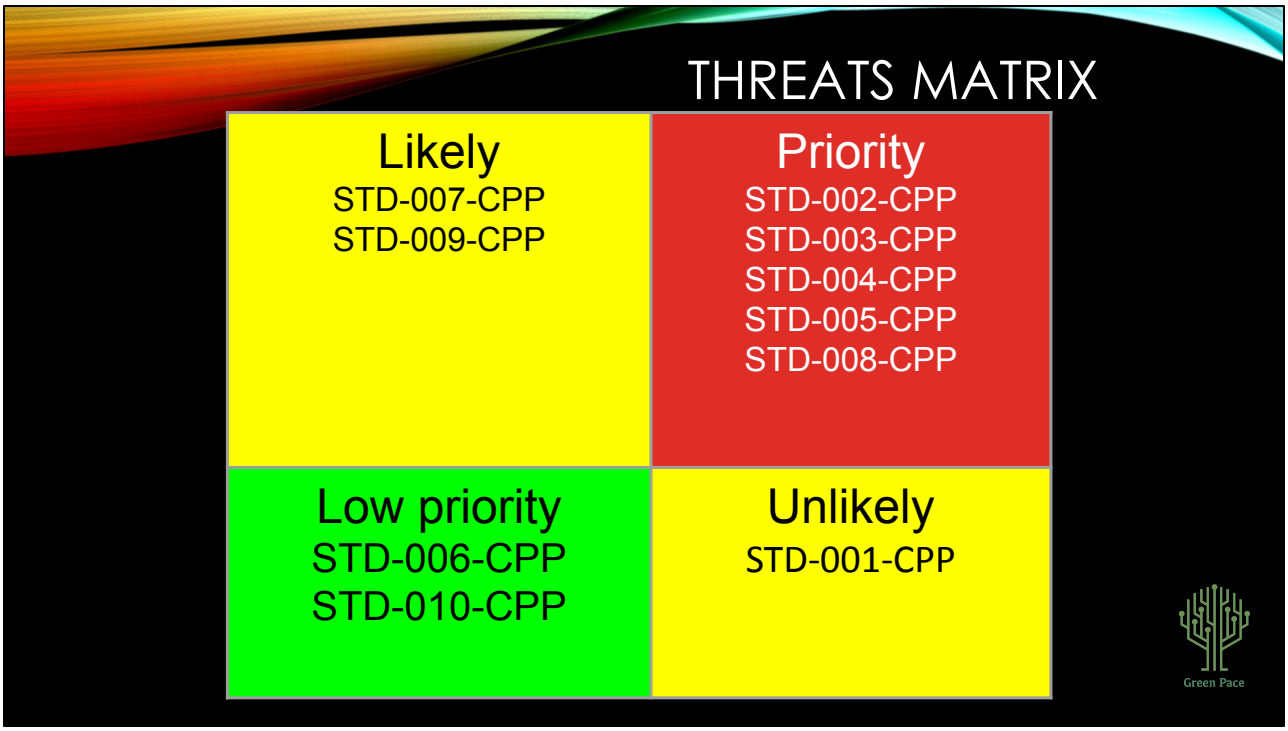
OVERVIEW: DEFENSE IN DEPTH



Green Pace

[**DiD Overview:]** To begin, what is the security policy, why is it needed, and how does it help? The security policy defines the core security principles; C/C++ coding standards, Triple-A standards, and encryption standards and policies. As we all know, the landscape of cyber security is a constantly evolving field, with new vulnerabilities and avenues of attack being consistently exploited. In order to meet the shifting threat (and pass our audit) we must alter the way we consider security from the ground up, and adopt DevSecOps. DevSecOps can be described as the practice of integrating security testing at every stage of the development process, including tools and processes that encourage collaboration among development teams to build software that is both safe and efficiently delivered (Amazon, 2023a). Through DevSecOps we can enforce the philosophy of security by design, and promote developers to integrate security consideration and mitigation throughout the entire development process, rather than strictly at the end of development. With so many considerations to attend to, addressing security at the end of development can be overwhelming and exceedingly challenging to deliver a robust and comprehensive information security

system. Thus, it is critical that security be considered, handled, and adapted according to project needs throughout development. Only then can we ensure to provide the most layered and relevant defense possible, adhering to defense in depth. This iterative approach to security takes advantage of the benefits inherent to DevOps over traditional methods, and applies them to security, with the primary downside being integration (Veritis Group, 2022). This will result in the most efficient development and comprehensive defense possible. By adopting and implementing the policies and standards present, developers will be well on their way to providing a comprehensive, layered, and relevant defense in depth system.



[Threat Matrix:]

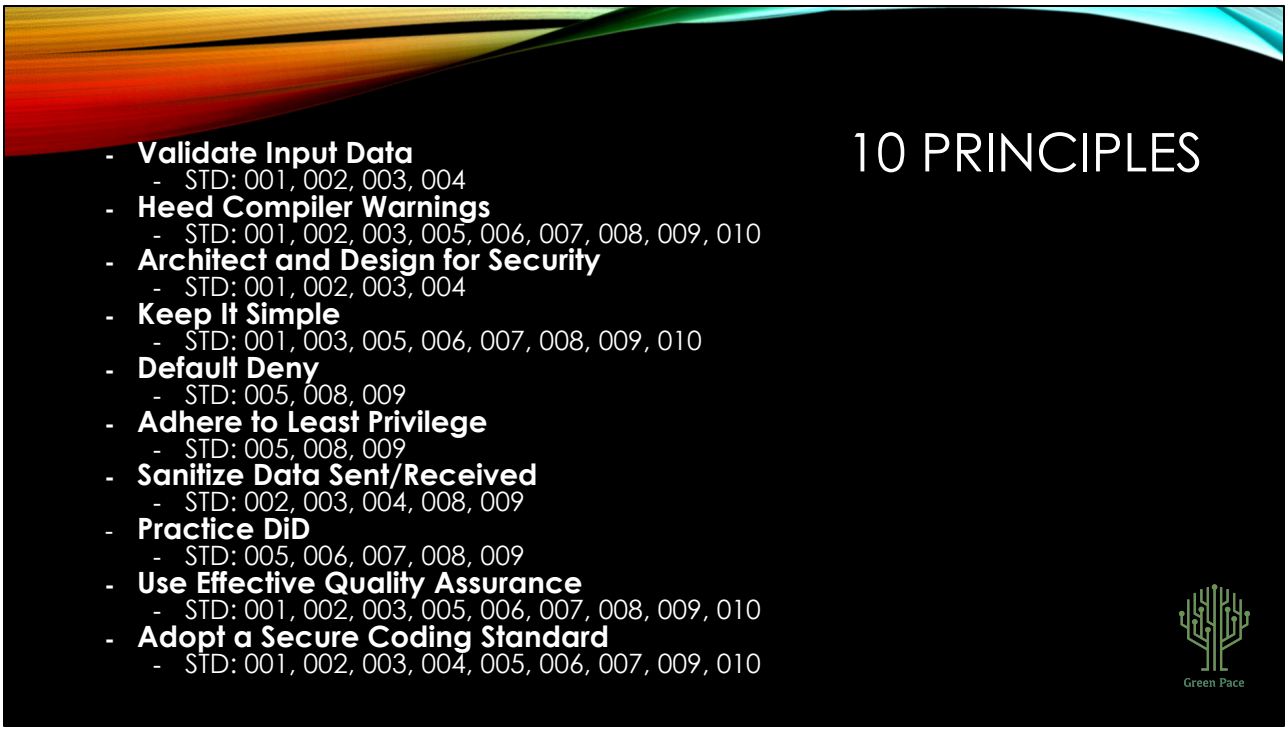
Starting from the lower left, that is to say the green square, denoting the least prioritized and least likely to occur vulnerabilities. We can see that two standards fall within this category for the following reasons;

STD-006-CPP: This coding standard aims to establish consistent and secure practices for handling assertions. Assertions act as concise documentation, explicitly defining the expected behavior at points in the program, and should be utilized for debugging purposes. As assertions are typically removed from the release build, it is important developers utilize assertions based on the standard. As assertions utilized in accordance with the standard and policies do not generally cause vulnerability or error, it is unlikely developers will need to devote much, if any time to troubleshooting assertions, hence the respective placement. As highlighted in the Security Policy document, developers are encouraged to utilize CppCheck for automated static code analysis throughout development to ensure adherence to the standard(Cppcheck, 2023).

STD-010-CPP: This coding standard aims to establish consistent and secure practices for documentation, commenting, and naming convention. For the sake of efficiency, readability, and to be in compliance with best practices, developers are required to provide documentation within their work. As properly implemented comments have no effect on the program's function, it is unlikely developers will need to troubleshoot comments for anything besides alignment with the standard, thus the low priority and severity rating. Although not directly related to commenting, CPPCheck static analysis tools can alert the developer when comments are incorrect to the program's function, or take on a different style than the majority (Cppcheck, 2023). Developers should configure analysis tools to include this functionality.

The yellow squares within the matrix represent medium severity vulnerabilities in relation to the standards, with the top left square being likely to happen, and the bottom right square being unlikely. Regarding the top left square, we have the following standards present:

STD-007-CPP: This coding standard aims to establish consistent and secure practices for handling exceptions. With exceptions there are a few key concepts to keep in mind, Developers should carefully review and understand the standard. As exceptions are an important part to error handling, and improper usage can result in unexpected behavior and inefficient resource usage, they have been given a medium severity (Grimm, 2023). Developers are required to employ static analysis tools like CPPCheck, MSVS Code Analysis, and Clang-Tidy. These tools can alert developers to improper implementation, uncleaned resources, irrelevant exceptions, and more.



[Principles:] The Security Policy Document highlights ten core principles that employees should familiarize themselves with. As we explore the document we will come across these principles, and their relevance to the standards outlined in the document. The ten core security principles are as follows (Seacord, 2011):

1. Validate Input Data: In the realm of secure coding standards, input validation remains paramount. This imperative underscores the necessity that any data received and utilized by a program across a trust boundary requires validation (Seacord, 2011). This security principle is relevant to the following standards found in the Security Policy Document;

STD-001-CPP: Data Type

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-004-CPP: SQL Injection

2. Heed Compiler Warnings: In the pursuit of secure coding practices an active

acknowledgement of compiler warnings assumes a pivotal role. To best fortify code against vulnerability, it is important to compile code using the highest warning level available, and to eliminate warnings via code refactoring (Seacord, 2011). This security principle is relevant to the following standards found in the Security Policy Document;

STD-001-CPP: Data Type

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-005-CPP: Memory Protection

STD-006-CPP: Assertions

STD-007-CPP: Exceptions

STD-008-CPP: Memory Management

STD-009-CPP: FIO Operations

STD-010-CPP: Naming, Documentation, Comment Standard

3. Architect and Design for Security Policies: For the sake of heightened security, software architecture and design should enforce security principles. This better ensures the adoption of Defense in Depth (DiD), and that security is not an afterthought or 'patchwork' solution. This security principle is relevant to the following standards found in the Security Policy Document;

STD-001-CPP: Data Type

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-004-CPP: SQL Injection

4. Keep It Simple: For the sake of security and efficiency, keeping a design as simple and small as possible remains an important tenet in the principles of security. Complex designs and implementations increase the likelihood that errors and vulnerabilities will

be introduced into the program(Seacord, 2011). This security principle is relevant to the following standards found in the Security Policy Document;
STD-001-CPP: Data Type
STD-003-CPP: String Correctness
STD-005-CPP: Memory Protection
STD-006-CPP: Assertions
STD-007-CPP: Exceptions
STD-008-CPP: Memory Management
STD-009-CPP: FIO Operations
STD-010-CPP: Naming, Documentation, Comment Standard

5. Default Deny: In the realm of security, the principle of default denial stands as a fundamental strategy. It is imperative to base access decisions on permission rather than exclusion. By default, access is denied until such conditions are met in which access is permitted. This security principle is relevant to the following standards found in the Security Policy Document;
STD-005-CPP: Memory Protection
STD-008-CPP: Memory Management
STD-009-CPP: FIO Operations

6. Adhere to the Principle of Least Privilege: Echoing the philosophy behind the principle of default deny, the principle of least privilege asserts that processes should execute with the least level of privileges necessary for the task. Any elevated permissions should be held for only the required duration. This security principle is relevant to the following standards found in the Security Policy Document;
STD-005-CPP: Memory Protection
STD-008-CPP: Memory Management
STD-009-CPP: FIO Operations

7. Sanitize Data Sent to and Received by Other Systems: In order to adhere to secure coding practices, it is imperative to sanitize data before transmitting to or receiving from external systems. By ensuring the information shared with or received from other systems is sanitized, developers help to mitigate the risk of injection attack and unauthorized manipulations (Seacord, 2011). This security principle is relevant to the following standards found in the Security Policy Document;

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-004-CPP: SQL Injection

STD-008-CPP: Memory Management

STD-009-CPP: FIO Operations

8. Practice Defense in Depth: Managing risk with multiple defensive strategies is essential to providing a robust and effective security atmosphere. DiD ensures that if one layer of security fails, there exists one or more subsequent layers of security to mitigate or prevent the attack or vulnerability. This security principle is relevant to the following standards found in the Security Policy Document;

STD-005-CPP: Memory Protection

STD-006-CPP: Assertions

STD-007-CPP: Exceptions

STD-008-CPP: Memory Management

STD-009-CPP: FIO Operations

9. Use Effective Quality Assurance Techniques: In the pursuit of robust security, effective quality assurance techniques are an indispensable means of verifying the absence of vulnerabilities beyond the development phase. A comprehensive and well-thought-out quality assurance strategy ensures the identification and mitigation of

vulnerabilities, contributing to system resilience (Seacord, 2011). This security principle is relevant to the following standards found in the Security Policy Document;

STD-001-CPP: Data Type

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-005-CPP: Memory Protection

STD-006-CPP: Assertions

STD-007-CPP: Exceptions

STD-008-CPP: Memory Management

STD-009-CPP: FIO Operations

STD-010-CPP: Naming, Documentation, Comment Standard

10. Adopt a Secure Coding Standard: The development or adoption of a secure coding standard is a fundamental practice in shielding software from potential threats. This practice both contributes to the security and resilience of an application, and remains fundamental in secure coding. This security principle is relevant to the following standards found in the Security Policy Document;

STD-001-CPP: Data Type

STD-002-CPP: Data Value

STD-003-CPP: String Correctness

STD-004-CPP: SQL Injection

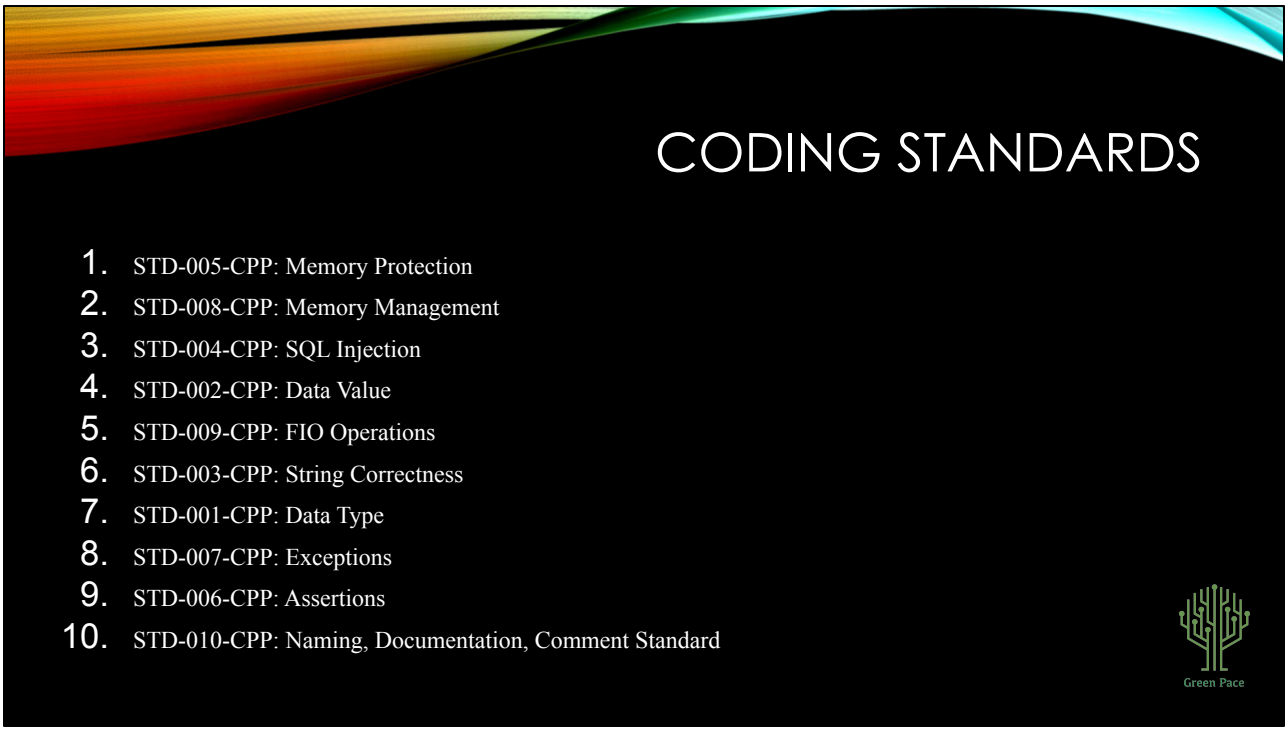
STD-005-CPP: Memory Protection

STD-006-CPP: Assertions

STD-007-CPP: Exceptions

STD-009-CPP: FIO Operations

STD-010-CPP: Naming, Documentation, Comment Standard



[Coding Standards:] The following coding standards aim to establish consistent and secure practices for development. Adhering to this standard will reduce the risk of vulnerabilities and exploits as well as enhance code quality. The standards are listed by severity and are as follows;

STD-005-CPP: Memory Protection

STD-008-CPP: Memory Management

STD-004-CPP: SQL Injection

STD-002-CPP: Data Value

STD-009-CPP: FIO Operations

STD-003-CPP: String Correctness

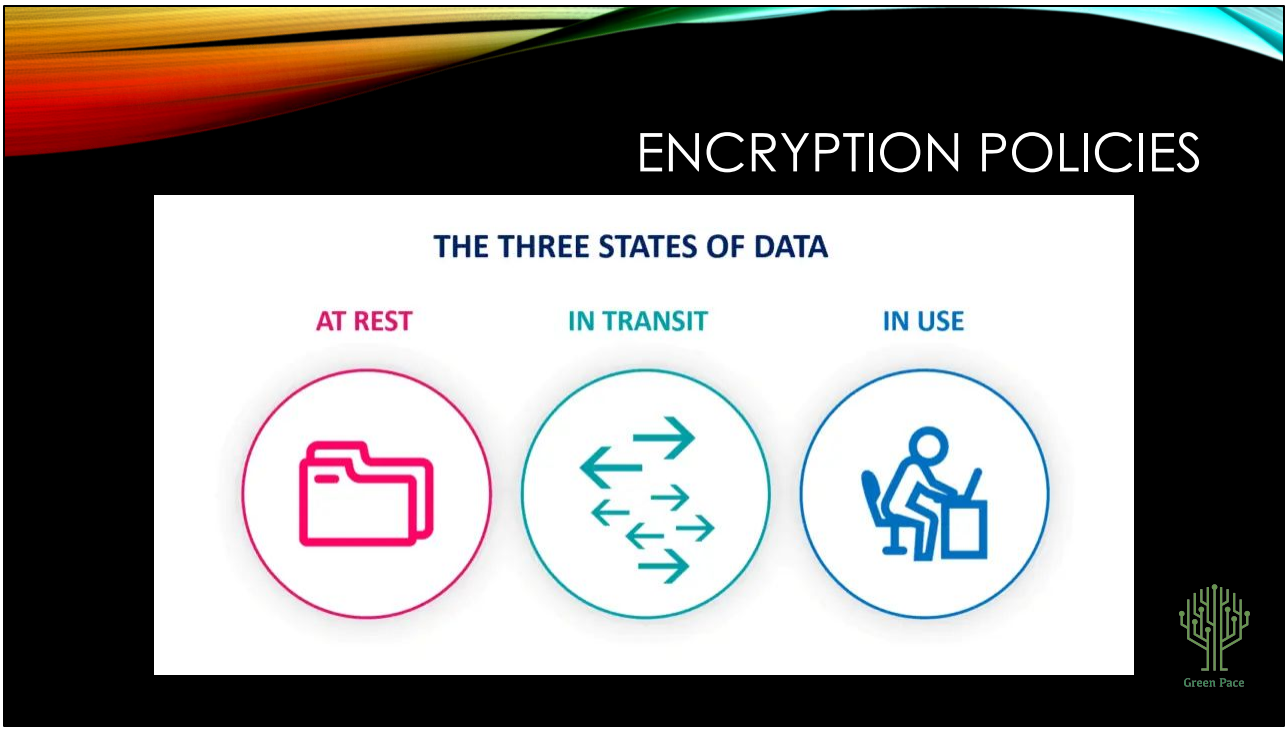
STD-001-CPP: Data Type

STD-007-CPP: Exceptions

STD-006-CPP: Assertions

STD-010-CPP: Naming, Documentation, Comment Standard

The severity of vulnerabilities range includes low, medium, high, and critical, with critical severity being reserved for what was identified as the most important standard, Memory Protection. Remediation cost ranges from low - high, with some standards sitting somewhere in between two ranges, due to the varied nature of potential vulnerabilities. The priority ranking ranges from low - high, with maximum priority being assigned to memory protection. Finally, the threat level for each standard represents a compilation of the previous scales, with 1 being low threat, 2 representing a medium threat that must be watched, and 3 representing a critical threat level that must be addressed throughout development.



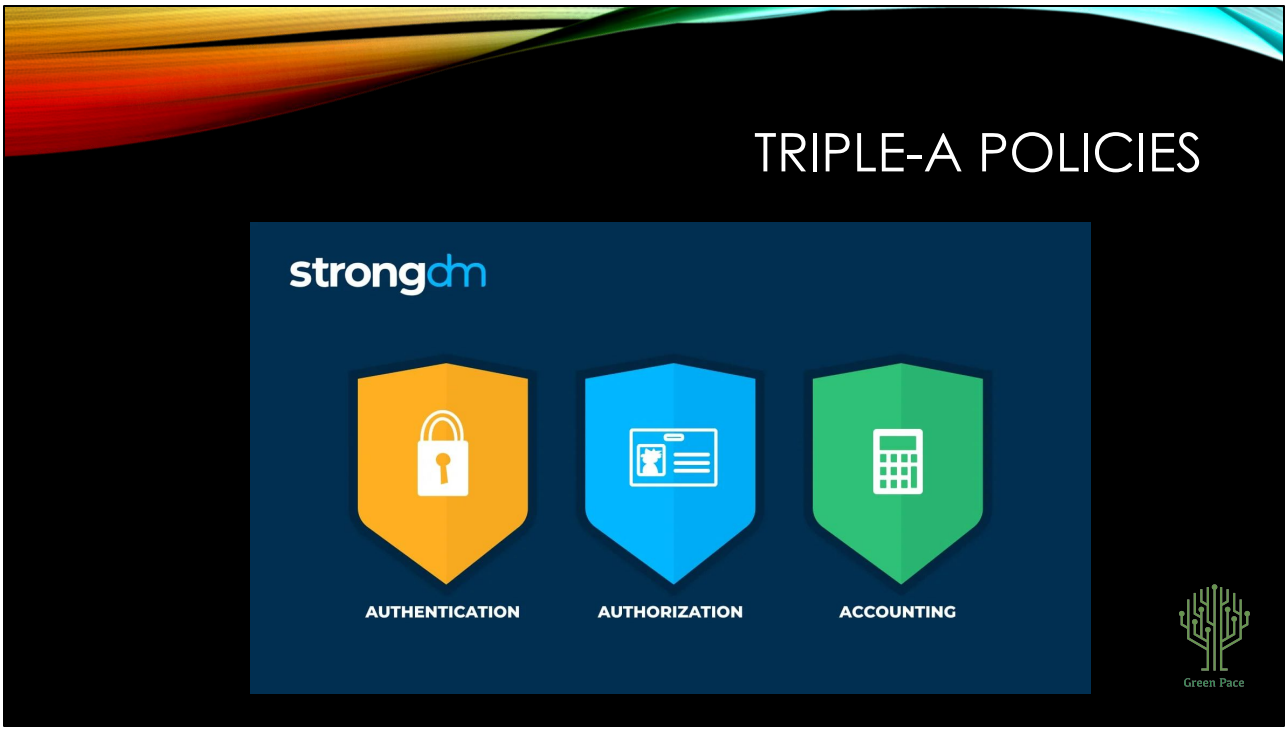
[Encryption Policies:] The following encryption policies have been drafted to establish consistent and secure practices for encryption. Adhering to these standards will help the developer adequately apply encryption and reduce the risk of vulnerability and information breach.

At Rest: Encryption at rest is a critical part of application security and a robust DiD system. Encryption at rest involves securing data when it is stored in non-volatile storage, such as a DB, file, or backup harddrive. All sensitive and confidential data must be stored in an encrypted format, and **NEVER** in plaintext. Developers are required to employ industry standard encryption algorithms, such as AES (IBM, 2023). Encryption at rest is essential because it mitigates the risk of data breach should a malicious agent gain access to the data. It provides a final layer of security within a DiD system.

At Flight or In Transit: Encryption in flight is similarly important to that at rest.

Encryption in flight refers to protecting data as it travels over a network, preventing unauthorized interception or eavesdropping. Communication over networks, both internal and external, must be encrypted using industry standard secure transport layer protocols (TLS/SSL) (Amazon, 2023b). Developers must implement end-to-end encryption for data transmission, utilize strong ciphers, and ensure protocols are regularly updated to address vulnerabilities. Encryption at flight safeguards data as it travels. Were this step not taken, any attacker intercepting the information would have access to it, similar to the rationale for encrypting data at rest, it provides a last line of defense should the DiD layers be breached.

In Use: Encryption in use or confidential computing focuses on protecting data while it is being actively processed or utilized by an application. On both the server and client sides, all data being processed in memory or during computational tasks must be encrypted. Developers should employ encryption libraries and technologies that allow for the processing of sensitive data in memory, such as Crypto++ (Crypto++, 2023). Encryption in use ensures that sensitive data remains protected during active processing, further reducing the risk of vulnerability and attack.



[Triple-A Framework:] The role of triple A in cybersecurity is paramount, Authentication ensures the verification of entities, Authorization dictates access permissions, and Accounting provides essential tracking for auditing and analysis. The following policies have been drafted to provide guidelines to developers on Triple-A best practices and can be found within the Security Policy Document's section on Policies for Encryption and Triple-A:

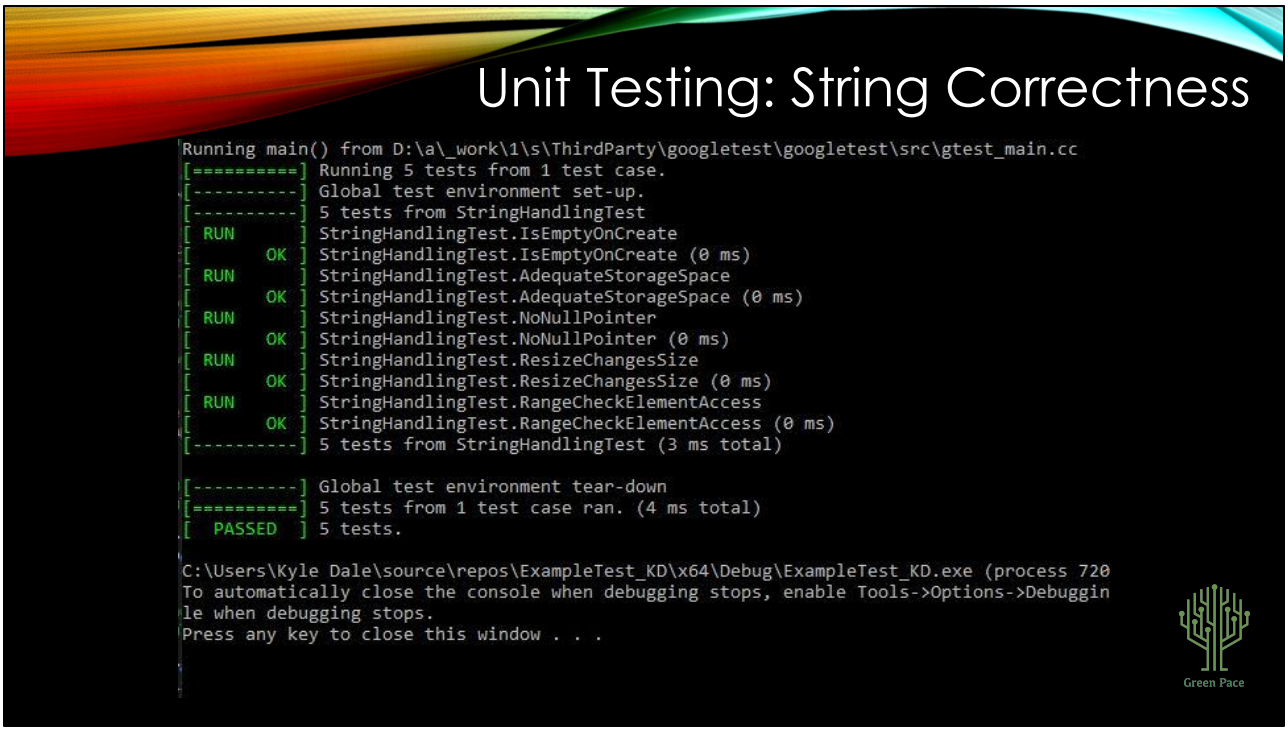
Authentication: Authentication is an important factor in computer security, and a key portion of AAA. In-line with Default Deny and Least Privilege, all users, systems, and entities must undergo authentication before gaining access to network information, resources, or applications (Gillis, 2023). Authentication helps to ensure that only authorized individuals gain access to the network or system. This will help minimize the risk of malicious agents taking advantage of the system or its resources.

Authorization: Authorization is an important factor in computer security, and a key

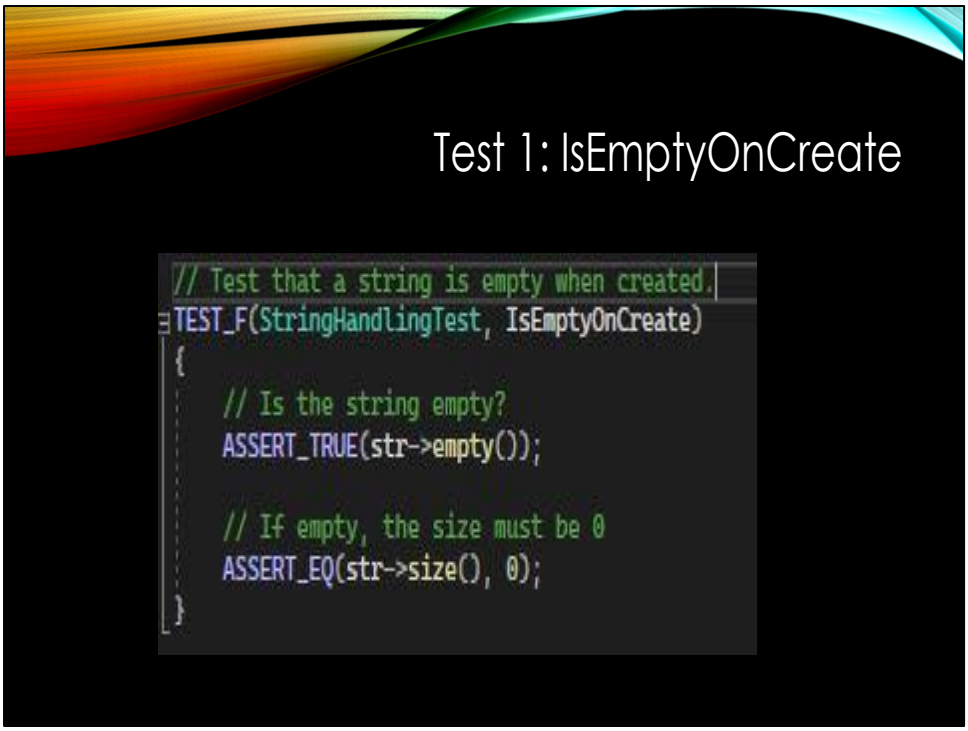
portion of Triple-A. Authorization is the process of determining the level of access or permissions that a viable user has access to within the network. Aligned with Default Deny and Least Privilege, all users, systems, and entities are assigned a level of access based on role and responsibility, only being allowed to access what is necessary. It is imperative to define and implement role-based access control (RBAC) and attribute-based access control (ABAC) mechanisms within the system (Gillis, 2023). Access should be regularly audited to update permissions as is necessary.

Accounting: Accounting is an important factor in computer security, and a key portion of Triple-A. Accounting refers to the process of tracking and logging activities related to the other A's, providing an auditable trail for accountability and security. It is important to record and monitor all authentication and authorization events, successful or failed. This will allow maintenance to generate and review logs regularly for suspicious activities or security breach incidents. Developers should implement centralized logging mechanisms and security information and event management systems (SIEM) (Gillis, 2023). Developers must ensure to configure the systems properly so that it logs relevant information. Accounting serves a crucial role in Triple-A, providing necessary information to audit, analyze, and monitor the system.

By adhering to the principles laid out for Triple-A and Encryption, developers can inherently ensure they are abiding by best practices and providing layers within their DiD system. Utilizing these practices in conjunction with the standards and principles laid out in this document will ensure we maintain robust security in every layer of development and throughout the company.

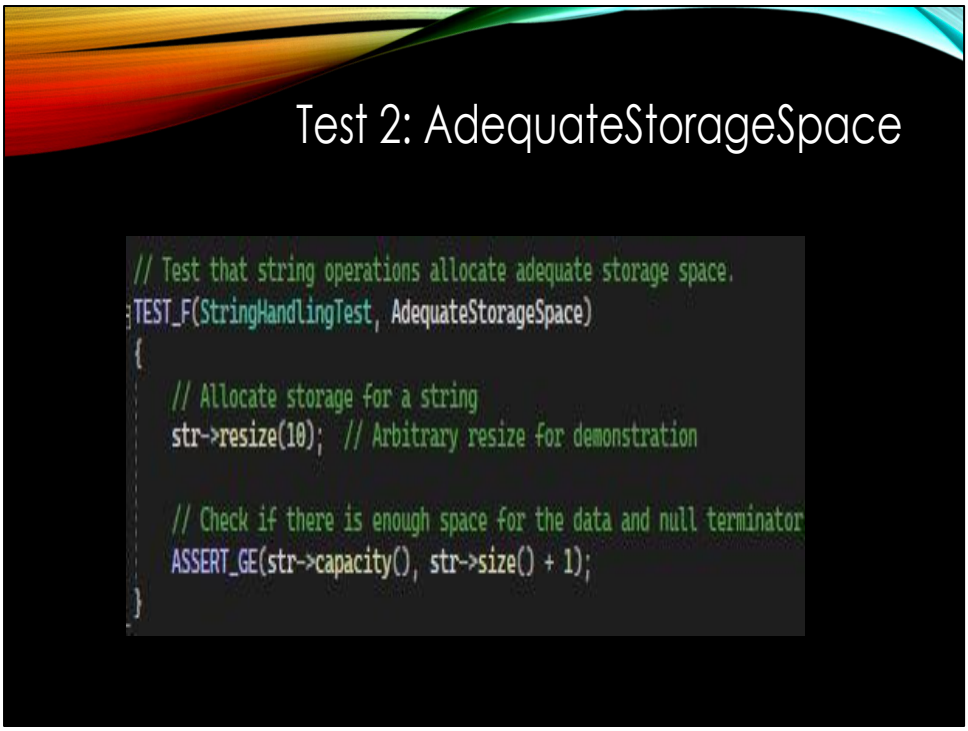


[Unit Testing Main:] Unit Testing is an important part of debugging code and catching potential errors. In the following few slides, I will be demonstrating some potential Unit Test implementations to verify String Correctness in accordance with STD-003-CPP. As you can see, I have created 5 unit tests, to address various aspects of the standard.



[Unit Testing 1:] IsEmptyOnCreate

The first Unit Test, IsEmptyOnCreate, verifies that the string object is empty when it is first created. This is done by asserting that the string is empty, as well as asserting that the size is equal to 0. Both of these assertions guarantee that string objects are being created properly empty.

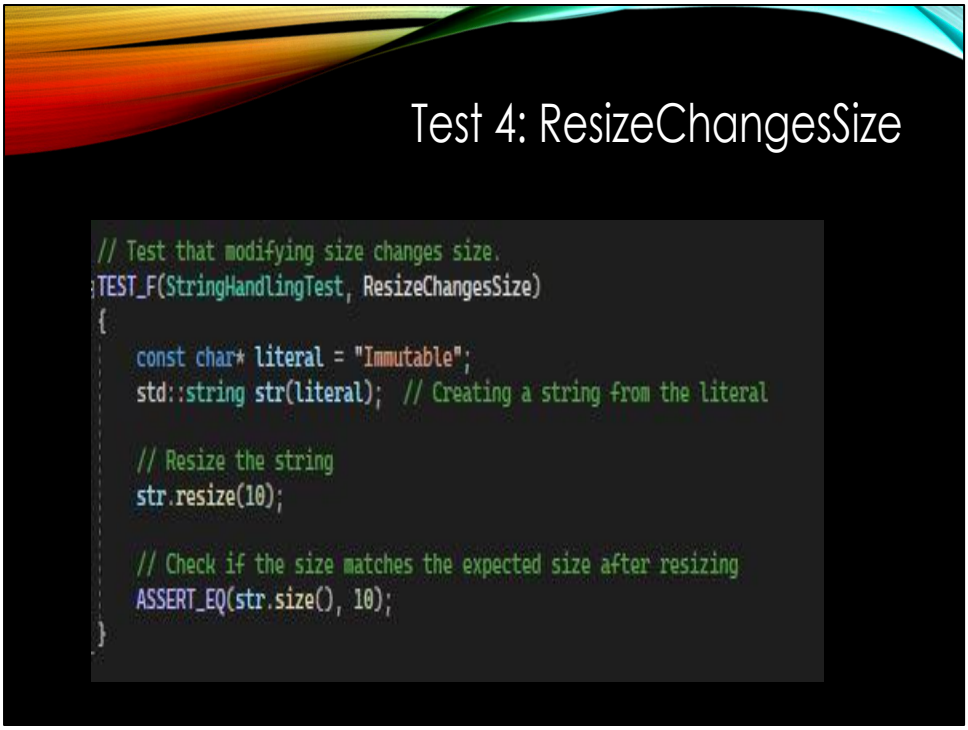


[Unit Testing 2:] AdequateStorageSpace

The second Unit Test, AdequateStorageSpace, verifies that string operations allocate adequate storage space for both the data and null terminator. This test helps guarantee proper string functionality and mitigates loss of data or out-of-bounds errors.

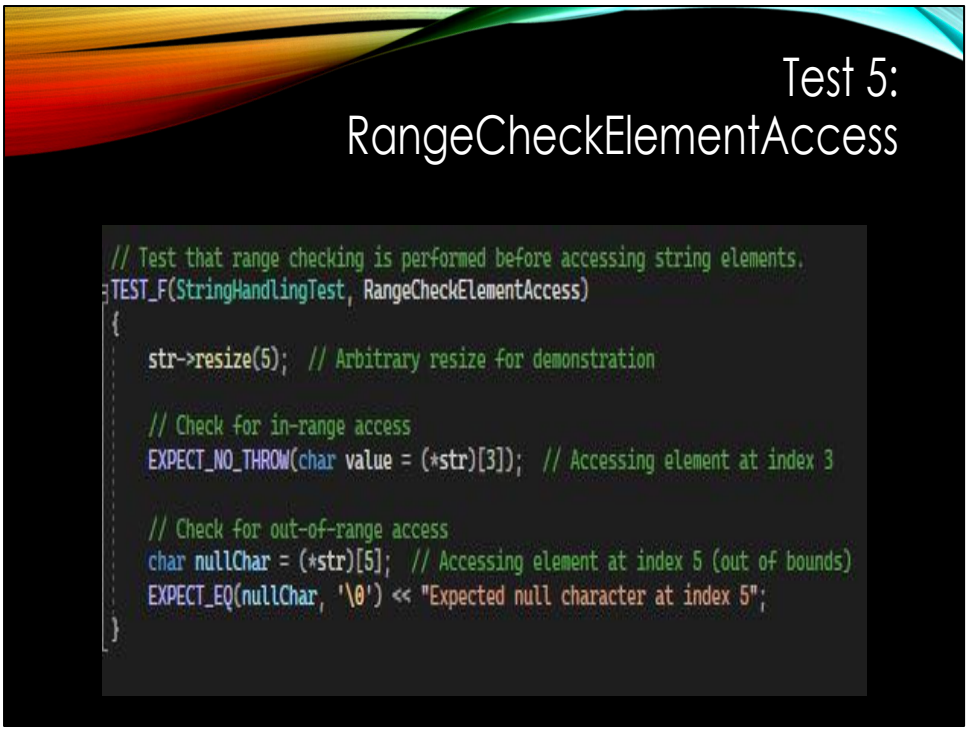
[Unit Testing 3:] NoNullPointer

The third test, NoNullPointer, verifies that strings created from null pointers throw an exception. The test verifies the pointer nullPointer is indeed nullptr, if it is, that implies creating a string from the null pointer has failed, as was expected (a negative outcome). Therefore this is also a negative test.



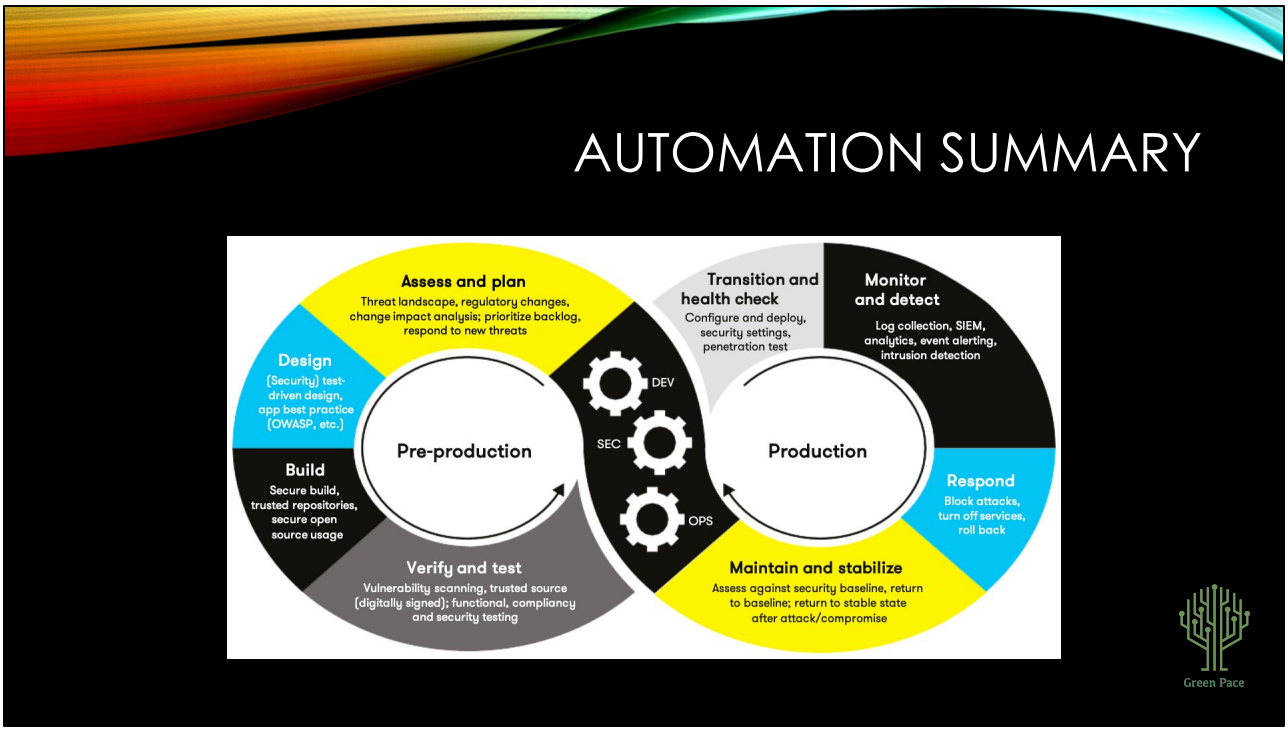
[Unit Testing 4:] ResizeChangesSize

The fourth test, ResizeChangesSize, verifies that modifying the size of a string correctly changes the size. This is done by resizing a string to 5 elements (0-4), then expecting that element 3 is accessible while also expecting element 5 will equal a null character, as it is out of range.



[Unit Testing 5:] RangeCheckElementAccess

The fifth and final test, RangeCheckElementAccess, checks whether range checking is performed before accessing string elements. This test verifies both in-range and out-of-range access by attempting to access element 3 and 5 in a string of length 5 (so it has elements 0-4).



[Automation Summary:] Considering security from the beginning of development provides many benefits to an application's health and robustness, including the ability to plan implementation for both software and hardware security features ahead of time. DevSecOps as a development style features automation in many of the processes involved in secure development. The following descriptions describe the stages involved in DevSecOps, as well as how automation may be incorporated at each stage:

Assess and Plan: Planning and Assessment involves defining requirements, setting goals, establishing a roadmap, and accessing necessary environments and resources for development. Although much of this process is creatively motivated and not automated, it can benefit from programs like project management tools, backlog generation, and virtual tracking systems (Tarimela, 2022). Additionally, IaC tools can be utilized to aid in the configuration process for project environments.

Design: The design phase of DevSecOps can be heavily automated to detect and

mitigate vulnerabilities and deviations from the standard. Although the actual code will need to be implemented largely by-hand on the part of a developer, there are many static analysis tools, such as the ones mentioned previously, that can help a developer quickly identify and solve vulnerabilities and errors. These tools should be heavily utilized throughout design to ensure that defects do not get baked further into development.

Build: Similar to the design phase, the build phase of DevSecOps can be heavily automated. Utilizing continuous integration tools, code can be automatically built and compiled when changes are committed. Concurrently, dynamic analysis tools can be integrated to perform analysis during the build process, working in conjunction with static tools (SAST) to identify vulnerabilities (Tarimela, 2022).

Verify and Test: Automated testing is a key aspect of the verification stage in DevSecOps. This can include, but is not limited to unit testing, integration testing, and security testing. Test automation tools (DAST) can aid developers by running test suites quickly and consistently, ensuring changes do not introduce vulnerability or error into the code base (Tarimela, 2022).

Transition and Health Check: The transition phase of DevSecOps involves deploying the application into production. Automation can feature in this stage in the form of continuous deployment tools. These tools aid developers by automatically validating if changes to a codebase are correct, stable, and ready for immediate autonomous deployment to a production environment (Pittet, 2023).

Monitor and Detect: Automation plays a pivotal role in monitoring and detecting defects within a deployed application. Automated monitoring tools can continuously track the performance and health of a deployed application (Tarimela, 2022). These

tools can alert developers to any noteworthy changes in the environment, enabling development teams to quickly identify and subsequently respond to issues or threats.

Respond: Although automation is not directly involved in the response portion of DevSecOps, as this will require human intervention, there are roles that automation can play in this phase. Automated systems have the capability to detect and isolate compromised systems, as well as rollback to previous deployments in the result of catastrophic failure. Human intervention is required for this phase, but automation can still play a part in assisting developers to work efficiently.

Maintain and Stabilize: Many routine maintenance processes can be automated, such as updating and patch management. Human intervention will similarly always be required for this phase of DevSecOps, but automation also has a part to play.

[Tools:] On screen are a list of various tools and programs that developers must utilize to aid them in the development process. Many of these tools have already been mentioned previously, and are as follows;

CPPCheck:
Version: 2.12
Checker: Comprehensive Static Analysis
Description: CPPCheck can perform a comprehensive static analysis when all checks are turned on. This will check for a variety of violations and style errors in C++ code. This tool can be incorporated during the design and build phases of development. Additionally, they can be utilized during the verify and test phase as well, but should take a secondary role to DAST for this phase. [https://cppcheck.sourceforge.io/]

MSVS Code Analysis:
Version: VS 17.8.2
Checker: Static Analysis
Description: Microsoft Visual Studios comes with a built-in static analysis tool that should be enabled in the project properties for further coverage. This tool will primarily be relied on during the design and build phases of development.
https://learn.microsoft.com/en-us/cpp/build/reference/analyze-code-analysis?view=msvc-170

Clang-Tidy:
Version: 18.0.0

Checker: Static Analysis
Description: Clang-Tidy is a static analysis tool that can check for a variety of errors. For implementation in C++ see the Security Policy Document. This tool will primarily be relied on during the design and build phases of development.
https://chat.openai.com/c/26243feb-b3a1-418f-9ef2-de1c3d2c60f5

PVS-Studio
Version: 7.22
Checker: Static Analysis
Description: PVS-Studio is a static analyzer that aids in code quality, security (SAST) and safety. More information on specific versions and languages, including C++, can be found in the Security Policy Document. This tool will primarily be utilized during the design, build, and testing phases of development.
https://pvs-studio.com/en/

Checkmarx
Version: 3.0
Checker: Checkmarx One
https://checkmarx.com/product/application-security-platform/.
Description: Checkmarx One is a cloud-native platform that provides a full suite of application security testing (AST). This tool will primarily be utilized during the verify and test phase of development.

AddressSanitizer
Version: Clang 18.0.0
Checker: Memory Sanitizer
https://clang.llvm.org/docs/MemorySanitizer.html
Description: AddressSanitizer with Memory Sanitizer capabilities can detect various memory-related issues enhancing protections. This tool will primarily be utilized during the design, build, and testing phases of development.

Valgrind
Version: 3.22.0
Checker: Memcheck
Description: Valgrind with Memcheck can aid in detecting memory-related errors and leaks. This tool will primarily be utilized during the design, build, and testing phases of development.

CircleCI
Version: 3.x
Checker: Continuous Integration
Description: CircleCI is a continuous integration and continuous delivery tool that will primarily be utilized during the Transition and Monitor phases of development.
https://circleci.com/

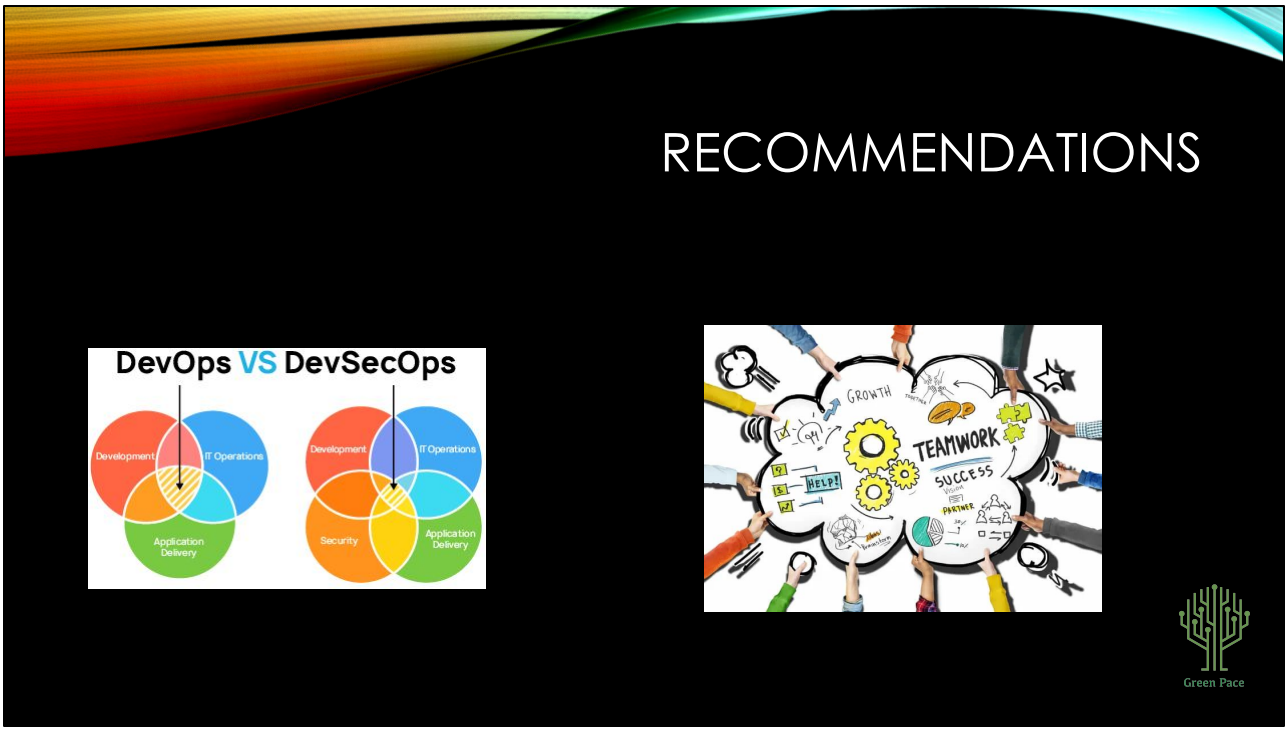
[Risks and Benefits:]

There are many risks associated with adopting an entirely new system of development. However, it is a necessary step to propel Green Pace firmly into the future. Additionally, the upcoming security audit demands we increase our emphasis on security at multiple levels throughout the company. Once integration of DevSecOps has occurred, it will not harm the efficiency seen in our current DevOps method of development. Rather, by incorporating security throughout the process, development and security teams can work in conjunction to implement best practices and ensure a solid foundation of security, while maintaining efficiency. As security is generally handled at the end of the development lifecycle, this often leads to a less robust and capable system that may not qualify as healthy DiD. The additional resources spent on maintenance and mitigation after deployment more than accounts for the small time investment during the development cycle seen in DevSecOps.

Although DevSecOps is a powerful and sensible method of development, there will be

difficulties in implementation. The most prominent difficulty associated with transitioning has shown to be developers being unwilling or slow to adapt (Patni, 2023). Adopting DevSecOps and the integration of tools associated with it can be a daunting task for those unfamiliar with the development style. In order to ensure the success of integration adequate time and resources must be spent toward easing the transition. Another difficulty with DevSecOps, specific to Green Pace, is that our prior method of development, DevOps, focuses on speed and time to market. Although DevSecOps is efficient, it will not be as fast as DevOps initially (Patni, 2023). Developers and security teams alike will need to practice patience, as they will be working in tandem with DevSecOps strategies. Teams will need to carefully balance speed and security, and as such communication will be vital to the success of DevSecOps. Teams should spend time exploring the ‘Automation’ section of the Security Policy Document to further explore plans for integration and a more detailed analysis of the benefits of DevSecOps.

The Security Policy document successfully outlines the policy changes that will take place in implementing DevSecOps, but does not thoroughly address the higher level changes and risks associated with the adoption. It can, and has proven to be a challenging transition for other organizations. All of us here at Green Pace will need to proactively ensure that the avenues of communication between teams and individuals remain strong. As the greatest downside to DevSecOps is the adoption and transition, by proactively supporting and engrossing ourselves in this transition, we can ensure our continued success, growth, and team focus that make this company what it is.

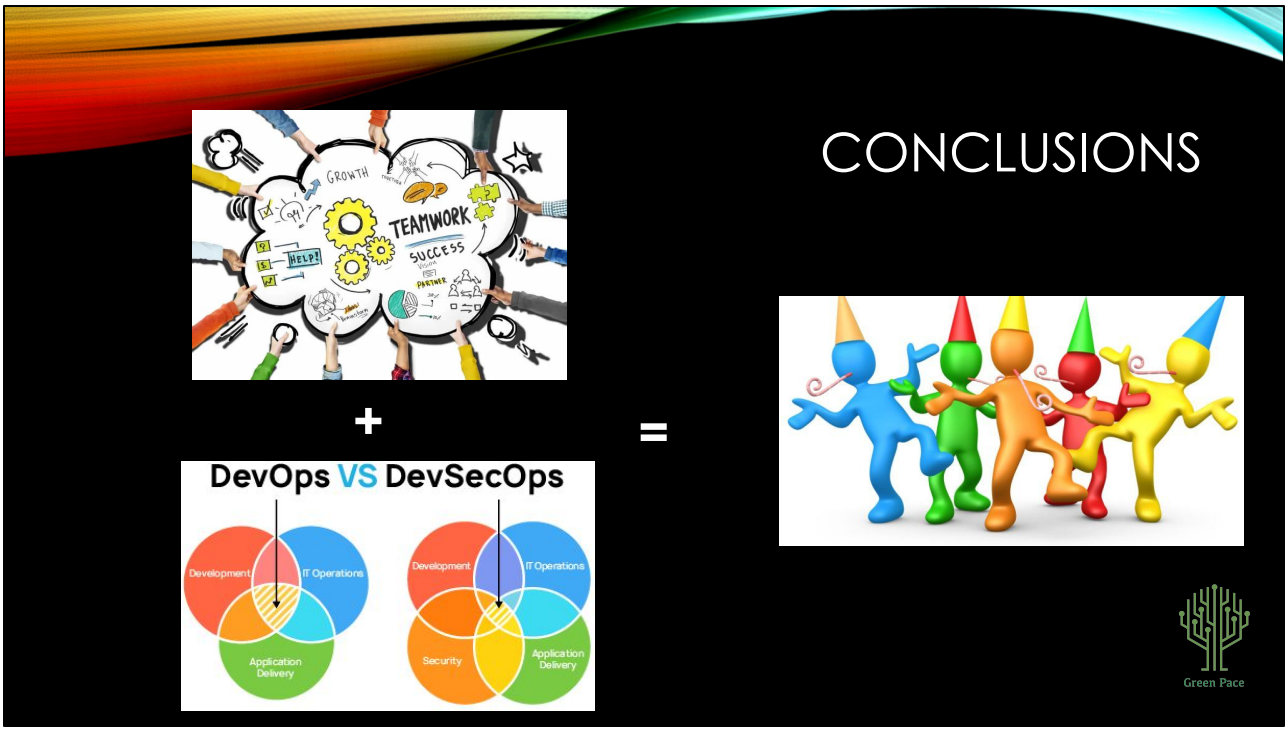


[Recommendations:] As touched upon previously in this presentation, although the DevOps pipeline is efficient in terms of time-to-market, it often results in a less than efficient and robust implementation of security. DevSecOps seeks to address this shortcoming by applying the powerful advantages inherent in DevOps to security. Considering security throughout the process of development is much easier said than done however.

The term “security” is rather broad, and encompasses a wide variety of necessary actions and steps needed to provide a comprehensive DiD strategy. As seen in the transition to DevSecOps by Allianz, the 34th largest company in the world, code policies were not the only thing needed (Foxen, 2017). Similar to us at Green Pace, Allianz needed to change how it delivered software. The company was struggling to meet deadlines while also providing robust and secure applications. Subsequently they were losing market share to newer, innovative start-ups practicing agile methods of development. After a successful limited trial run and subsequent integration to

DevSecOps, Allianz has recognized building team trust, proactive security education for all employees, and promoting a sense of unity throughout the transition, as the most important keys to a successful implementation (Foxen, 2017). Ultimately, after allowing room for the transition period, Allianz has reported a greater level of employee engagement (due to the team processes and individual involvement), a massive decrease in investment toward fixing vulnerabilities and maintenance post launch, and a boost of overall sales due to the speed at which the product was delivered, and subsequently how relevant it was (Foxen, 2017).

As reiterated through this document, Green Pace will need to carefully adopt the new security principles as well as foster teamwork and unity throughout the company. Only then can we safely assure the transition to DevSecOps will be successful and we are prepared to tackle shifting security threats efficiently.



[Conclusion:] In essence, the transition to DevSecOps marks a crucial milestone in Green Pace's journey, albeit a challenging one. This transformative shift requires the collective efforts of every employee, ranging from acquiring foundational security knowledge to developers seamlessly integrating innovative tools and methodologies into their workflow.

Drawing inspiration from the success story of Allianz, our success hinges on our ability to implement best practices such as DiD and Triple-A, support and listen to one another, and foster a cohesive team spirit. DevSecOps emerges as the solution to our security concerns, acting as the key to passing audits, safeguarding our clients' information, and positioning us at the forefront of technology and security. To thrive in this evolving landscape, we must embody collaboration and unity as we navigate the dynamic realm of DevSecOps. This is not merely a transition; it's a strategic move that propels us into a future where security is not just a measure but an integral part of our identity.

REFERENCES

Amazon. (2023a). *What is DevSecOps?*. Amazon. https://aws.amazon.com/what-is/devsecops/.

Amazon. (2023b). *What is SSL/TSL Certificates*. Amazon. https://aws.amazon.com/what-is/ssl-certificate/.

Carnegie Mellon University. (2023). Sei cert C++ coding standard. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682.

Cppcheck. (2023). https://cppcheck.sourceforge.io/.

Crypto++. (2023). *Crypto++® library 8.9*. Crypto++. https://www.cryptopp.com/.

Foxen, B. (2017, November 29). *Real-world stories of DevSecOps*. Contino. https://www.contino.io/insights/real-world-stories-of-devsecops-in-the-enterprise.

Gillis, A. S. (2023, October 11). *What is authentication, Authorization and Accounting?: Definition from TechTarget*. Security. https://www.techtarget.com/searchsecurity/definition/authentication-authorization-and-accounting.

Grimm, R. (2023, June 26). *Home*. MC++ BLOG. https://www.modernescpp.com/index.php/c-core-guidelines-rules-to-exception-handling/.

IBM. (2023, June 6). *Content encryption: In flight and at rest*. IBM. https://www.ibm.com/docs/en/aspera-on-cloud?topic=encryption-content-in-flight-rest.

Microsoft . (2023, June 14). *Addresssanitizer*. Microsoft Learn. https://learn.microsoft.com/en-us/cpp/sanitizers/asan?view=msvc-170

OWASP. (2023a). *Explore the worldof cyber security*. OWASP. https://owasp.org/.

[References:] Now that we have reached the end of the presentation please take a minute to review the references. The full list can be found in the accompanying transcript. Thank you for your time, good luck to everyone with this transition, and please refer to the Security Policy Document for clarification on any questions. Take care.

REFERENCES CONTINUED

OWASP. (2023b). *SQL injection prevention cheat sheet¶*. SQL Injection Prevention - OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html#defense-option-2-stored-procedures.

Patni, A. (2023, January 30). *DEVSECOPS Challenges & Top 5 solutions for its implementation (2023)*. Practical DevSecOps. https://www.practical-devsecops.com/devsecops-challenges-and-top-5-solutions-for-its-implementation/#:~:text=Cannot%20fully%20automate-,Not%20willing%20to%20change%20and%20adapt,doing%20things%20will%20take%20time.

Pittet, S. (2023). *Continuous deployment*. Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/continuous-deployment#:~:text=Continuous%20deployment%20(CD)%20is%20a,deployment%20to%20a%20production%20environment.

Seacord, R. (2011, March 1). *Top 10 Secure Coding Practices*. CERT. https://web.archive.org/web/20160404152049/https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices.

Tarimela, A. (2022, October 12). *What are the phases of Devsecops* . Go to Veritis Group Inc. https://www.veritis.com/blog/what-are-the-phases-of-devsecops/.

Valgrind. (2023). *memcheck: A memory error detector*. Valgrind. https://valgrind.org/docs/manual/mc-manual.html.

Veritis Group Inc. (2022, July 1). *Waterfall vs agile vs DevOps methodologies comparison*. Go to Veritis Group Inc. https://www.veritis.com/blog/waterfall-vs-agile-vs-devops-which-production-method-should-you-take/.

IMAGE REFERENCES

Checkmarx. (2023, August 5). *Checkmarx one industry's most comprehensive AppSec platform*. Checkmarx.com. https://checkmarx.com/product/application-security-platform/.

Cppcheck. (2023). https://cppcheck.sourceforge.io/.

Dent, J. (2018, March 8). *Collaboration or teamwork – what's the difference?* LinkedIn. https://www.linkedin.com/pulse/collaboration-teamwork-whats-difference-dent-lssbb-dmx2.

IONOS. (2023). *It security - reporting security threats*. IONOS " Hosting Provider. https://www.ionos.com/it-security.

Magnusson, A. (2023, October 5). *What is AAA Security? authentication, authorization, and accounting*. StrongDM. https://www.strongdm.com/blog/aaa-security.

PVS. (2023). *We develop a static analyzer for C, C++, C#, and java code*. PVS. https://pvs-studio.com/en/.

Sagaran, S. (2016, October 27). *Defining celebration (part-1)*. Beyond Horizons. https://www.beyondhorizons.biz/defining-celebration-part-1/.

Sealpath. (2021, October 26). ➢*the three states of data guide - description and how to secure them*. Sealpath. https://www.sealpath.com/blog/protecting-the-three-states-of-data/.

Shoaib, A. (2020, May 5). *Teamwork and its importance*. Medium. https://medium.com/@hafizahmedshoaib/teamwork-and-its-importance-8153ff1d3c1e.

Udemy. (2023). *Audit procedures: 7 Ways to Streamline Operations - Udemy blog*. Udemy. https://blog.udemy.com/audit-procedures/.

Wikimedia Foundation. (2023c, December 10). *Circleci*. Wikipedia. https://en.wikipedia.org/wiki/CircleCI.

Wikimedia Foundation. (2023a, September 15). *Valgrind*. Wikipedia. https://en.wikipedia.org/wiki/Valgrind.

Wikimedia Foundation. (2023b, December 7). *Visual studio*. Wikipedia. https://en.wikipedia.org/wiki/Visual_Studio.