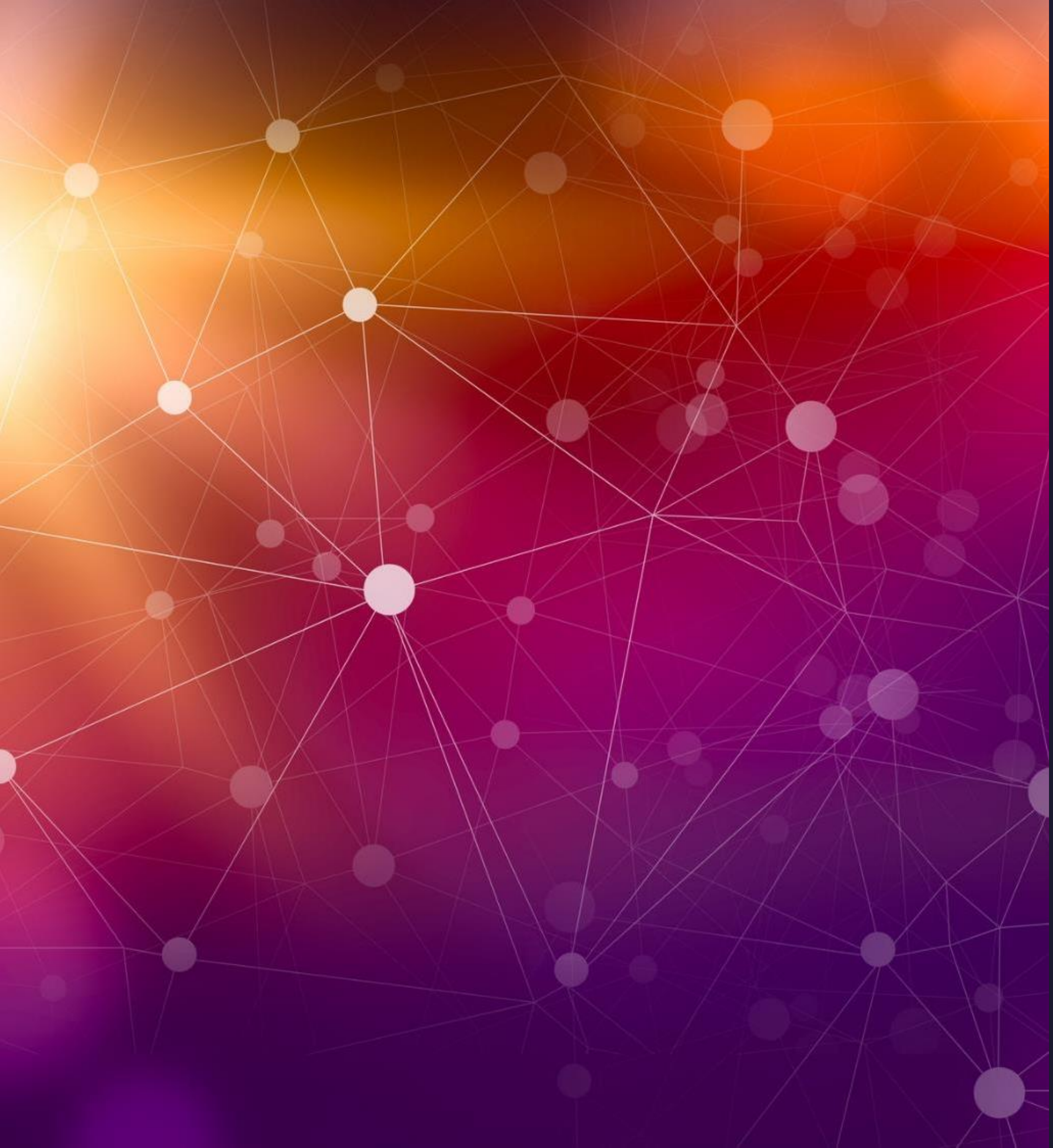


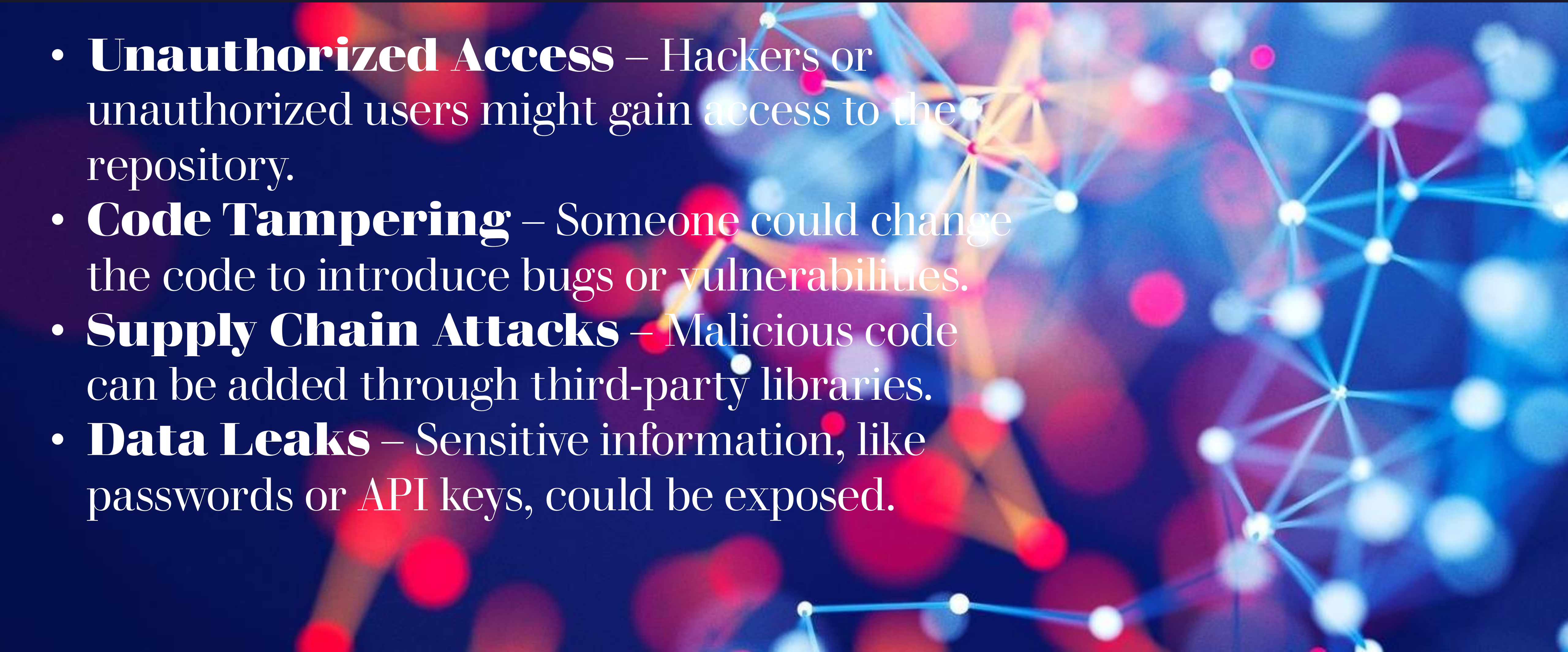
Security Controls in Shared Source Code Repositories

Jordany Gonzalez
CSD 380
March 2, 2025

Introduction

When developers work together on code, they often use shared source code repositories like GitHub, GitLab, or Bitbucket. These platforms make it easy for teams to collaborate, track changes, and build projects together. But just like anything on the internet, if security isn't taken seriously, things can go wrong. Hackers could steal or mess with the code, sensitive information could get leaked, or malicious software could be added without anyone realizing it. That's why it's important to have strong security controls in place. In this presentation, we'll go over the best ways to keep shared repositories safe and secure.

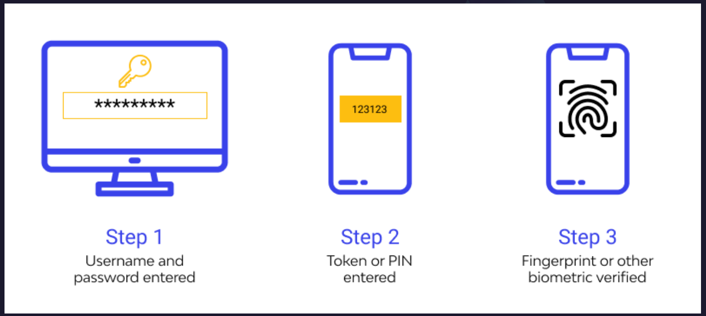
Common Security Threats in Shared Repositories



- **Unauthorized Access** – Hackers or unauthorized users might gain access to the repository.
- **Code Tampering** – Someone could change the code to introduce bugs or vulnerabilities.
- **Supply Chain Attacks** – Malicious code can be added through third-party libraries.
- **Data Leaks** – Sensitive information, like passwords or API keys, could be exposed.

Implementing Strong Authentication & Access Controls

One of the best ways to protect a shared repository is to control who can access it. Using strong passwords isn't enough anymore—multi-factor authentication (MFA) should be required for all users. This means that even if someone's password gets stolen, they won't be able to log in without an extra security step, like a code sent to their phone. It's also important to limit who has access to certain parts of the repository. For example, not everyone needs permission to delete or modify important files. By following the **least privilege principle**, teams can make sure that only the right people have access to critical areas of the code. Platforms like GitHub and GitLab have built-in security settings to help enforce these controls.


****
123123
Step 1
Username and password entered
Step 2
Token or PIN entered
Step 3
Fingerprint or other biometric verified

Enforcing Secure Code Reviews & Approvals

- **Pull Request (PR) Reviews:** Require at least one other person to approve code changes before merging.
- **Automated Security Scans:** Use tools like SonarQube and Snyk to check for security issues.
- **Branch Protection Rules:** Prevent direct changes to the main code without approval.
- **Example:** GitHub allows admins to require code reviews before changes are made.

Securing Secrets & Sensitive Data

One of the biggest mistakes developers make is accidentally including passwords, API keys, or other sensitive data in their repositories. Once something is pushed to a public or even private repository, it can be difficult to remove it completely. Instead of storing sensitive information in the code, developers should use secret management tools like HashiCorp Vault or AWS Secrets Manager. Another good practice is to set up **Git hooks** that automatically prevent sensitive data from being committed to the repository. Some platforms, like GitHub, even offer **secret scanning tools** that can detect and alert developers if something sensitive is exposed.

Monitoring & Auditing Repository Activities

- **Enable Logging & Monitoring:** Keep track of who accesses and modifies the repository.
- **Audit Logs:** Regularly check logs for any suspicious activity.
- **Alerting Systems:** Set up automatic notifications for unusual changes.

Example: GitHub Enterprise and GitLab have built-in logs that help track repository actions.

Securing Third-Party Dependencies

Most software projects rely on third-party code, like open-source libraries, to work properly. While these dependencies save time, they can also introduce security risks if they aren't carefully managed. Attackers have been known to sneak malicious code into widely used libraries, which can then infect thousands of projects. To prevent this, developers should use automated tools like **Dependabot** or **Snyk** to check for outdated or vulnerable dependencies. Another smart practice is to only use libraries from trusted sources and regularly update them to the latest secure versions. Organizations that take software security seriously often require **Software Composition Analysis (SCA)**, which scans all dependencies for known security risks before they are used in a project.

References

- National Institute of Standards and Technology (NIST). "SP 800-63: Digital Identity Guidelines."
- OWASP. "OWASP Secure Coding Practices Guide."
- GitHub Docs. "Security Features and Best Practices."
- SonarQube. "Automated Static Code Analysis for Secure Development."