Alexander Ray 10/19/2025

**Slide 1: Title Page**

"Good afternoon, everyone. Thank you for being here. Today, I'll be walking you through our new Secure Software Development Lifecycle, or SDL, policy.

My name is Alexander, and my goal is to show you how this policy will help us build more secure products efficiently and effectively."

**Slide 2: Overview (Part 1/2) - Policy Introduction**

"Let's start with the 'why'. Our previous approach to security was reactive. We often found critical issues late in the game, which was expensive and risky.

The SDL policy is our move to a proactive model. The core principle is to 'shift security left' embedding security into every stage of the development process, right from the start.

This directly supports our 'Defense-in-Depth' strategy. While we have firewalls and network security, many modern attacks target the application layer the code we write. The SDL is our framework for hardening that critical layer."

**Slide 2: Overview (Part 2/2) - Threats Matrix & Automation**

"To know what to defend against, we use a Threats Matrix. This helps us prioritize our efforts on what matters most threats that are both highly likely and have a high impact, like SQL Injection.

The foundation of our SDL is automation. We have three types of automated tools that work together to find vulnerabilities.

First is SAST, which you can think of as a security spell-checker for our source code. It gives developers instant feedback before the code is even run.

Second is DAST, which acts like an automated hacker, testing our running application in a staging environment to find vulnerabilities that only appear at runtime.

And third is SCA, which scans all the open-source libraries we use. A vulnerability in a third-party library is still our vulnerability, and SCA helps us manage that risk.

**Slide 3: Principles**

Our policy is guided by 10 core principles. These are the fundamental rules that inform our standards and practices."

I'll highlight three key ones: First, #1 Validate Input. This is the bedrock of application security—never trust data coming from outside the application.

Second, #4 Enforce Access Control. This is the principle of least privilege. Once a user is authenticated, they should only have access to the absolute minimum they need to perform their function.

And third, #9 Use Secure Dependencies. Modern software is built, not bought. This principle acknowledges that we are responsible for the security of all the third-party components we build into our products.”

**Slide 4: Coding Standards**

"Moving from high-level principles to concrete actions, we have our coding standards. These are prioritized based on mitigating the most common and impactful attack vectors first.

As you can see, Input Validation and the use of Parameterized Queries are at the top of the list. Getting this right prevents entire classes of vulnerabilities like SQL Injection.

The list follows a logical flow: secure the inputs, protect the data with encryption, enforce strict access control, and ensure we have safe defaults and robust logging.”

**Slide 5: Encryption Strategy**

"Encryption is non-negotiable for protecting data. Our policy covers data in its three states.

In Flight: All data moving across any network, internal or external, must be encrypted with modern standards like TLS 1.2 or higher. This prevents eavesdropping.

At Rest: All data stored on hard drives, in databases, or in backups must be encrypted. If a device is ever lost or stolen, the data on it remains unreadable.

In Use: This is an advanced, forward-looking part of our policy. We are adopting technologies like secure enclaves to protect data even while it's being actively processed in memory."

**Slide 6: Triple-A Framework**

"To manage access to our systems, we use the Triple-A framework: Authentication, Authorization, and Accounting.

Authentication answers, 'Who are you?' Our policy mandates Multi-Factor Authentication across the board. A password alone is no longer considered sufficient.

Authorization answers, 'What are you allowed to do?' Here, we enforce the Principle of Least Privilege through Role-Based Access Control. You only get the keys to the doors you absolutely need to open.

Accounting answers, 'What did you do?' We maintain detailed logs of all actions. This is critical for monitoring, incident response, and proving compliance."

**Slide 7: Unit Testing**

"How do we prove our defenses work? We write tests that try to break them. This slide shows a practical example for a login function.

We test for normal behavior first—does a valid user get in? Then, we write tests that use actual attack payloads, like the classic SQL Injection string ' OR '1'='1'.

For a security test, a 'PASS' means the function correctly rejected the attack. These automated tests run with every code change, acting as a continuous security regression suite."

**Slide 8: Automation Summary**

"This diagram shows how security automation is woven into our entire development flow. It's not a separate stage, but a continuous process."

Notice the compiler's position in the Build stage. This is a critical control point where we integrate SAST and SCA scanning to catch issues before the code ever gets to a test environment.

From threat modeling in the design phase to real-time monitoring in production, automation provides the guardrails that allow our developers to move fast, but safely."

**Slide 9: Risks and Benefits**

"Let's talk about the bottom line. The problems with a manual, delayed approach to security are clear: it's slow, error-prone, and expensive.

The solution is automation. It allows us to catch issues early when they are cheapest to fix, enforce our policies consistently, and even automatically block insecure code from being deployed.

The risk of inaction is significant—it leaves us vulnerable to costly breaches and reputational damage. The benefit of acting now is a more secure product, a faster development cycle, and a stronger security posture overall."

**Slide 10: Recommendations and Conclusion (Part 1/2) - Gap Analysis**

"Our policy is a huge step forward, but we're not done. We've identified four key gaps we need to address.

First, we need to better empower our developers with real-time security tools and training.

Second, our threat modeling needs to become a continuous habit, not a one-off task.

Third, we must formalize our approach to supply chain risk. We need to be as diligent about the security of the code we import as we are with the code we write.

Finally, we need to create better feedback loops so that lessons from production incidents are used to systematically improve our security."

**Slide 11: Recommendations and Conclusion (Part 2/2) - Standards Adoption**

"To guide our improvements, we will align our program with three globally recognized standards.

The NIST Cybersecurity Framework will provide the high-level structure for managing risk.

"OWASP SAMM will give us a detailed roadmap for maturing our specific software development practices.

And ISO 27001 will ensure our program meets international standards for security management, which is critical for compliance and customer trust.

By taking these steps, we are building a security program that is not only effective today but is also structured for continuous improvement.”