
TRIDIUM
NS2024
POWER OF PARTNERSHIP

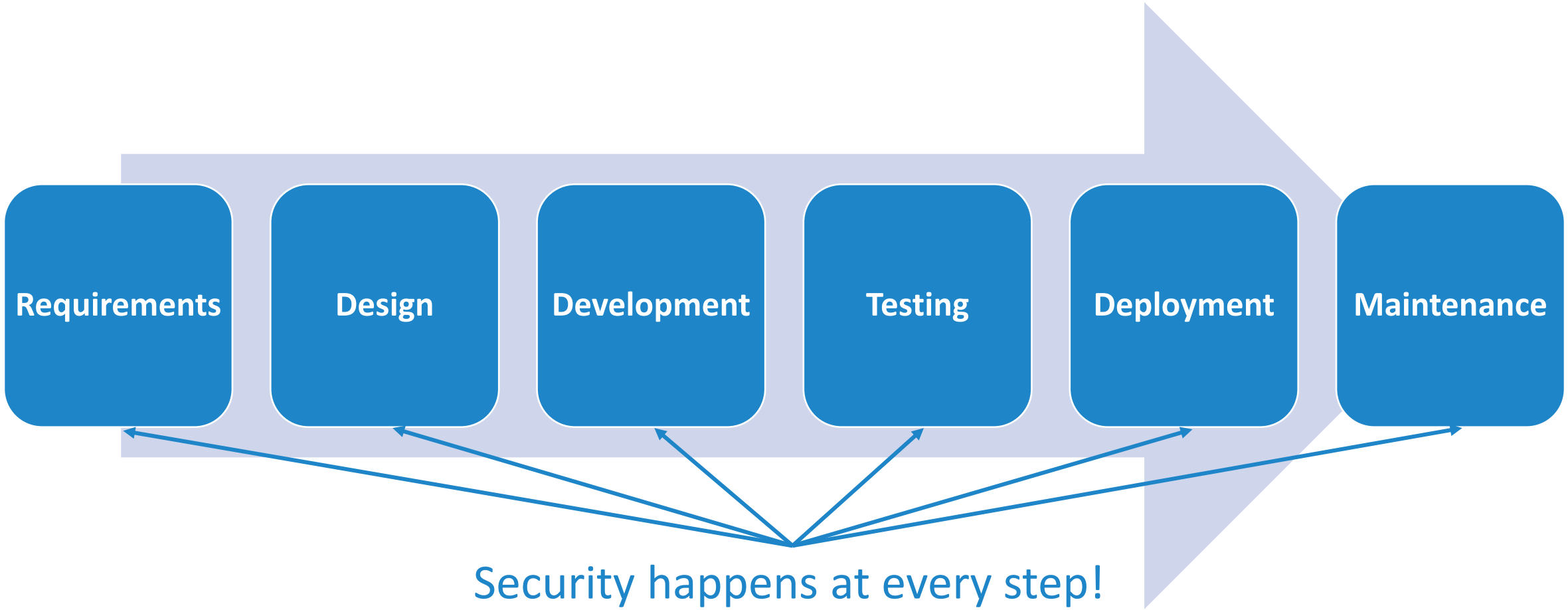
TRIDIUM

Secure Coding 101

Melanie Coggan
Software Engineer, Security Advocate

TRIDIUM

When Does Security Happen?



Secure Implementation Best Practices

Security Coding Standards

- Defines best practices
- Helpful reference to developers

Static Code Analysis

- Can detect security issues
- Can be set up to run on every check in

Security Code Reviews

- Performed by a person
- Validates code against standards

Secure Coding Standards in Niagara

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Secure Implementation Best Practices

Security Coding Standards

- Defines best practices
- Helpful reference to developers

Static Code Analysis

- Can detect security issues
- Can be set up to run on every check in

Security Code Reviews

- Performed by a person
- Validates code against standards

Static Code Analysis: Picking a Tool

- Programming language support
- Build tool integration
- Supported issues and findings

Secure Implementation Best Practices

Security Coding Standards

- Defines best practices
- Helpful reference to developers

Static Code Analysis

- Can detect security issues
- Can be set up to run on every check in

Security Code Reviews

- Performed by a person
- Validates code against standards

Security Code Reviews

- Performed by a security person
- Catches things automated tools don’t
- Ensures security coding standards are met
- Should be performed on all code

Secure Implementation Best Practices

Security Coding Standards

- Defines best practices
- Helpful reference to developers

Static Code Analysis

- Can detect security issues
- Can be set up to run on every check in

Security Code Reviews

- Performed by a person
- Validates code against standards

Example: Access

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Access Standards

Use access modifiers:

- private, package-private, protected, and public

Restrict access by default

Open access (with caution) if needed

Example: Access

```
public static void fetchUserFullName(String user) throws ConfigException
{
  try
  {
    String apiKey = AccessController.doPrivileged((PrivilegedExceptionAction<String>)() ->
      new String(SecurityUtil.toHexChars(keyRing.getKey("summit2024.apiKey"))));
    fetchDataFromRemoteServer("userFullName", user, apiKey);
  }
  catch (PrivilegedActionException e)
  {
    throw new ConfigException(e.getException(), "Could not retrieve api key");
  }
}
```

Uses the API key in a controlled way

Example: Access

```
public static void fetchUserFullName(String user) throws ConfigException
{
  try
  {
    String apiKey = AccessController.doPrivileged((PrivilegedExceptionAction<String>)() ->
      new String(SecurityUtil.toHexChars(keyRing.getKey("summit2024.apiKey"))));
    fetchDataFromRemoteServer("userFullName", user, apiKey);
  }
  catch (PrivilegedActionException e)
  {
    throw new ConfigException(e.getException(), "Could not retrieve api key");
  }
}

public static void fetchUserCreationDate(String user) throws ConfigException
{
  try
  {
    String apiKey = AccessController.doPrivileged((PrivilegedExceptionAction<String>)() ->
      new String(SecurityUtil.toHexChars(keyRing.getKey("summit2024.apiKey"))));
    fetchDataFromRemoteServer("userCreationDate", user, apiKey);
  }
  catch (PrivilegedActionException e)
  {
    throw new ConfigException(e.getException(), "Could not retrieve api key");
  }
}
```

The API key is still used in a controlled way

Example: Access

```
public static void fetchUserFullName_updated(String user) throws ConfigException
{
  fetchDataFromRemoteServer("userFullName", user, getApiKey());
}

public static void fetchUserCreationDate_updated(String user) throws ConfigException
{
  fetchDataFromRemoteServer("userCreationDate", user, getApiKey());
}

public class Utils
{
  static String getApiKey() throws ConfigException
  {
    try
    {
      return AccessController.doPrivileged((PrivilegedExceptionAction<String>)() ->
        new String(SecurityUtil.toHexChars(keyRing.getKey("summit2024.apiKey"))));
    }
    catch (PrivilegedActionException e)
    {
      throw new ConfigException(e.getException(), "Could not retrieve api key");
    }
  }
}
```

The API key is still protected

Example: Access

```
public static void fetchUserFullName_updated(String user) throws ConfigException
{
  fetchDataFromRemoteServer("userFullName", user, getApiKey());
}

public static void fetchUserCreationDate_updated(String user) throws ConfigException
{
  fetchDataFromRemoteServer("userCreationDate", user, getApiKey());
}

public class Utils
{
  public static String getApiKey() throws ConfigException
  {
    try
    {
      return AccessController.doPrivileged((PrivilegedExceptionAction<String>)() ->
        new String(SecurityUtil.toHexChars(keyRing.getKey("summit2024.apiKey"))));
    }
    catch (PrivilegedActionException e)
    {
      throw new ConfigException(e.getException(), "Could not retrieve api key");
    }
  }
}
```

The API key is **exposed**

Access in Niagara

- Slots are public
- Flags are just UI hints
- Use BPassword for sensitive data
- Use IPropertyValidator to restrict user changes

Example: Error Handling

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Error Handling: Client-Server

Helpful to users vs. helpful to attackers

~~Username not found.~~
~~Incorrect password.~~
Invalid username/password combination. ✓

Error Handling: Client-Server

Consider ALL responses

Account exists

Account does not exist

Discover valid accounts to attack

Error Handling: Client-Server

Consider response timing

"Pass" vs. "Password10"

"Passwxxxxx" vs "Password10"

Use javax.baja.nre.util.SecurityUtil.equals()
or javax.baja.security.BPassword.validate()

Discover the password one character at a time

Error Handling: Client-Server

Applies after authentication

Resource exists	Resource does not exist
403 Forbidden	404 Not Found

Discover resources to attack

Send the same response to the client in both cases.

Exception Handling

- Do not expose sensitive information
- Close resources – use try-with-resources
- Restore state in multi-step operations
- Fail secure

Exception Handling: Fail Secure

```
Role getUserRole(String username)
{
  Role role = Role.ADMIN;
  try
  {
    role = findRoleForUser(username);
  }
  catch (Exception e)
  {
    // This should never happen
    log.warning("Could not find role for user: " + username);
  }
  return role;
}
```

If an exception occurs, role remains Role.ADMIN

Exception Handling: Fail Secure

```
Role getUserRole(String username)
{
  Role role = Role.NONE;
  try
  {
    role = findRoleForUser(username);
  }
  catch (Exception e)
  {
    // This should never happen
    log.warning("Could not find role for user: " + username);
  }
  return role;
}
```

If an exception occurs, role remains Role.NONE

If code is added that sets 'role', it still could end up in an incorrect state

Exception Handling: Fail Secure

```
Role getUserRole(String username)
{
  try
  {
    return findRoleForUser(username);
  }
  catch (Exception e)
  {
    log.warning("Could not find role for user: " + username);
    return Role.NONE;
  }
}
```

Guaranteed to
return Role.NONE

Example: Input Validation

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Input Validation

- Input is in expected format
- Input is within expected parameters
- Input is checked before using
- Avoids unintended behavior

Example: Input Validation

```
ResultSet getRecordsForDates(Connection connection, String username, UserInput userInput)
  throws SQLException
{
  String startDate = userInput.getStartDate();
  String endDate = userInput.getEndDate();

  String query = "SELECT * FROM records WHERE recordDate BETWEEN '" + startDate + "' AND " +
    endDate + "' AND username='" + username + "'";
  try (Statement statement = connection.createStatement())
  {
    return statement.executeQuery(query);
  }
}
```

An attacker could set endDate to:

`2024-04-15' AND username='anotherUser' --`

And the query becomes:

```
SELECT * FROM records WHERE recordDate BETWEEN '2024-01-01'
AND '2024-04-15' AND username='anotherUser' -- nothing after this matters
```

Example: Input Validation

File Access

- new File(baseDirectory, "../../escape/to/new/directory/ohNo"

Running an OS command

- Runtime.getRuntime().exec(unintendedCommandInUserInput);

Building URLs

- https://example.com/path?userInputParam=value1&badParam=value2

Example: Input Validation

```
ResultSet getRecordsForDates(Connection connection, String username, UserInput userInput)
  throws SQLException
{
  String startDate = userInput.getStartDate();
  String endDate = userInput.getEndDate();

  String query = "SELECT * FROM records WHERE recordDate BETWEEN ? AND ? AND username=?";
  try(PreparedStatement statement = connection.prepareStatement(query))
  {
    statement.setString(1, startDate);
    statement.setString(2, endDate);
    statement.setString(3, username);

    return statement.executeQuery();
  }
}
```

Each parameter is treated as an individual entity and cannot break out

How to Validate Input: Allow Lists

- Lists of allowed inputs
- Can look at input as a whole
- Can look at individual characters

Allow List Example

```
static boolean isDayValid(String userInput)
{
  return VALID_DAYS.contains(userInput);
}

private static final List<String> VALID_DAYS = Collections.unmodifiableList(
  Arrays.asList("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"));
```

Great for small input sets!

How to Validate Input: Regular Expressions

- Pattern that input must match
- Well-defined structure
- Lots of different combinations

Regular Expression Example

```
static boolean isValidSocialSecurityNumber(String userInput)
{
  return ssnPattern.matcher(userInput).matches();
}

private static final Pattern ssnPattern =
  Pattern.compile("^(?!666|000|9\\d{2})\\d{3}-(?!00)\\d{2}-(?!0{4})\\d{4}$");
```

Consider getting your
regex from a trust source
e.g. OWASP

https://owasp.org/www-community/OWASP_Validation_Regex_Repository

More Ways to Validate Input

Check input length

- Very quick to check
- Can be used before another more expensive check

Semantic validation

- Checks meaning vs. format
- Even well-formed data can cause problems

Example: Cryptographic Functions

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Example: Cryptographic Functions

- Do not write your own crypto
- Use modern, industry-standard algorithms
- Properly manage secret keys
- Use SecureRandom

Using Cryptographic Functions

```
void useEncryption(byte[] bytesToEncrypt)
{
  byte[] initializationVector = generateInitializationVector();
  byte[] encryptedBytes = encrypt(bytesToEncrypt, initializationVector);
  // do something with encryptedBytes
}
```

```
private byte[] generateInitializationVector()
{
  SecureRandom secureRandom = new SecureRandom();
  byte[] iv = new byte[12];
  secureRandom.nextBytes(iv);
  return iv;
}
```

Uses SecureRandom

Using Cryptographic Functions

```
void useEncryption(byte[] bytesToEncrypt)
{
  byte[] initializationVector = generateInitializationVector();
  byte[] encryptedBytes = encrypt(bytesToEncrypt, initializationVector);
  // do something with encryptedBytes
}
```

```
private byte[] encrypt(byte[] bytesToEncrypt, byte[] iv)
{
  try (SecretBytes encryptionKey = getEncryptionKey())
  {
    Cipher aesCipher = Cipher.getInstance("AES/GCM/NoPadding");
    AlgorithmParameterSpec parameterSpec = new GCMParameterSpec(128, iv);
    SecretKey secretKey = new SecretKeySpec(encryptionKey.get(), "AES");
    aesCipher.init(Cipher.ENCRYPT_MODE, secretKey, parameterSpec);
    return aesCipher.doFinal(bytesToEncrypt);
  }
  catch (Exception e)
  {
    throw new SecurityException("Could not encrypt bytes: " + e.getMessage());
  }
}
```

Uses integrated crypto provider

Uses industry standard algorithm

Using Cryptographic Functions

```
private SecretBytes getEncryptionKey()
  throws Exception
{
  BPassword secretKeyPassword;
  if (getSecretKey().isNull())
  {
    KeyGenerator keyGen = KeyGenerator.getInstance("AES");
    SecretKey secretKey = keyGen.generateKey();
    secretKeyPassword = BPassword.make(SecurityUtil.toHexChars(secretKey.getEncoded()));
    setSecretKey(secretKeyPassword);
  }
  else
  {
    secretKeyPassword = getSecretKey();
  }

  return AccessController.doPrivileged((PrivilegedAction<SecretBytes>)() ->
    secretKeyPassword.getSecretChars().asSecretBytes());
}
```

Securely stores secret key in a BPassword slot

Example: Cryptographic Functions

Access	Security Manager	Error Handling	Auditing	Logging	Injection
Authentication	Authorization	Input Validation	Output Sanitization	Return Values	Cookies
Protecting Sensitive Data	Passwords	Third Party Libraries	Secure by Default	Cryptographic Functions	Niagara-specific considerations

Resources

- OWASP (Open Web Application Security Project)
  - https://owasp.org/
- SANS Institute
  - https://www.sans.org/
- CERT Secure Coding Standards
  - https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
- Oracle's Java: The Secure Coding Guidelines
  - https://www.oracle.com/java/technologies/javase/seccodeguide.html
- List of Static Code Analysis Tools
  - https://owasp.org/www-community/Source_Code_Analysis_Tools

Any questions?

TRIDIUM