CODE SECURITY REVIEWS

Stephen Schaub

Approaches to Security Verification

- Penetration testing
- Static analysis tools
- Code reviews

Why Do Code Reviews?

- Code reviews are perhaps the single most effective technique for identifying security flaws
- Code reviews are required for compliance to certain industry standards and government regulations
- Use in combination with other approaches

Audit vs. Collaboration

- Developers hate audits
- Create an atmosphere of collaboration
- Reviewer as advisor, not policeman

Preparation

Reviewers must be familiar with

- Application Platform
- Application Context
- Application Audience
- Application Availability Requirements

Ensuring Good Value

- Want to ensure reviewers find most important risks, and not focus on inconsequential issues
- Reviewers must understand context of application/module being reviewed
- Prepare a threat model with answers to key questions:
  - What type/how sensitive is the data/asset in the application/module?
  - Is the application/module internal or external facing? Are the users trusted?
  - How important is the application to the enterprise?

Code Crawling

- Not necessary or efficient to review every line in application/module
- Focus on areas that may have application security implications
- Code Crawling involves using tools to identify areas of application to review
  - Tools are usually simple text search (grep, editor find function)
  - Search for uses of key API's (ex. req.query, req.body)

Attack Surface

- **Attack surface** is the set of application interfaces an attacker can use to perform unauthorized activity
- Inputs can come from
  - HTTP
  - Configuration files
  - Data feeds
  - Environment variables
  - …

Reviewing Techniques

Vulnerability: Broken Access Control

1. Determine what assets need to be secured
2. Determine what security controls are needed to secure the assets
3. Determine whether security controls are in place and correctly implemented

Consider examples/express/carscsrf

Vulnerability: Security Misconfiguration

- Checklist:
  - Are error stack traces revealed to end users?
  - Are default accounts / passwords unchanged?
  - Are unneeded features such as directory listings enabled?

Vulnerability: Injection

1. Search for use of API's that evaluate expressions or connect to external systems:
   - Database SQL interfaces
   - Command Shell interfaces
   - JavaScript/Python eval()
2. Inspect lines of code that call these API's for insertion of data from program variables
   - SQL: Does it use a safe mechanism for inserting variable data?
3. Determine where info in program variables came from
   - Untrusted source?
   - If so, has it been appropriately sanitized/escaped and/or validated?

Vulnerability: Cross-Site Scripting

- Inspect code that generates HTML responses
  - For Node Express/Handlebars apps, begin with the .hbs files
- Look for variables inserted into the output
  - For variables that hold data from untrusted sources, must check that information has been appropriately sanitized, validated, and/or encoded for the context where it appears
  - Work backwards to find the source of each variable

Vulnerability: Cross-Site Request Forgery

- Checklist:
  - Does the application use a CSRF protection library?
  - Is it correctly configured?
  - Are all routes that update the database POST, and not GET?