Introduction to security code review for the web

Louis@pentesterlab.com

PentesterLab

Picking a methodology

- What are you trying to achieve?
  - Find 0days?
  - Software free of exploitable bugs?
  - Secure code/following best practices?
  - Code not backdoored?
- How much time do you have?
  - An hour
  - A week
  - More

Read everything

- Good for coverage

  To an extend…

- Good when you tackle a new framework

- Time consuming

PentesterLab

Grep

- Search for vulnerable functions or vulnerable patterns
- Great for finding low hanging fruits
- Relies on the fact that you know what you are looking for

PentesterLab

Random

- Open a file, start reading, find bugs
- Great if you are lucky
- Great if you have enough time
- Low level of assurance

PentesterLab

Follow user input

- For each available “page/URI”:
  - Review all the steps the user’s inputs are going through
- Time consuming
- Great level of assurance

PentesterLab

By functionality

- Pick a functionality, review the code involved
  - "reset password"
  - "authentication"

- Best results if used across multiple projects:
  - "Session management across Java frameworks"
  - Compare and find what controls are missing

PentesterLab

Tips & tricks

PentesterLab

Don’t trust anyone

- Don’t trust comments

```
// encrypt data
```

- Don’t make assumptions
  - *“it’s probably secure”*
  - *“it’s encrypted”*
  - *“it’s signed”*
  - *“Let’s assume the framework is secure and focus on the custom development”*

PentesterLab

Don’t trust anyone

```
public class TrustAllX509TrustManager implements
X509TrustManager {
    public X509Certificate[] getAcceptedIssuers(){
        return new X509Certificate[0];
    }
    public void
checkClientTrusted(java.security.cert.X509Certificat
e[] certs,  String authType) {
    }
    public void
checkServerTrusted(java.security.cert.X509Certificat
e[] certs, String authType) {
    }
}
```

PentesterLab

Compare

- All these frameworks have this check, this one doesn't
- All these methods have a @Valid for the 2nd argument, this one doesn't
- …

PentesterLab

Compare

- For CRUD applications:
  - check the controls in place for the index/list functions
  - Compare with the controls in place for Read, Delete, Update

It's more likely that the controls on index/list are correct, otherwise other people's records will be visible.

Understand the code

- Difference in mapping between URL and code:
  - Based on the file system
  - Convention over configuration
  - Based on a “route” file
  - XML

Use the existing tests

- See what is checked and **what isn't**
- Check what tests get modified to work with the new version of the code
- Write your own tests

PentesterLab

Déjà-Vu

- Read advisories:
  - Review the code changes
  - Add the pattern to your known patterns list
  - Review the same issue on other (similar) code base

You're unlikely to find a new bug class. You're likely to find new instance of a know bug class or a known bug class with a small twist.

Have a “lab”

- Get/build a sample application that uses the same framework
- Create small snippet to validate behaviour

```
🍺 koriander ~/code/java % ls
OneTimePassword.class   TestFile.java          TestRemove.class
OneTimePassword.java    TestHashMap.class      TestRemove.java
ScriptEngine            TestHashMap.java       TestToken.class
Session.class           TestJSF.class          TestToken.java
Session.java            TestJSF.java           TestTrustFact.class
Test.class              TestLong.class         TestTrustFact.java
Test.java               TestLong.java          TestURL.class
TestAnnotation.java     TestMatch.class        TestURL.java
TestBigInt.class        TestMatch.java         TestUUID.class
TestBigInt.java         TestMatch2.class       TestUUID.java
TestCanonical.class     TestMatch2.java        int_no_sqli
TestCanonical.java      TestPattern.class      processbuilder
TestEquals.class        TestPattern.java       ssl
TestEquals.java         TestRandom.class
TestFile.class          TestRandom.java
```

Know the language/framework pitfalls

- PHP comparison: `==` vs `===`
- PHP `header()` that doesn't stop the execution flow
- Ruby end-of-line in regular expression:

  `/\Atest\z/` vs `/^test$/`

- Java URL class that handles `file://`

PentesterLab

Example

PentesterLab

Context

- Web application that allows users to download files
- To limit access, the application signs the file's name
- The application verifies the signature and serve the file if the signature is valid

PentesterLab

Snippet 1

```
if signature && !valid?(filename,
                        signature)
  return :unauthorized
else
  send_file filename
end
```

PentesterLab

Snippet 1

```
if signature && !valid?(filename,
                        signature)
  return :unauthorized
else
  send_file filename
end
```

- What if a user doesn't provide a signature

Snippet 2

```
def valid?(filename, signature)
  return sign(filename) == signature
end
```

PentesterLab

Snippet 2

```
def valid?(filename, signature)
  return sign(filename) == signature
end
```

- Non time-constant comparison of strings

Snippet 3

```
def sign(filename)
   return md5(getsecret()+filename)
end
```

Snippet 3

```
def sign(filename)
   return md5(getsecret()+filename)
end
```

- This code is vulnerable to length-extension: an attacker can generate valid hash by "continuing the hashing function"

PentesterLab

Snippet 4

```
def getsecret()
  return “superlongstring”
end
```

PentesterLab

Snippet 4

```
def getsecret()
   return "superlongstring"
end
```

- Hard coded secret
- Trivial secret (offline attack)
- Should use a key derivation function (PBKDF2)

PentesterLab

Conclusion

- Given enough code, you will find security issues (that may be exploitable)

- Being able to review code will get you to the next level:
  - Better at spotting bugs in black-box tests
  - Better at fixing bugs

PentesterLab

Questions?

PentesterLab