Application Logic Flaws

Professor Larry Heimann
Web Application Security
Information Systems
Discussion of Lab 6

• Due in one week on October 18th (11:30am)

• Complete findings record, attach screenshots and/or code files to confirm

• Don’t look at Ruby code

• Don’t collaborate with others

• Grades will be curved if necessary
“You cannot defend against threats you cannot see.”

-- Mr. H, chess coach

“You cannot defend against threats you cannot see.”

-- Prof. H, 67-327
“Zwischenzug”
White is expecting Black to capture the bishop on b8, after which he will win a piece with 10. Qa4+ and then 11. Qxb4
Black surprised White by playing 9. ... Nd5! first, which protects the bishop on b4 and threatens a knight fork on e3. *Zwischenzug.*
Logic flaws ("Zwischenzug")

• Logic is everywhere in applications.

• Presents an intricate attack surface that is often overlooked.

• Flaws like XSS and SQL injection receive more attention because they have a standard “signature”.

• Flaws in application logic are harder to characterize, and may appear to be one-offs.

• Not usually identified by vulnerability scanners.

• Therefore ... logic flaws are of great interest to attackers.
Nature of logic flaws

• All involve some defect in the application’s logic.

• Developer reasons:

  "If A happens, then B must be the case."

• They don’t ask:

  "But what if X occurs?"

  where X violates some assumption in their reasoning.

• Best way to learn about logic flaws is by example.
Case 1: Password change function

• Password change functions usually ask for your existing password.

• When functions are used by administrators to reset users’ passwords, they don’t ask for the existing password.

• Defective assumption: if the existing password parameter is not submitted, the request is being made by an administrator:

```java
String existingPassword = request.getParameter("existingPassword");
if (null == existingPassword)
{
    trace("Old password not supplied, must be an administrator");
    return true;
}
else
{
    trace("Verifying user's old password");
    ...
}
```

• Any user can change another user’s password by removing the “old_password” parameter altogether (both name and value).
Case 2: Avoiding payment

• A typical multi-stage purchase process:
  • Add items to shopping cart
  • View cart and select “checkout”
  • Enter payment information
  • Enter delivery information

• Defective assumption: if a user reaches stage #4, they must have passed through stage #3.

• Any user can purchase items for free by forcing their browser to skip stage #3.
Forced browsing

• A key technique for finding and exploiting logic flaws.

• Circumvents any controls on navigation imposed by the browser.

• Involves accessing functionality out of the expected sequence.

• Directly applicable to multi-stage processes – need to consider every GET and POST request made, including redirects.

• Can be used to reach privileged functions if access control is only imposed by the interface.

• Also involves submitting request parameters to unexpected locations or in unexpected sequences.

• Can be used to uncover subtle but devastating flaws.
Case 3: Banking registration

• Existing banking customers can register for online banking.
• Users supply some basic personal details, and are sent login info by mail.
• Defective assumption: no way for users to self-register and gain direct access to sensitive information.

But ...

• The developers used an existing application component to track the user’s identity during self-registration.
• When personal information has been processed, an object representing the user’s identity is instantiated and stored in their session.
• The same object is used by the main application functionality to control access.
• Hence, an attacker could access any customer’s account by submitting the relevant data in registration, and then proceed to the protected function.
Discovering logic flaws

• Use forced browsing to access multi-stage functions out of sequence.

• Transmit parameters to different functions where they are not expected.

• Remove individual request parameters altogether (name and value).

• Work systematically, targeting each parameter and function in turn.

• Include every request in your testing, including auto-generated redirects.

• In situations where users transition between different trust levels, determine whether you can accumulate appropriate state to make the transition in an unauthorized way (as in banking registration).

• Where numeric limits and checks are enforced, try unusual input (like negative numbers) to defeat the logic being applied.

• When probing input validation logic, always check if the escape character is being handled safely.
Avoiding logic flaws

• No silver bullet – need to apply good practice and think laterally.

• Document every detail of the application’s design in sufficient detail for an outsider to understand.

• Explicitly document every assumption being made within the design – this step alone will cause many unsafe assumptions to be identified.

• Mandate that all source code is clearly commented to include the following:
  • The purpose and intended uses of each code component.
  • The assumptions made by each component about anything that is outside of its direct control.
  • References to all client code which makes use of the component.
Avoiding logic flaws

• Perform security-focused design reviews:
  • Identify and evaluate every assumption being made.
  • Try to identify circumstances in which they might be violated.

• Perform security-focused code reviews, and in particular consider:
  • The ways in which unexpected user behavior and input will be handled by the application.
  • The potential side-effects of any dependencies and interoperation between different code components and application functions.
  • Beware of “GOD” objects and how they are being used (or can be abused)
Summary

- Logic flaws are very widespread.
- Often appear to be one-offs, but many common themes exist.
- They are not going to go away, and cannot be detected by vulnerability scanners.
- Many can be detected using the standard testing steps described.
- Others require a degree of lateral thinking.
- Try to think like a developer ...
  - Imagine you were working to a tight deadline, focusing on functionality not security, modifying an existing code base, and using someone else’s APIs.
  - What would you get wrong?
Comic of the Day...

http://xkcd.com/792/