
NS2024
POWER OF PARTNERSHIP

WEB SECURITY AND NIAGARA UI BEST PRACTICES

Vikram Nagulan
Software Engineer @ Tridium

NIAGARA FRAMEWORK SECURITY

- Who are we defending against?
  -  actors external to the system
  -  users who already have access
- What are we defending?
  -  Niagara station assets
    - components, files, histories
  -  Safety of the actual users of the system.
  - Other sensitive data

PART 1 - USERS OF THE SYSTEM

How can we secure the system from authorized users with limited access?

NIAGARA SECURITY MODEL OVERVIEW

- Make ***Categories*** to group station assets
- Create ***Roles*** and map permissions to Categories
- Define ***Users*** and assign Roles
- Authenticate and allow only Authorized Users access
- Audit trail for user actions

WEB SECURITY

How to code to make this security model work?

- always pass the ***Context***
- resolve ORDs with Context
- invoke Actions with Context
- Servlets and RPCs have Context; use it

CODE SNIPPETS

How to get the context from a HttpServlet?

```
public class MyHttpServlet extends javax.servlet.http.HttpServlet
{
  @Override
  protected void doGet(HttpServletRequest req,
                       HttpServletResponse resp)
    throws ServletException, IOException
  {
    Context cx = (Context) req.getAttribute("niagara.context");
    resp.getWriter().write("The user context is: " +
      Encode.forHtml(String.valueOf(cx)));
  }
}
```

Results

CODE SNIPPETS (CONT.)

How to get the context from a BWebServlet?

```
@NiagaraType
public class BMyWebServlet extends BWebServlet
{
  @Override
  public void doGet(WebOp op) throws Exception
  {
    Context cx = op;
    HttpServletResponse resp = op.getResponse();
    resp.getWriter().write("The user context is: " +
      Encode.forHtml(String.valueOf(cx)));
  }
}
```

Results

CODE SNIPPETS (CONT.)

How to use the Context to check permissions?

```
// Already have Context
// Now try to resolve an ORD with it
OrdTarget target = ord.resolve(base, cx);
// Check read permissions
if (!target.canRead())
{
  throw new PermissionException("Not enough privileges to the target");
}
```

CODE SNIPPETS (CONT.)

Get more specific on permission checks

```
// Already have Context
// I want to check if a user specifically has
// operator write permissions on the service 'myService'

// Check operator write permissions
BPermissions permissions = cx.getUser().getPermissionsFor(myService);
if (!permissions.has(BPermissions.operatorWrite))
{
  throw new PermissionException(
    "User does not have write permissions to 'MyService'.");
}
```

RPC

```
@NiagaraRpc(
  permissions = "r",
  transports = {
    @Transport(type = TransportType.box),
    @Transport(type = TransportType.web)
  }
)
public void doSomething(String ordString, Context cx)
{
  BOrd ord = BOrd.make(ordString);
  OrdTarget protectedTarget = ord.resolve(base, cx);
  // User may have the 'r' permission but they may not have
  // permissions to the 'protectedTarget' resolved here.
  if (!protectedTarget.canRead())
  {
    // throw a permissions exception
  }
}
```

ENFORCE CONTEXT AT COMPILE TIME

```
public void doSomething(BOrd ord, Context cx)
{
  OrdTarget protectedTarget = ord.resolve(null, cx);
  doSomethingImportantWith(protectedTarget);
}
// doSomething(ordToMyProtectedTarget, null)
```

A FEW THINGS TO NOTE

It can be very useful to send appropriate error codes

```
// Good to also send a locale friendly error message
final String errorMsg403 = lex.getHtmlSafeText('forbidden.error', cx);
final String errorMsg401 = lex.getHtmlSafeText('unauthorized.error', cx);

// If the user has insufficient privileges - HTTP Error Code 403
resp.sendError(HttpServletResponse.SC_FORBIDDEN, errorMsg403);

// If the user is not authorized - HTTP Error Code 401
resp.sendError(HttpServletResponse.SC_UNAUTHORIZED, errorMsg401);

// ... and so on
```

A FEW THINGS TO NOTE (CONT.)

```
// If it is a SecurityException or SecurityException
// in its chain of causes
// Check the WebService's boolean slot "showStackTrace"
if (isSecurityException(e))
{
  if (webService.getShowStackTrace()) { /* log the error */ }

  resp.sendError(HttpServletResponse.SC_FORBIDDEN,
    lex.getHtmlSafeText('forbidden.error', cx));
}
else
{
  if (webService.getShowStackTrace())
  {
    exception.printStackTrace(resp.getWriter());
  }
}
```

PART 2 - EXTERNAL UNAUTHORIZED USERS

How can we secure the system from unauthorized users?

WHO ARE THESE BAD ACTORS AND HOW CAN OUR SYSTEM BE COMPROMISED?

- Hackers and scammers
- Phishing attacks
- Injection

- Request forgers

ⓘ https://owasp.org/www-project-top-ten/

CROSS-SITE REQUEST FORGERY (CSRF)

- Prevented by validating every client request against a server generated "token"
    - RPCs, Servlet requests, ajax requests or any other form of requests from a client to the server
    - This token is typically set as a header with every request
    - More details here module://docDeveloper/doc/security/csrfProtection.html

ⓘ https://owasp.org/www-community/attacks/csrf

CSRF BY EXAMPLE

```
// Forced to execute via Social Engineering
function doOp() {
  $.ajax({
    url: 'https://mystation.com/myServlet',
    data: { op: 'deleteUser', user: 'userX' },
    contentType: 'application/json',
    method: 'POST'
  });
}
```

CsrfProtectedFilter

```
<!-- CSRF Protect NiagaraRpcServlet servlet -->
<filter>
  <filter-name>csrfProtectedNiagaraRpcFilter</filter-name>
  <filter-class>javax.baja.web.filters.CsrfProtectedFilter</filter-class>
  <init-param>
    <param-name>httpMethod</param-name>
    <param-value>GET,POST</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>myServletCSRFProtectionFilter</filter-name>
  <url-pattern>/myServlet</url-pattern>
</filter-mapping>
```

CSRF BY EXAMPLE (CORRECTED)

```
function doOp() {
  const { CSRF_TOKEN_HEADER_KEY, getCsrfToken } = csrfUtil;

  $.ajax({
    url: 'https://mystation.com/myServlet',
    headers: { [CSRF_TOKEN_HEADER_KEY]: getCsrfToken() },
    data: { op: 'deleteUser', user: 'userX' },
    contentType: 'application/json',
    method: 'POST'
  });
}
```

CROSS SITE SCRIPTING (XSS)

- Reflected, Stored and DOM based
- Can be injected via
  - HTML controls
  - Lexicons
- Can be prevented
  - Always escape user generated data
  - Validate all inputs on the server side
- CSP and XSS Protection headers

ⓘ https://owasp.org/www-community/attacks/xss/

XSS BY EXAMPLE

```
dialogs.showOk({
  content: 'My Component is: ' + comp.getDisplayName() // no escaping
});

// corrected
dialogs.showOk({
  content: 'My Component is: ' + _.escape(comp.getDisplayName())
});
```

```
  // Use "Safe" APIs or helpers wherever applicable
  _.escape() // Via popular utilities (like underscore)
  // Only set the text content of a HTML
  span.textContent = 'User generated content'
  lex.getSafe() // Javascript safe method to get a lexicon value
  lex.getHtmlSafe()/lex.getHtmlSafeText() // Java safe method to get a
lexicon value
  //... and so on
```

BHttpHeaderProviders

- Introduced in Niagara 4.10.
- Lives on the WebService.
- Customize existing Response Headers or implement your own.
- Enabled by default but can be disabled independently.
- Some Header providers come built in which just needs configuration.
    - CSP, XSS Protection are a couple of example built in Header providers.

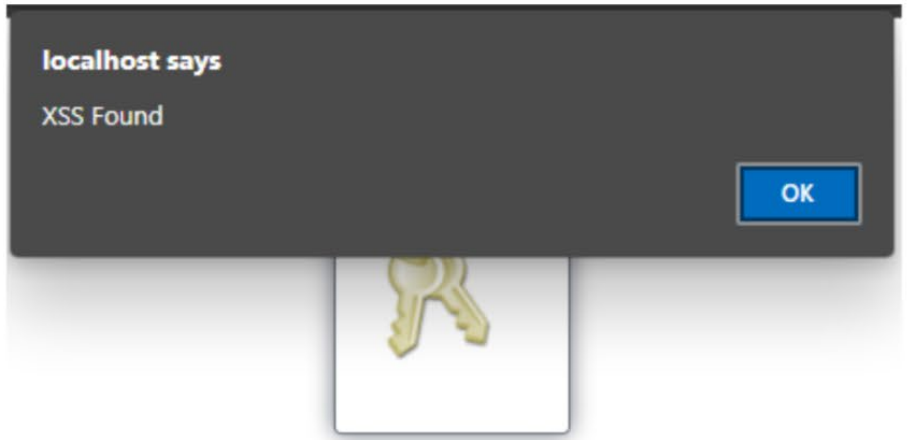
IMPLEMENT BIHttpHeaderProvider

```
@NiagaraType
public class BMyHttpHeaderProvider extends BComponent
  implements BIHttpHeaderProvider
{
  @Override
  public void applyHeaders(HttpServletRequest req,
                           HttpServletResponse resp)
  {
    final String headerValue = getHeaderAfterSomeProcessing();
    resp.addHeader("myHeader", headerValue);
    // Or for example, if you just want to set a header and not append it
    // final String mySpecificOrigin = getMySpecificOrigin();
    // resp.setHeader("Access-Control-Allow-Origin", mySpecificOrigin);
  }
}
```

BGenericHttpHeaderProvider

DIAGNOSTIC LEXICON

```
  #System.properties
  niagara.lexicon.diagnostics=true
  # niagara.lexicon.diagnostics.prefix=lexiconModule-lexiconLang-
lexiconKey=
  niagara.lexicon.diagnostics.suffix=&lt;img src="/a"
onerror="window.alert('XSS Found');">
```



~~lex.get("login.username")~~

Username:<img src="/a" onerror="window.alert('XSS Found');"> admin

Change User<img src="/a" onerror="window.alert('XSS Found');">

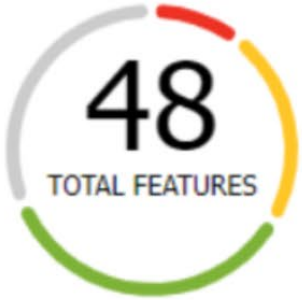
Password:<img src="/a" onerror="window.alert('XSS Found');">

Login<img src="/a" onerror="window.alert('XSS Found');">

lex.getHtmlSafe("login.username")

OTHER SECURITY RELATED TOPICS

- Security Dashboard
- Auto-logoff



Alert **5** Hide | Warning **11** Hide | OK **17** Hide | Info **15** Hide

Auto Logoff feature is disabled for these User accounts:

Http Enabled is set to true, allowing non-secure connections.

Https Min Protocol is TLSv1.0+.

The following schemes are configured to a weak password strength:...

Host header validation is ENABLED.
The Host header is used to generate certain redirects and URLs. Turning on validation and setting a list of allowed values for the Host header ensures that these URLs are being constructed with known, secure values.

All HTTP security headers are enabled.
For best security, all security-related HTTP headers should be enabled.

The X-Content-Type-Options HTTP header is set to: nosniff.
The X-Content-Type-Options HTTP header governs how the browser attempts to auto-detect the content type of downloaded files. The recommended value is "nosniff".

The X-Frame-Options HTTP header is set to: Sameorigin.
The X-Frame-Options HTTP header governs whether a site is allowed to embed your Niagara station's web interface into an iframe. "DENY" is the most secure and prevents any iframe embedding. "SAMEORIGIN" allows your web interface to embed its own internal pages. "ANY" may cause a clickjacking vulnerability and is not recommended. If you need to allow an external site to embed your Niagara web interface, configure a "frame-ancestors" directive under Content-Security-Policy.

The X-XSS-Protection header is set to: 1; mode=block.
The X-XSS-Protection header instructs the browser to stop loading a page if it detects a reflected XSS attack. This header is mostly superseded by Content-Security-Policy, but should still be enabled to improve security in browsers that do not support Content-Security-Policy. The recommended value is "1; mode=block".

HELPFUL RESOURCES

- Workbench -> Help Documents -> Developer Guide -> Security
  - The "Security" section has helpful links to all things Security in Niagara
- module://docDeveloper/doc/niagaraRpc.html
- https://www.owasp.org

CULTURE OF VIGILANCE

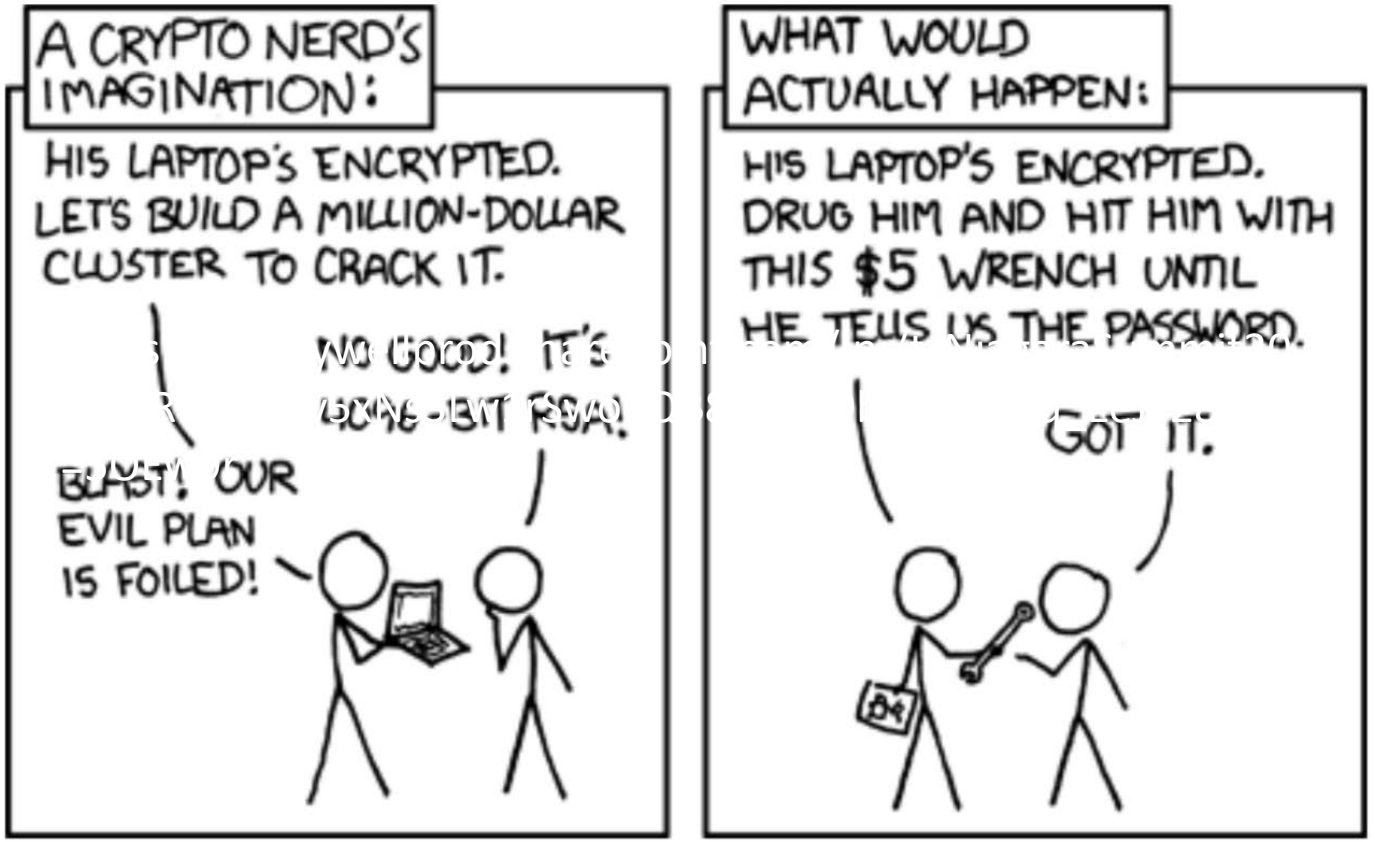


Image source: https://xkcd.com/538/

TRIDIUM