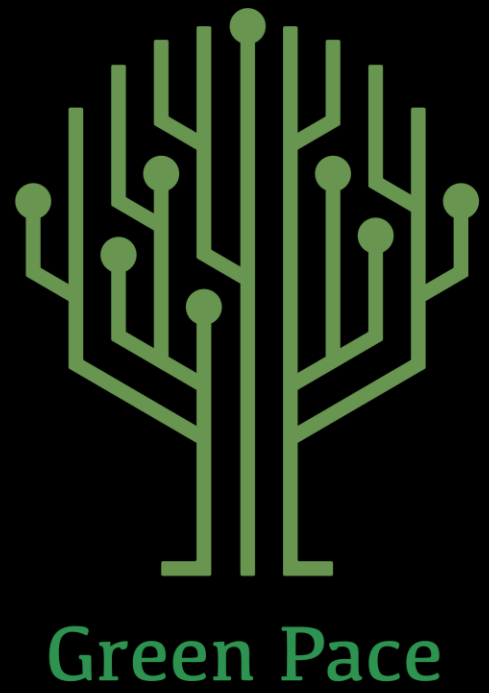
Green Pace

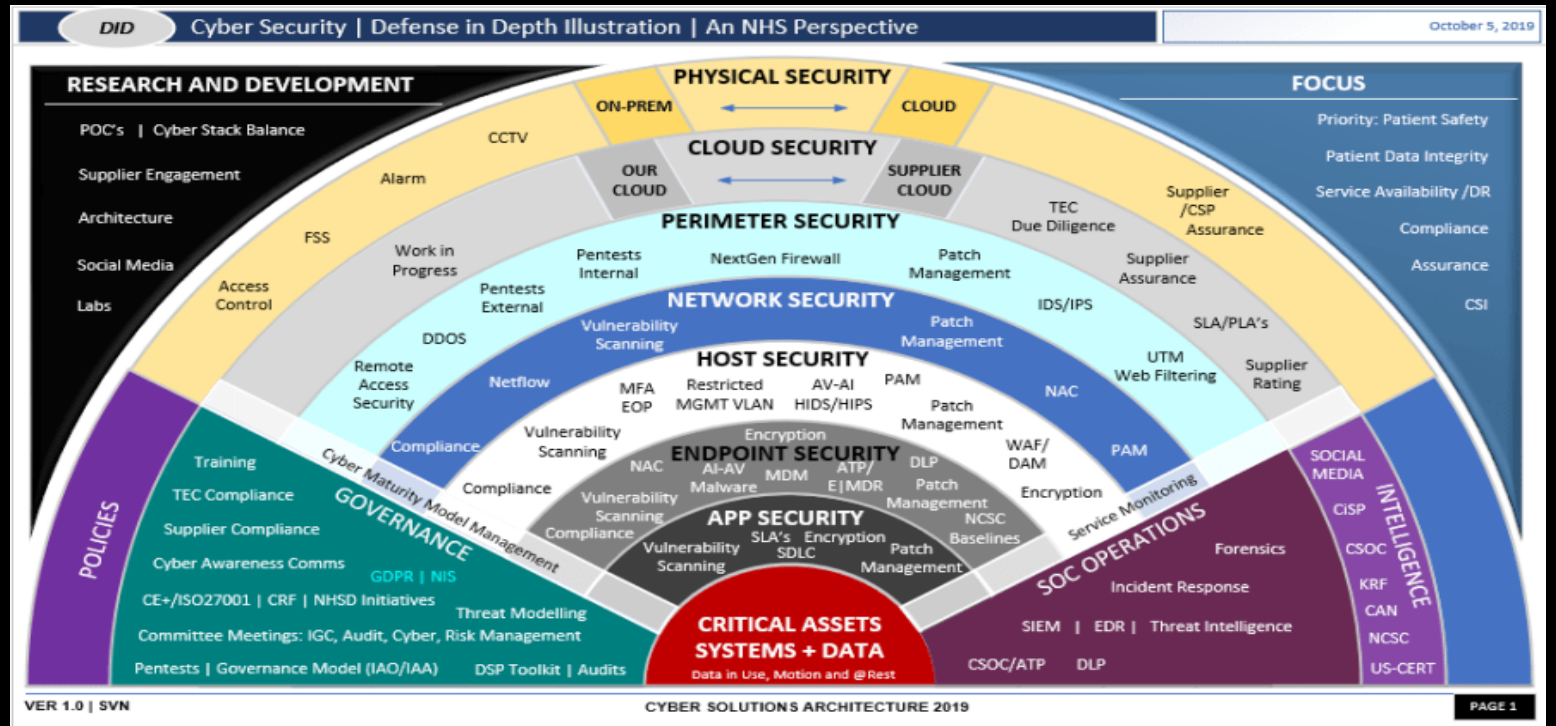
Security Policy Presentation

Developer: David Hughes



Overview: Defense in Depth

- The Green Pace Security Policy ensures consistent implementation of secure principles for all software development. This policy aligns with best practices for defense-in-depth, protecting against a wide range of potential vulnerabilities.



Green Pace

Threats Matrix

- Examples of threats: SQL Injection, XSS, Buffer Overflow, Hard-Coded Secrets. Risk levels and mitigations detailed in the matrix.

Threat Type	Risk Level	Likelihood	Remediation Cost	Priority
SQL Injection	High	High	Medium	1
Cross-Site Scripting (XSS)	High	Medium	Medium	2
Buffer Overflow	Critical	Medium	High	1
Hard-Coded Secrets	High	High	Low	1
Uninitialized Memory Access	High	Medium	Low	3

Green Pace

10 Principles

1. Validate Input Data
2. Heed Compiler Warnings
3. Architect and Design for Security Policies
4. Keep It Simple
5. Default Deny
6. Adhere to the Principle of Least Privilege
7. Sanitize Data Sent to Other Systems
8. Practice Defense in Depth
9. Use Effective Quality Assurance Techniques
10. Adopt a Secure Coding Standard

Coding Standards

- Prioritized list: Input Validation, SQL Injection Prevention, Secure Password Storage, Memory Management. Examples provided for compliant and non-compliant practices.

Green Pace

Encryption Policies

- Data Encryption:
  - At Rest: AES-256 for storage security
  - In Flight: TLS for data transmission
  - In Use: Enclaves for runtime protection.

Green Pace

Triple-A Framework

- Authentication: MFA and password policies
- Authorization: RBAC for resource access
- Accounting: Logging user activities with Splunk or ELK Stack.

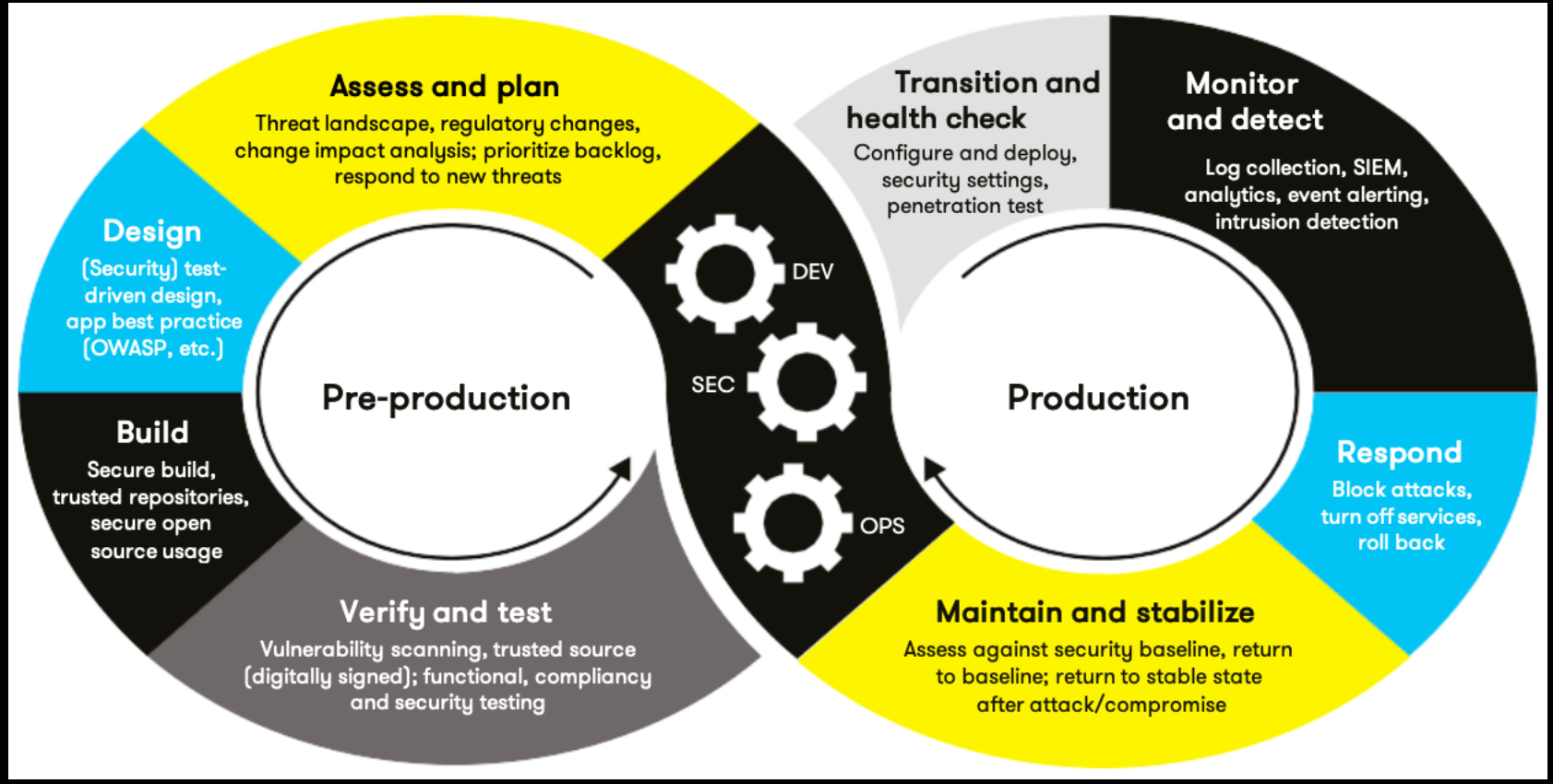
Green Pace

Unit Testing

- Unit tests for SQL injection, memory management, and input validation using tools like Coverity, SonarQube, and Valgrind.

Green Pace

Automation Summary



Green Pace

Risks and Benefits

- Current Gaps: Lack of automated testing for vulnerabilities
- Benefits: Improved compliance and reduced breach risks.

Green Pace

Recommendations

- Focus on automating validation and enhancing monitoring. Adopt secure libraries for cryptographic functions.

Green Pace

Conclusion

Policy Evolution Needs

- Enhanced Automation: Transition manual processes like vulnerability identification into automated workflows using tools like SonarQube and Coverity. Expand DevSecOps integration to ensure continuous scanning and monitoring.
- Regular Updates: Review security standards annually or when new threats arise to maintain effectiveness. Incorporate lessons learned from incidents into updated practices.
- Training and Awareness: Conduct workshops and training on secure coding practices and emerging threats such as AI-driven exploits and supply chain attacks.
- Runtime Protection: Focus on securing runtime environments through encryption in use, like Intel SGX, and robust memory management.
- Gap Analysis: Identify gaps in current tools and policies, such as hard-coded credentials or insufficient logging.

Steps for Future-Proofing

- Proactive Threat Detection: Implement AI-driven tools for predictive vulnerability detection and develop a dynamic risk matrix.
- Comprehensive Auditing: Expand audit policies to cover APIs, third-party libraries, and cloud integrations with automated alerts for anomalies.
- Collaboration: Foster teamwork between developers, security analysts, and architects for secure design reviews.
- Strengthen Data Security: Adopt quantum-resistant encryption and enforce robust key management.
- Adopt Standards: Align policies with NIST, ISO/IEC 27001, and OWASP Top Ten frameworks, while pursuing security certifications.
- Feedback Loops: Use audits and penetration testing results to continuously refine and enhance policies.

REFERENCES

- CERT C++ Coding Standards. (2024). CERT C++ Secure Coding Standard. Retrieved from https://wiki.sei.cmu.edu/confluence/display/cplusplus
- Microsoft. (2024). Using Static Analysis Tools for Secure Code Development. Retrieved from https://learn.microsoft.com/en-us/cpp/code-quality/
- SonarQube. (2024). Static Code Analysis Tool for Code Quality and Security. Retrieved from https://www.sonarqube.org/
- OWASP Foundation. (2024). OWASP Top Ten Security Risks. Retrieved from https://owasp.org/www-project-top-ten/
- National Institute of Standards and Technology (NIST). (2024). AES Encryption Standards. Retrieved from https://csrc.nist.gov/publications/detail/fips/197/final
- Splunk. (2024). Log Management and Monitoring for Security. Retrieved from https://www.splunk.com/
- GitHub. (2024). GitHub Advanced Security: Secret Scanning. Retrieved from https://docs.github.com/en/code-security/secret-scanning
- OWASP ZAP. (2024). Zed Attack Proxy (ZAP) for Security Testing. Retrieved from https://www.zaproxy.org/

Green Pace

REFERENCES (Continued)

- PVS-Studio. (2024). Static Analysis for C, C++, and C#. Retrieved from https://pvs-studio.com/
- Clang Static Analyzer. (2024). Clang Tools for Code Quality and Security. Retrieved from https://clang-analyzer.llvm.org/
- Axivion Bauhaus Suite. (2024). Static Code Analysis and Architecture Verification. Retrieved from https://www.axivion.com/
- Checkmarx. (2024). Application Security Testing Tools. Retrieved from https://checkmarx.com/
- TruffleHog. (2024). Detecting Secrets in Source Code. Retrieved from https://trufflesecurity.com/
- Coverity. (2024). Static Analysis Tools for Software Security and Quality. Retrieved from https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html

Green Pace