Secure Code Changes

Lotfi ben Othmane and Moataz Abdelkhalek
Iowa State University, USA

2018 Annual Computer Security Applications Conference
December 6, 2018
San Juan, Puerto Rico
Problem

- Companies develop and deploy frequent releases of their software.

- PB: How to maintain the security of the software when the code changes?

```java
public class MyClass {
    public void Method_A() {
        // Do Something
        FileIOPermission myPerm =
                new FileIOPermission(PermissionState.Read);

        myPerm.Demand(); // Do Something
    }
}
```
Current solutions

- Companies use two common approaches:
  - Perform full security assessment of the software in each new release.
  - Use keywords, e.g., encrypt, secure, hash.
  - Use notations and peer-review of all the changes by the senior developers.

- Both approaches are:
  - Impractical
  - Time-consuming
  - Expensive
  - Cause conflicts based on the different understandings of software architecture.
Research Question

How to trace the impacts of code changes on the security of a given software?
Approach

1. Model the security assurance of the software using security assurance cases.
2. Associate code parts to security claims/requirements.
3. Related code changes to attack surface entry points.

=> Relate assurance case elements to code changes.
Security Assurance Case

C1: User authentication credentials are protected in storage
  - E1: Verify hashing of the user's password
  - E2.1: Verify Password Complexity Settings
  - E2.2: Verify all default passwords are omitted
  - E2.3: Verify multi-factor authentication is used

C2: No Weak Credentials used (e.g., account creation, change password, recover password, weak session IDs)
  - E3.1: Verify if SIDs are not imposed in the URL (e.g., URL rewriting)
  - E3.2: Verify no session fixation attacks
    - E3.2.1: SIDs timeout after a specific amount of time
    - E3.2.2: SIDs are rotated after successful login
    - E3.2.3: SIDs or authentication tokens are properly invalidated during logout, idle, and absolute timeouts

C3: Strong Session Management
  - E4: Verify the use of encrypted connections for passwords and session IDs transmission.
  - E5.1: Verify failed login attempts are limited
  - E5.2: Verify using the same generic error message for all outcomes
  - E5.3: Verify logging of all failures in contrived SEM solution and alerting of administrators

C4: Secure transmission of Passwords, session IDs, and other credentials
  - E6.1: Verify personal security questions are not used
  - E6.2: Verify using the same generic error message for all outcomes
  - E6.3: Verify usage of phone number and alternate email address for password recovery

C5: Protection Against automated attacks such as brute forcing
  - E6.4: Verify personal security questions are not used
  - E6.5: Verify using the same generic error message for all outcomes
  - E6.6: Verify usage of phone number and alternate email address for password recovery

C6: Strong effective registration, credential recovery, and forgot password processes

Main Claim: Safe Authentication and Session Management

Strategy: The main claim is achieved by satisfying all the sub claims
Example of Call Graph -- WALA
Call Graph – Security Functions
Example Claim – Strong Session Management

1. Verify if SIDs are not exposed in the URL (e.g., URL rewriting)
2. Verify No session fixation attacks
3. SIDs timeout after specific amount of time
4. SIDs are rotated after successful login
5. SIDs or authentication tokens are properly invalidated during logout, idle, and absolute timeouts.

Related nodes
- compiere/process/SessionEndAll, main
- compiere/process/SessionEndAll, clinit
Any Comments!

Thank you

Lotfi Ben Othmane

othmanel@iastate.edu
Example of Call Graph -- WALA