
NS
18
NIAGARA
SUMMIT
CONNECTING
THE WORLD

Niagara 4 Security

Best practices for developing secure applications for Niagara 4

Patrick Sager, Melanie Coggan
Software Engineers

How can we prevent this?

```
Runtime r = Runtime.getRuntime();

r.exec(new String[] {"net", "user", "evil",
       "Password10", "/add",});

r.exec(new String[] {"net", "localgroup",
       "administrators", "evil", "/add"});
```

Security Manager

Restricts privileges of potentially malicious code

AccessControlException

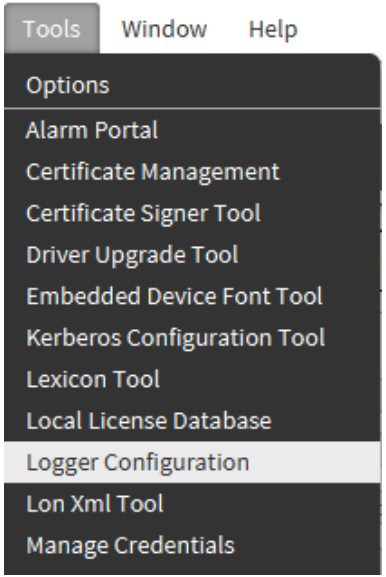
```
java.security.AccessControlException: access denied
("java.net.SocketPermission" "google.com:80" "connect,resolve")

  ...

  at sun.net.www.protocol.http.HttpURLConnection.getInputStream...
  at com.patrick.myModule.MyClass.makeConnection(MyClass.java:50)

  ...
```

Niagara Policy Logging

Remote Station | logSetup

Log Configuration											
Log	**Level**	**OFF**	**SEVERE**	**WARNING**	**INFO**	**CONFIG**	**FINE**	**FINER**	**FINEST**	**ALL**	**DEFAULT**
DEFAULT	INFO	[ ]	[ ]	[ ]	[X]	[ ]	[ ]	[ ]	[ ]	[ ]	Not Applicable
alarm	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
alarm.database	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
power	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
search	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
security.keyMaterial	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
security.keyRing	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
security.niagaraPolicy	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[X]	[ ]	[ ]	[ ]	[ ]
security.niagaraPolicy.permissions	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
subSpaceFile	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]
sun.awt	SEVERE	[ ]	[X]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
sun.awt.AppContext	DEFAULT	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[X]

Niagara Policy Logging



Niagara Policy Logging

```
FINE [14:39:15 17-Mar-17 EDT][security.niagaraPolicy] Access denied

  Permission that failed: ("java.net.SocketPermission"
"google.com:80" "connect,resolve")

  ProtectionDomain that failed: file:/c:/niagara/niagara-
4.3.58.6/modules/myModule-rt.jar

  Stack trace causing the failure:
  ...
```

Niagara Permission Groups

AUTHENTICATION
BACKUPS
DIAGNOSTICS
GET_ENVIRONMENT_VARIABLES
LOAD_LIBRARIES
LOGGING
MANAGE_EXECUTION
MODIFY_IO_STREAMS
UNAUTHENTICATED_ACCESS
NETWORK_COMMUNICATION
RUNTIME_EXECUTION
SET_SYSTEM_TIME
SHUTDOWN_HOOKS
SYSTEM_PROPERTIES
UI
RENAME_AND_RESTART
REFLECTION *
HSM_SIGNING *

* Requires signed module

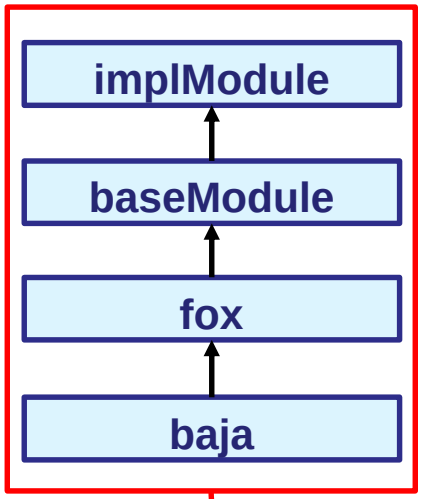
module-permissions.xml

```
<permissions>
  <niagara-permission-groups type="station">
    <req-permission>
      <name>NETWORK_COMMUNICATION</name>
      <purposeKey>Communicate with websites.</purposeKey>
      <parameters>
        <parameter name="hosts" value="*"/>
        <parameter name="ports" value="80,443"/>
      </parameters>
    </req-permission>
  </niagara-permission-groups>
</permissions>
```

Privileged Actions

```
InputStream in = AccessController.doPrivileged(
  (PrivilegedAction<InputStream>)() ->
{
  URL url = new URL("http://google.com");
  URLConnection con = url.openConnection();
  return con.getInputStream();
});
```

Privileged Actions



Custom implementation that triggers a permission check

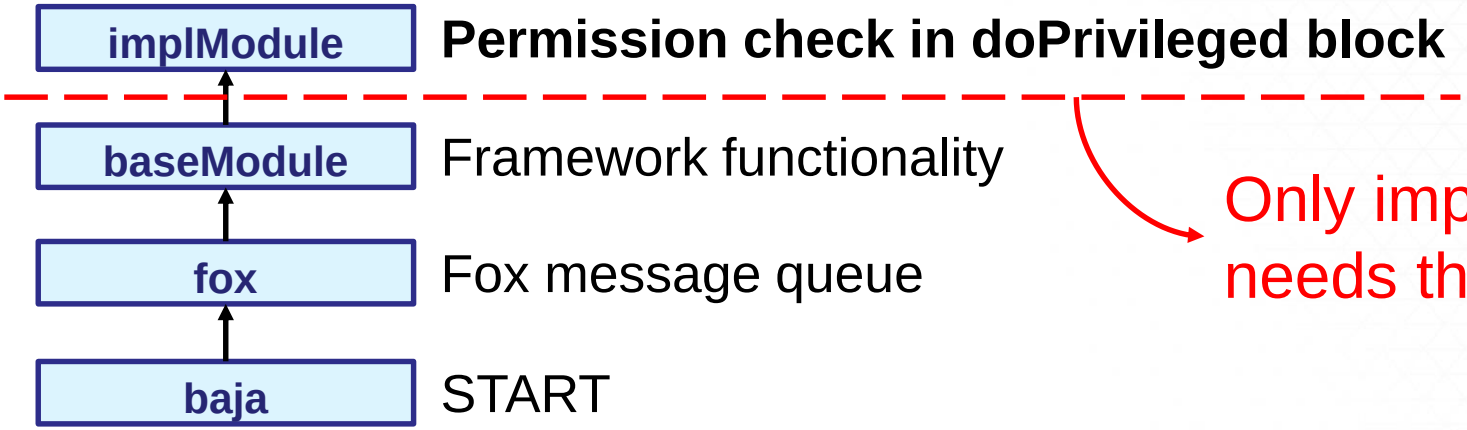
Framework functionality

Fox message queue

START

Every module on the stack needs the permission!

Privileged Actions



File Operations

Security manager restricts file operations to:

Station Home

Sys.getStationHome()

Niagara User Home

Sys.getNiagaraSharedUserHome()

Keeping Secrets

Protecting sensitive information

Secure Network Communication

TLS

- Confidentiality
  - Encryption prevents eavesdropping
- Trust
  - Certificates identify the server
- Integrity
  - Integrity checks prevent message manipulation

URLConnection

```
URL url = new URL("https://google.com/?q=kittens");
URLConnection con = url.openConnection();

con.getInputStream();
```

CertManagerFactory

```
ICryptoManager manager = CertManagerFactory.getInstance();
SocketFactory factory =
    manager.getClientSocketFactory(BSslTlsEnum.default);

Socket socket = factory.createSocket("google.com", 443);
```

Credential Management

- Reversible password encoding
  - Use when you need to authenticate to another service.
- Irreversible password encoding
  - Use when someone is authenticating to your service.

Reversible Password Encoding

```
BPassword password = BPassword.make("Password",
      BAes256PasswordEncoder.ENCODING_TYPE);

String value = AccessController.doPrivileged(
       (PrivilegedAction<String>)password::getValue);

connect(value);

...

String pass = "Password";
boolean match = password.validate(pass);
```

Irreversible Password Encoding

```
BPassword password = BPassword.make("Password",
      BPbkdf2HmacSha256PasswordEncoder.ENCODING_TYPE);

String pass = "Password";
boolean match = password.validate(pass);
```

Common Security Vulnerabilities

Cross site scripting, cross site request forgery,
and missing function level access control

https://www.owasp.org/index.php/Top_10-2017_Top_10

Cross Site Scripting

Allows attackers to execute scripts in a victim's browser

Leads to

- Session hijacking
- Credential theft
- Data theft
- Unauthorized operations

Cross Site Scripting

Types of cross site scripting

- Stored XSS
  - User input is stored on server, then written to HTML
- Reflected XSS
  - Part of request is written to HTML

Cross Site Scripting

```
HtmlWriter out = op.getHtmlWriter();
out.write("<h3>Player X: " + xUser.getFullName() + "</h3>");
out.write("<h3>Player O: " + oUser.getFullName() + "</h3>");
```

Cross Site Scripting

```
melanie</h3>
<script>
  window.onload = function() {
    document.getElementsByName("a2")[0].click()
  }
</script>
<h3>
```

XSS Prevention

Escape untrusted data

- HtmlWriter.safe()
- jQuery.text()
- org.owasp.encoder.Encode
- <script> → <script>

XSS Prevention

```
HtmlWriter out = op.getHtmlWriter();
out.write("<h3>Player X: ");
out.safe(xUser.getFullName());
out.write("</h3>");
out.write("<h3>Player O: ");
out.safe(oUser.getFullName());
out.write("</h3>");
```

Cross Site Request Forgery

Allows attackers to trick victim into submitting requests to the target server.

Cross Site Request Forgery

```
String slot = request.getParameter("selection");
BTicTacToeEnum value = game.getValue(currentUser);
game.set(slot, value);
```

Cross Site Request Forgery

```
https://localhost/ord/station:%7Cslot:/ticTacToeGame
    %7Cview:ticTacToe:HxTicTacToeView?selection=a2
```

CSRF Prevention

CSRF token

- Java: SessionUtil.getCurrentNiagaraSuperSession().getCsrfToken()
- JavaScript: csrfUtil.getCsrfToken()
- Hx: hx.getCsrfToken()
- Velocity/Mobile: $csrfToken
- web.xml: CsrfProtectedFilter
- Java: CsrfUtil.verifyCsrfToken()

CSRF Prevention

```
try
{
  CsrfUtil.verifyCsrfToken(request);
  String slot = request.getParameter("selection");
  BTicTacToeEnum value = game.getValue(currentUser);
  game.set(slot, value);
}
catch (CsrfException e)
{
  response.sendError(403, e.getMessage());
}
```

Missing Function Level Access Control

Allows attackers to perform unauthorized operations by bypassing UI level access control.

Missing Function Level Access Control

```
out.write("<button name='selection' value='" + slot + "'");
if (!myTurn || !value.isEmpty())
{
  out.write(" disabled");
}
out.write(">" + value + "</button>");
```

Missing Function Level Access Control Prevention

Always perform server-side access control checks

Missing Function Level Access Control Prevention

```
public BObject save(HxOp op)
{
  ...
  if (!myTurn || !value.isEmpty())
  {
    op.sendError(403, new Exception("unauthorized"));
  }
  ...
```

What we learned

- How to work with the Security Manager
- How to protect sensitive data
- How to prevent common security vulnerabilities