Seamless Upgrades for Credential Security in Apache Tomcat
Christopher Schultz
Chief Technology Officer
Total Child Health, Inc.

Password Security Failures

- Lifeboat (Minecraft) (MD5)
- Ashley Madison (bcrypt.... but also MD5)[1]
- VTech (MD5)[2]
- LinkedIn (SHA-1)
- Pre-NT Microsoft Windows passwords (awful DES-based algorithm, 14 chars max, case-insensitive)[3,4]
- Microsoft Outlook (CRC32) [3]

Password Security Failures

- No credential security (plaintext/cleartext)
- Rolling your own security
  - Existing tools are inconvenient
  - NIH syndrome
- Using known poor or outdated algorithms
  - MD5, SHA1
- Using inappropriate algorithms
  - Simple hashes (e.g. MD[0-9], SHA-[1-9]+)
Password Security Failures

- Bad credential security means that users are at risk, even when they aren't using your application.
- Note that this is different than application security, where the service itself is at risk, not necessarily the users.
What Exactly Are We Protecting?

- Only really protects the user database
  - Container protects the application from users
  - Application protects the data from users
- Mitigates an attack where the user database is stolen
  - Might have bigger problems on your hands
- User database is still important
  - May allow lateral attacks against other services
    - email, finance, medical records
  - Even admins shouldn't have users' passwords
What Exactly Are We Protecting?

- Think your user database won't be stolen?
- Just ask LinkedIn, eHarmony, and Last.fm
  - All hacked within a week in 2012
  - All had their user databases published
User Database Attacks

- User database contents
  - Username
  - Email address
  - Credentials (password)
- Username and/or email address may be valid elsewhere
  - Password might be valid elsewhere, too
- Compromise of one user database may allow access to other services
User Database Attacks

• Many users aren't very creative when it comes to setting passwords
  – 5up3rsecr37!
  – firstname2016

• Many users “know” that re-using passwords isn't a good idea
  – They use a “high-security” password only for high-security sites (e.g. bank)
  – What happens when your bank's user database gets hacked?
Attacking User Databases

- **Cleartext**
  - Trivial: password is right there

- **Simple hashing algorithms (MD5, SHA1, SHA2)**
  - Rainbow tables
  - Online services with massive hash databases

- **Salted hashes**
  - More difficult, often requires brute-force

- **Key-derivation algorithms (PBKDF2, bcrypt)**
  - Very difficult, usually requires brute-force
Determined Adversaries

- A quick note about a determined adversary
  - Well-funded and state-level adversaries have the computing resources to brute-force many algorithms
  - If your user database has been compromised, any individual user should be considered compromised
  - Which user? Who knows...

- Best strategy is to use the highest security available to you in all cases
  - Use a password-hashing algorithm
Key-Derivation Algorithms

• Difficult by design
  – Slow – many dependent operations
  – High memory requirements

• Compare to message-digest hashing algorithms
  – Very fast
  – Often implemented directly in hardware
Key-Derivation Algorithms

- PBKDF2 (1991)
  - NIST standard
  - FIPS-140 compatible
  - No known weaknesses

- bcrypt
  - Open-source origin (1999)
  - Non-standard, based upon Blowfish cipher
  - Can be tuned to be arbitrarily expensive (iterations)
  - No known weaknesses
Historical Tomcat Support

- Tomcat has supported simple message-digest-based algorithms since at least Tomcat 3.x
  - Anything java.security.MessageDigest supported
  - No salting
  - No iterations
  - No 3\textsuperscript{rd}-party plug-ins
- Using custom credential-manipulation code required a custom Realm
  - Realms must support lots of unrelated stuff
Historical Tomcat Support
Modern Tomcat Support

- Still supports message-digest-based algorithms
  - java.security.MessageDigest
  - Backward-compatible
  - Adds salting and iterations if desired
- New pluggable CredentialHandler interface
  - Sky is the limit
- Included CredentialHandler implementation
  - PBKDF2 (if supported by JVM)
  - Good example for custom implementations
Modern Tomcat Support
Modern Tomcat Support

- Does not support other algorithms like bcrypt, etc.
  - Possible licensing issues, need to pick a vendor
  - Did not want compile-time dependency on 3<sup>rd</sup>-party library
  - Easy enough to plug-in, not a high-priority to include in Tomcat's distribution
Modern Tomcat Support

- Includes NestedCredentialHandler
  - Allows more than one CredentialHandler to be used
  - This allows for seamless upgrades between algorithms
CredentialHandlers

- Simple Java interface

```java
public interface CredentialHandler {
    boolean matches(String inputCredentials,
                     String storedCredentials);
    String mutate(String inputCredentials);
}
```

- Easy to implement anything you want
- Interface can be used to mutate as well as validate
  - Can use directly in your applications
Using CredentialHandlers

● Easy to configure

```
<Realm
className="org.apache.catalina.realm.DataSourceRealm"...>
  <CredentialHandler
className="org.apache.catalina.realm.MessageDigestCredentialHandler" algorithm="MD5" />
</Realm>
```

● Above configuration is NOT RECOMMENDED
  – Uses insecure MD5 hashing algorithm
Using CredentialHandlers

• Easy to improve security

```xml
<Realm
    className="org.apache.catalina.realm.DataSourceRealm"...>
    <CredentialHandler
        className="org.apache.catalina.realm.MessageDigestCredentialHandler" algorithm="MD5" saltLength="16" iterations="10000" />
</Realm>
```

• Above configuration is more secure than pure MD5
  - Uses salted passwords
  - Uses many MD5 iterations
Aside: Salted Hashes

- A “salt” is a nonce used to add randomness to something that is not random at all (i.e. passwords aren't random). A nonce is one-time use “word”.

- Stored salted passwords look different from each other even when the password is the same, since the nonce is different

- Example: password=\texttt{tiger}, salt=982736549
  salted password=982736549\texttt{tiger}

- System stores both the salt and the hashed salt+password as the credential

- This (usually) defeats rainbow table attacks
Using CredentialHandlers

• Easy to significantly improve security by using a key-derivation algorithm

```xml
<Realm
  className="org.apache.catalina.realm.DataSourceRealm"...>
  <CredentialHandler
className="org.apache.catalina.realm.SecretKeyCredentialHandler" />
</Realm>
```

• Above configuration is very secure
  – Uses PBKDF2 algorithm (default)
Using CredentialHandlers

- Looks like great stuff
- But all my users have MD5-based passwords
- How is this relevant for me?
Upgrading CredentialHandlers

- Easy to migrate from one strategy to another

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"...>
  <CredentialHandler
      className="org.apache.catalina.realm.NestedCredentialHandler">
    <CredentialHandler
      className="org.apache.catalina.realm.SecretKeyCredentialHandler" />
    <CredentialHandler
      className="org.apache.catalina.realm.MessageDigestCredentialHandler"
      algorithm="MD5" />
  </CredentialHandler>
</Realm>
```

- Above configuration will support both systems
  - First tries PBKDF2
  - Falls-back to MD5
Upgrading CredentialHandlers

- Easy to migrate from one strategy to another

```xml
<Realm className="org.apache.catalina.realm.DataSourceRealm"...>
  <CredentialHandler
      className="org.apache.catalina.realm.NestedCredentialHandler">
    <CredentialHandler
        className="org.apache.catalina.realm.SecretKeyCredentialHandler" />
    <CredentialHandler
        className="org.apache.catalina.realm.MessageDigestCredentialHandler" algorithm="MD5" />
  </CredentialHandler>
</Realm>
```

- Above configuration will support both systems
  - First tries PBKDF2
  - Falls-back to MD5

  It is **vitally** important not to configure plaintext as a fall-back!
Using CredentialHandlers

- Looks like great stuff
- But all my users have MD5-based passwords
- How can I get my users to change to something better?
CredentialHandlers in Webapps

• Tomcat makes the CredentialHandler available to applications through the application context

```java
CredentialHandler ch = (CredentialHandler)application.getAttribute(Globals.CREDENTIAL_HANDLER);
String stored = ch.mutate(plaintext);
// update stored credentials in user database
```

• Applications can use Tomcat's API directly

* Since Tomcat 9.0/8.5, 8.0.34, and 7.0.70
CredentialHandlers in Webapps

• Use reflection if you don't want Tomcat as a build-time dependency
  – Avoid build-time dependencies via reflection

```java
Class<?> globals = Class.forName("org.apache.catalina.Globals");
String attrName = (String)globals
    .getDeclaredField("CREDENTIAL_HANDLER").get(null);
Object ch = context.getAttribute(attrName);
Class<?> ich =
Class.forName("org.apache.catalina.CredentialHandler");
Method mutateMethod = ich.getMethod("mutate", new Class[]
{ String.class} );

String stored = (String)mutateMethod.invoke(plaintext);
```

• Same effect with simpler dependencies
CredentialHandlers in Webapps

- Can also check existing credentials
  - Verify current password before update
  - Check password history

```java
CredentialHandler ch = (CredentialHandler)application.getAttribute(Globals.CREDENTIAL_HANDLER);
if(ch.matches(old_password, stored)) {
    // Allow update
} else {
    // Invalid current password! Veto profile update!
}
```
Custom CredentialHandlers

- Support currently-unsupported algorithms
- Don't roll your own security
- CredentialHandler should be plumbing code, not an algorithm implementation
Custom CredentialHandlers

- Support currently-unsupported algorithms
- Don't roll your own security
- CredentialHandler should be plumbing code, not an algorithm implementation
- Don't roll your own security
Custom CredentialHandler

- Let's implement bcrypt
Custom CredentialHandler: bcrypt

- Let's implement bcrypt
- Choose a Java implementation
  - https://github.com/jeremyh/jBCrypt (Maven fans)
- Understand the existing API
- Wire-into a simple CredentialHandler class
Custom CredentialHandler: bcrypt

- Implementation is trivial

```java
public class BCryptCredentialHandler {
    public boolean matches(String inputCredentials, String storedCredentials) {
        return BCrypt.checkpw(inputCredentials, storedCredentials);
    }

    public String mutate(String inputCredentials) {
        return Bcrypt.hashpw(inputCredentials,
                              Bcrypt.gensalt(getLogRounds(), getRandom()));
    }
}
```

I've left out some support details like get/setLogRounds, and a SecureRandom member. Full implementation is available along with these slides online.
Custom CredentialHandler: bcrypt

• Configuration is trivial

```xml
<Realm className="org.apache.catalina.realm.DataSourceRealm"...>
  <CredentialHandler
      className="my.package.BCryptCredentialHandler"
      logRounds="12" />
</Realm>
```

• Tomcat handles calling our `setLogRounds` method

• Make sure your stored-password field can support the format (60 ASCII characters in this case)
Custom CredentialHandler: bcrypt

- Passwords are now stored in bcrypt format
  - $2a$12$SGvTib1z7PiNihnoV7zJyuiq214MyQF/JdJE0gwuozi0OwUgDeqIi

- Compare to MD5
  - 84da2a74e610e8029431a6540c07d66b

- Compare to plaintext
  - Tomcat is the best
Tomcat Authentication

- Historically, Tomcat only supported MessageDigest-based credential security, and building a custom solution was cumbersome.
- Recent Tomcat versions (since late 2014) support pluggable CredentialHandlers which significantly simplifies this process; support for better algorithms is now included with Tomcat.
Tomcat Authentication

- Plugging-in new algorithms (e.g. bcrypt) is trivial
- Applications can access the CredentialHandlers directly if necessary
- Users' existing passwords can be migrated to higher-security storage schemes
Questions

Sample code available in the same directory.