Security Practical Answers

Dr Chris G. Willcocks

Last Modified: October 21, 2018

Practical 1

Prevent dictionary attack

This solution uses heat to prevent a dictionary attack. While they can do it any way they wish, this approach is efficient as it requires no data structures or loop iterations. Also, the heat does not need to be decremented in a loop as you can simply keep track of when the last heat was added. There are many different ways to code this, each with its own implications, and this serves as a basic example:

Listing 1: Java

```
import java.lang.Math;

...

private double heat = 0;
private long lastHeat = 0;

public void run() throws Exception {
    BasicAuthenticator auth = new BasicAuthenticator("get") {
        @Override
        public boolean checkCredentials(String user, String pass) {
            // Update heat
            heat -= 0.0001*(System.currentTimeMillis()-lastHeat); // cooldown
            heat = Math.min(Math.max(heat, 0.0), 1000.0); // constrain heat within range to
                prevent blacklisting exploit
            heat += 1.0;

            if (heat < 5.0) { // make sure we're cool enough to accept logins
                // Check if login was successful
                ...
            }
            // Else there was a failed login attempt
            lastHeat = System.currentTimeMillis();
            System.out.println("Heat: " + heat);
            return false;
        }
    };
    // Setup and start the server
    ...
}
```

Hash passwords

They can either store their hashes as bytes arrays or base64 encoded strings. Here's some examples:

Listing 2: Java

```java
import java.security.MessageDigest;
import javax.xml.bind.DatatypeConverter;

...

private List<String> hashesSHA_256 = Arrays.asList(
    "CF0B854F5A17FDAD773D462438D4D7328722B817D40A74ECB8D9AD79F98AA251",
    "28F6155150517676AFDEF5E125740B054CA6E8E3655FCFC4D294ECD240654392",
    "E8F56862D74EF5599AF4EECA73924BFA44A6773A497AF0C29C48E18729BA6FF0",
    "059A00192592D5444BC0CAAD7203F98B506332E2CF7ABB35D684EA9BF7C18F08",
    "28F6155150517676AFDEF5E125740B054CA6E8E3655FCFC4D294ECD240654392"
);

private List<String> hashesSHA_1 = Arrays.asList(
    "11273D57B954F7B4A41CEE3F98C2F90BC80D2F59",
    "1D799D2F9BC2C79DA3F88238CF532763298F10EB",
    "E0C95748A455C27A80FD289269120D4944D1F318",
    "C6922B6BA9E0939583F973BC1682493351AD4FE8",
    "1D799D2F9BC2C79DA3F88238CF532763298F10EB"
);

...

public String passToHash(String pass) {
    try {
        byte[] messageBytes = pass.getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] digest = md.digest(messageBytes);
        String hex = DatatypeConverter.printHexBinary(digest);
        return hex;
    } catch(Exception e) {
    }
    return null;
}

...

if (user.equals(users.get(i)) && passToHash(pass).equals(hashesSHA_256.get(i))) {

...
```

Salt passwords

They can either store their hashes as bytes arrays or base64 encoded strings. Here's some examples:

Listing 3: Java

```
private List<String> hashesSHA_1 = Arrays.asList(
    "7B8E5558BE44B9CE7A78CFCD2FA9B0FC9CF885BB",
    "B9CF0122C4020115F187361D484A62748092DA43", <-- Ensure this is different
    "B86BE932B76403135112B6CDC1D34B81F0D6DFF1",
    "282863582F79C89FFC6F7BD313DDCD7C5743CEC5",
    "C58478B2660AD52D7E40E05D0803238C64609A10"  <-- Ensure this is different
);

private List<String> salts = Arrays.asList(
    "EEB70DECD28A6FC4",
    "8A76B21D1AE3B0CC",
    "C3EBB7D97544EC24",
    "DD1C28FFE094E18C",
    "85B73453F2FC12C4"
);

public String passToHash(String pass, String salt) {
    try {
        byte[] messageBytes = (pass+salt).getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] digest = md.digest(messageBytes);
        String hex = DatatypeConverter.printHexBinary(digest);
        return hex;
    } catch(Exception e) {
    }
    return null;
}

...

if (user.equals(users.get(i)) && passToHash(pass, salts.get(i)).equals(hashesSHA_1.get(i)))

...
```