TRIDIUM

NS2022

ACCELERATING INNOVATION

TRIDIUM

Security Dashboard API

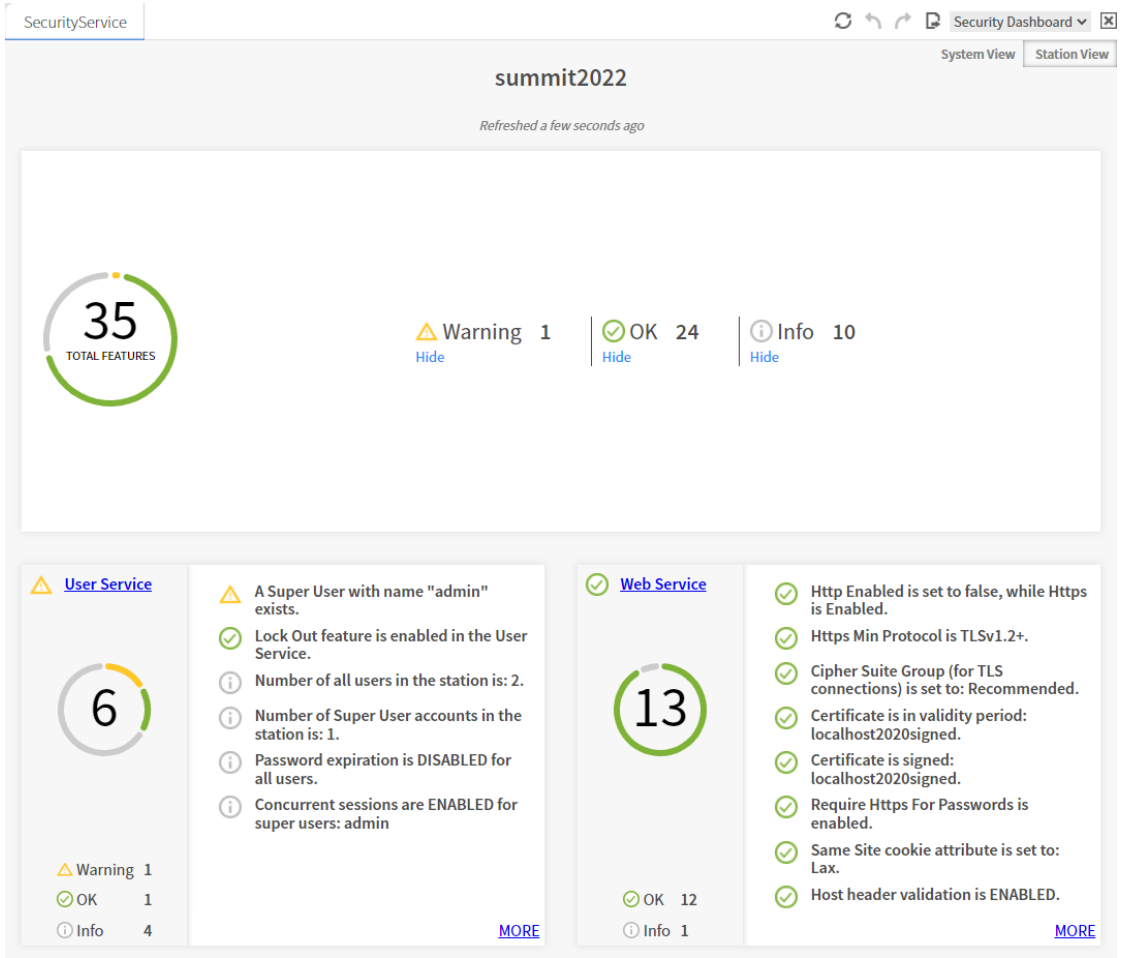
Melanie Coggan

Software Engineer – Tridium

What Is the Security Dashboard?

TRIDIUM

Bird's Eye View of Security

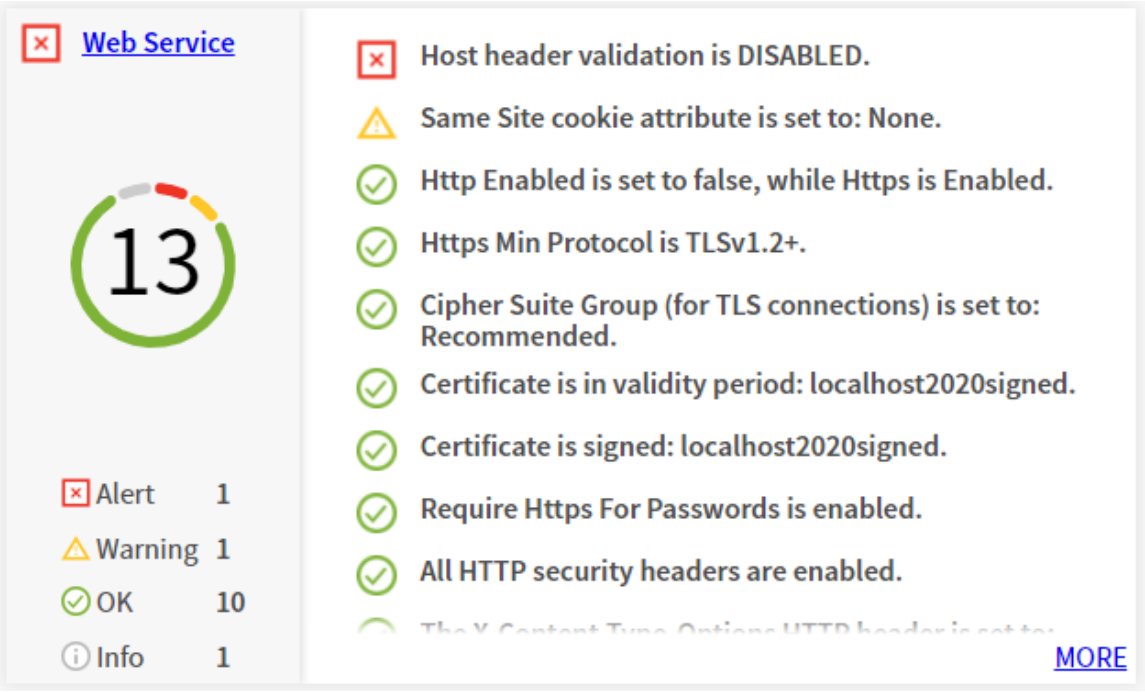


Security settings all in one spot

Easy to read color coding

Direct links to components that need to be fixed

Security Dashboard Card

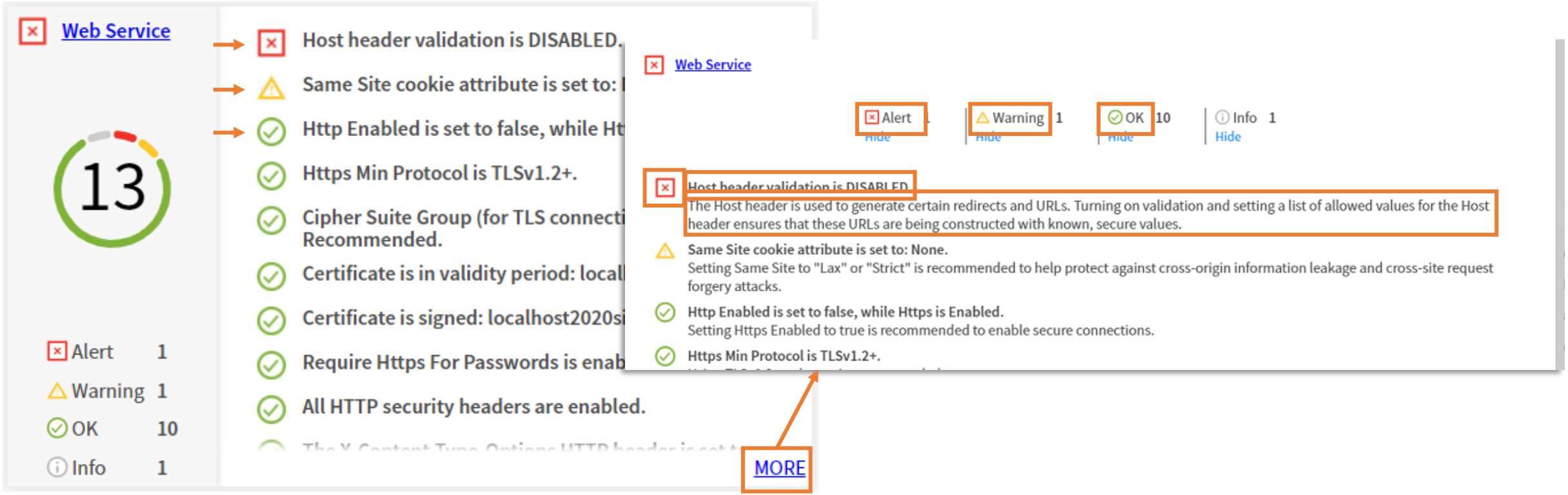


Contains many related items

Links back to a single component

Security Dashboard Item

Consists of a status, summary, and description

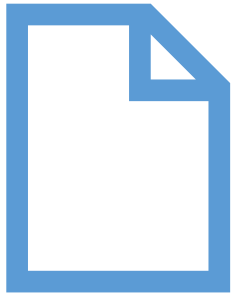


What's the Benefit?

Makes security easier!

Why We're Here

Public API allows custom modules to:



Add a new card
to the dashboard

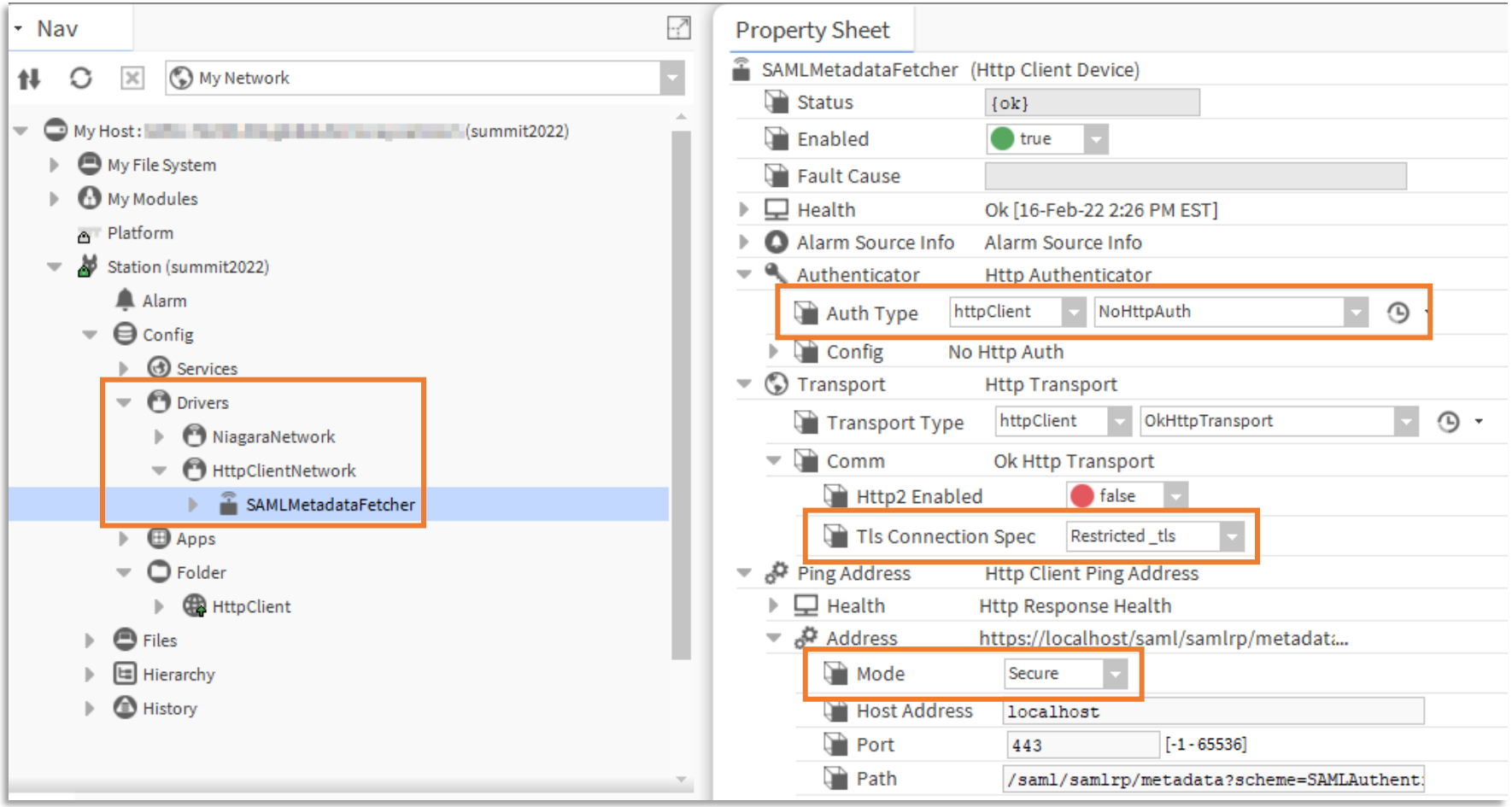
Add items to an
existing card

Sample Module

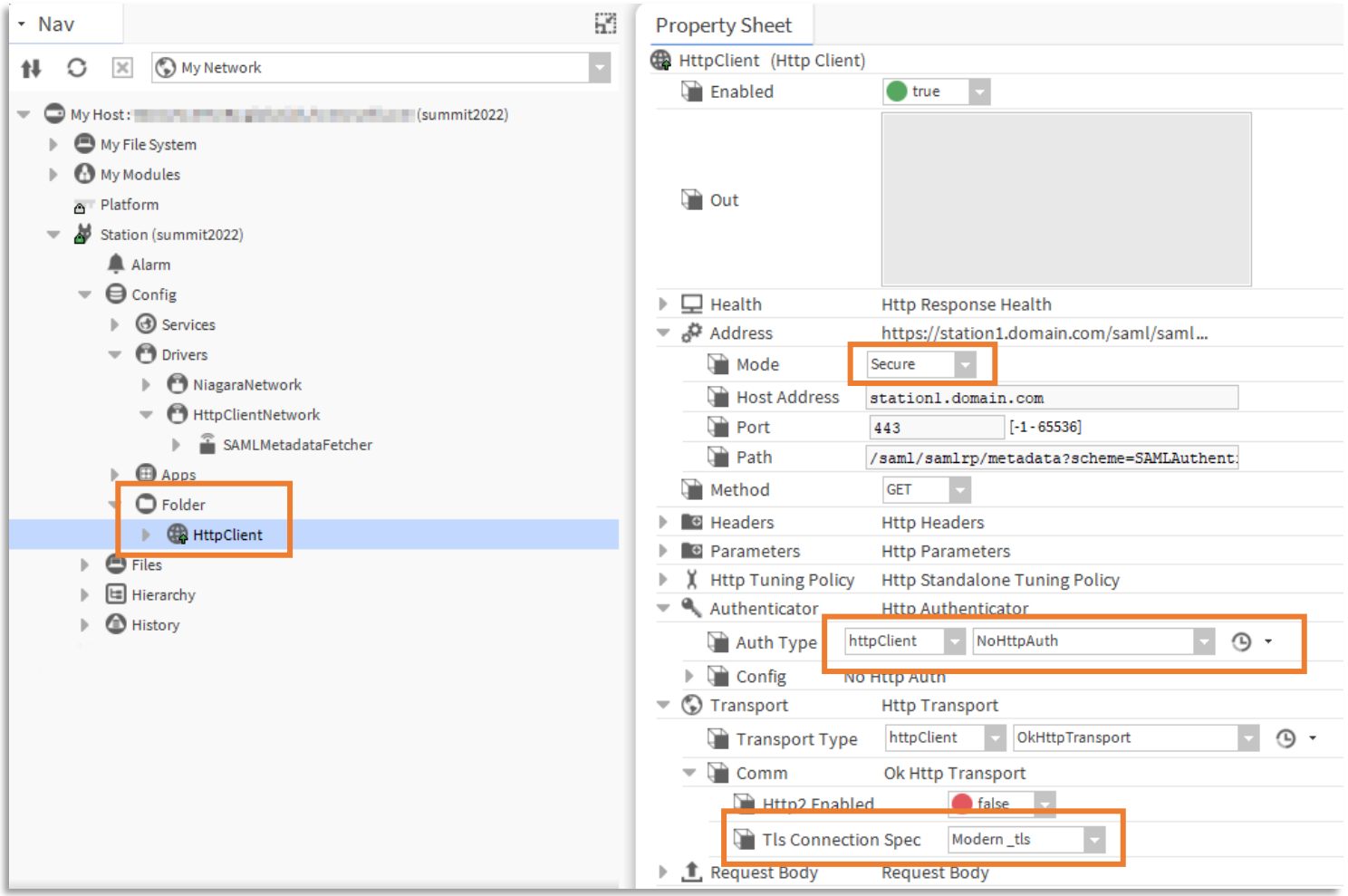
HTTP Client

TRIDIUM

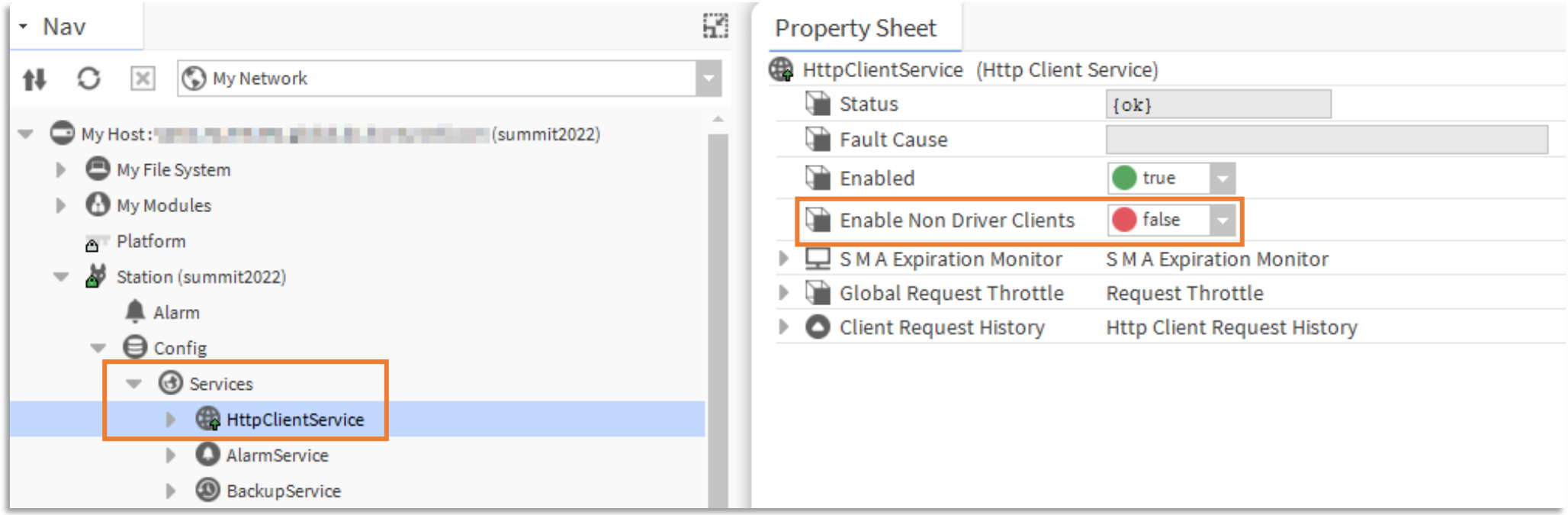
HttpClientDevice



HttpClient


Nav
My Network
My Host: (summit2022)
My File System
My Modules
Platform
Station (summit2022)
Alarm
Config
Services
Drivers
NiagaraNetwork
HttpClientNetwork
SAMLMetadataFetcher
Apps
Folder
HttpClient
Files
Hierarchy
History
Property Sheet
HttpClient (Http Client)
Enabled
true
Out
Health
Http Response Health
Address
https://station1.domain.com/saml/saml...
Mode
Secure
Host Address
station1.domain.com
Port
443
[-1 - 65536]
Path
/saml/samlrp/metadata?scheme=SAMLAuthent:
Method
GET
Headers
Http Headers
Parameters
Http Parameters
Http Tuning Policy
Http Standalone Tuning Policy
Authenticator
Http Authenticator
Auth Type
httpClient
NoHttpAuth
Config
No Http Auth
Transport
Http Transport
Transport Type
httpClient
OkHttpTransport
Comm
Ok Http Transport
Http2 Enabled
false
Tls Connection Spec
Modern _tls
Request Body
Request Body

HttpClientService



Security Dashboard API

`BISecurityDashboardItemProvider`

Adds items to an existing card.

`BISecurityDashboardProvider`

Adds a new card.

`BISecurityDashboardProviderAgent`

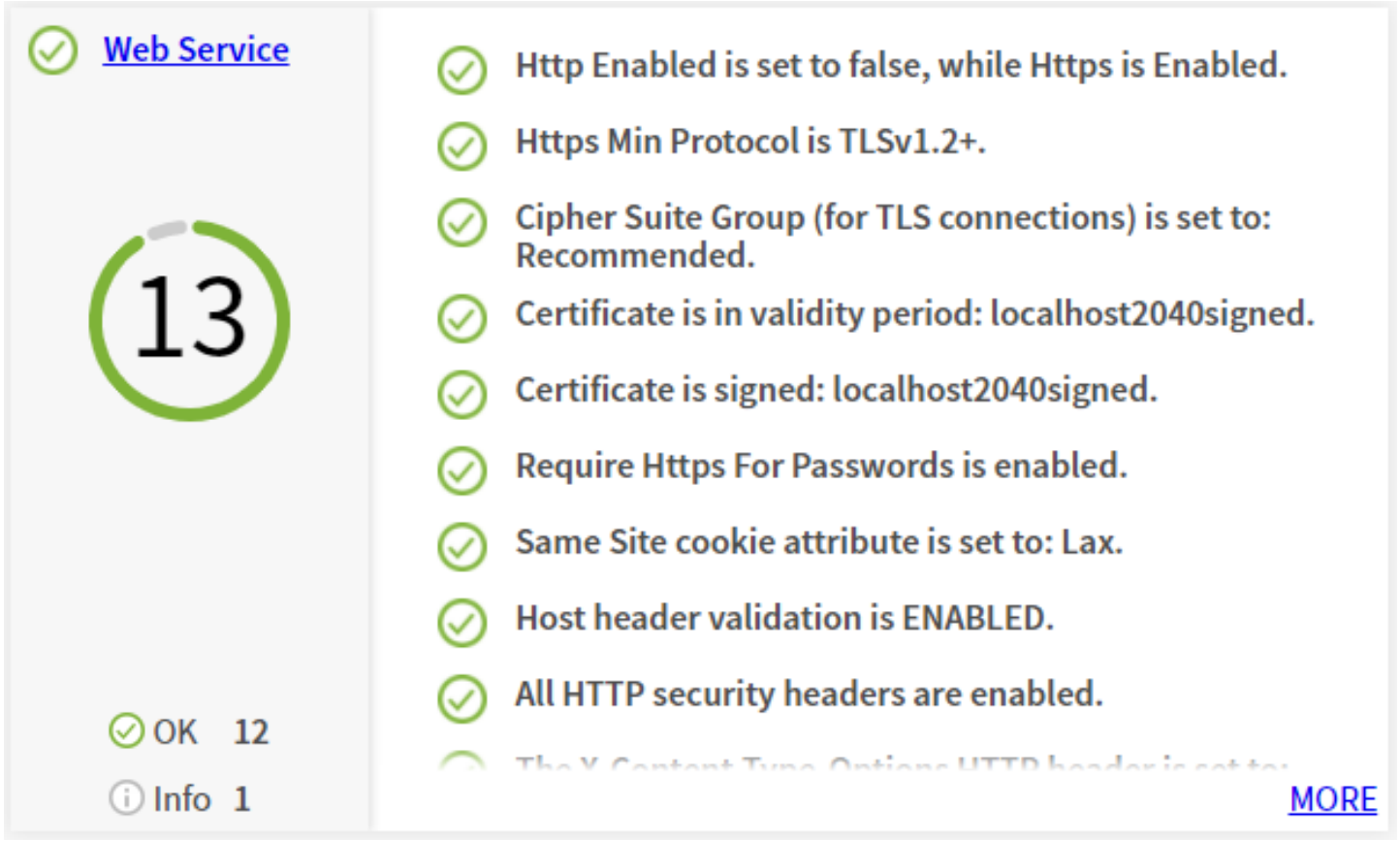
Adds a new card. Lets you split your code.

Add Items to an Existing Card

BISecurityDashboardItemProvider

TRIDIUM

Web Service Card



BISecurityDashboardItemProvider

Must implement **`BIAgent`**

Must be an agent on a
`BISecurityDashboardProvider` or
`BISecurityDashboardProviderAgent`

BISecurityDashboardItemProvider

```
@NiagaraType
public interface BISecurityDashboardItemProvider
  extends BInterface
{
  ...

  /**
   * Sets the object from which security dashboard items will be generated.  The default
   * implementation does nothing and is suitable for types that implement
   * BISecurityDashboardProvider (because that type is itself the items source) or types that will
   * retrieve the source(s) themselves.
   */
  default void setSecurityDashboardItemsSource(BIObject object)
  {
  }

  /**
   * Returns the version of the security dashboard items.
   */
  int getSecurityDashboardItemsVersion();

  /**
   * Returns a List of SecurityDashboardItem objects.
   * @see SecurityDashboardItem
   * @param cx The context, used for localization
   * @return List of the SecurityDashboardItems
   */
  List<SecurityDashboardItem> getSecurityDashboardItems(Context cx);
}
```

BISecurityDashboardItemProvider

```
@NiagaraType(agent = @AgentOn(types = "web:WebService"))
public class BHttpClientSecurityDashboardItemProvider
  extends BObject
  implements BISecurityDashboardItemProvider, BIAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    webService = object.as(BWebService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkClientsForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkClientsForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);
    checkDevicesForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);

    BHttpClientService service = BHttpClientService.service();
    if (service != null && service.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_CLIENTS_SUMMARY_OK, NON_DRIVER_CLIENTS_SUMMARY_WARNING,
        NON_DRIVER_CLIENTS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
```

BISecurityDashboardItemProvider

```
@NiagaraType(agent = @AgentOn(types = "web:WebService"))
public class BHttpClientSecurityDashboardItemProvider
  extends BObject
  implements BISecurityDashboardItemProvider, BIAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    webService = object.as(BWebService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkClientsForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkClientsForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);
    checkDevicesForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);

    BHttpClientService service = BHttpClientService.service();
    if (service != null && service.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_CLIENTS_SUMMARY_OK, NON_DRIVER_CLIENTS_SUMMARY_WARNING,
        NON_DRIVER_CLIENTS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
```

"object" is a **BWebService** because we're an agent on **BWebService**, which is a **BISecurityDashboardProvider**

BISecurityDashboardItemProvider

```
@NiagaraType(agent = @AgentOn(types = "web:WebService"))
public class BHttpClientSecurityDashboardItemProvider
  extends BObject
  implements BISecurityDashboardItemProvider, BIAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    webService = object.as(BWebService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNI
    checkClientsForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BA
    checkClientsForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION,
    checkDevicesForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);

    BHttpClientService service = BHttpClientService.service();
    if (service != null && service.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_CLIENTS_SUMMARY_OK, NON_DRIVER_CLIENTS_SUMMARY_WARNING,
        NON_DRIVER_CLIENTS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
```

```
private void checkClientsForWarning(final List<SecurityDashboardItem> items, String okSummaryKey,
                                    String warningSummaryKey, String descriptionKey,
                                    Predicate<BIHttpCommClient> secCheck, Context cx)
{
  List<String> warningClients = HttpClientRegister.getClients().stream()
    .filter(secCheck)
    .map(commClient -> getClientName(commClient, cx))
    .sorted()
    .collect(Collectors.toList());

  if (!warningClients.isEmpty())
  {
    items.add(builder.makeWarning()
      .withSummary(warningSummaryKey, String.join(", ", warningClients))
      .withDescription(descriptionKey));
  }
  else
  {
    items.add(builder.makeOk()
      .withSummary(okSummaryKey, String.join(", ", warningClients))
      .withDescription(descriptionKey));
  }
}
```

BISecurityDashboardItemProvider

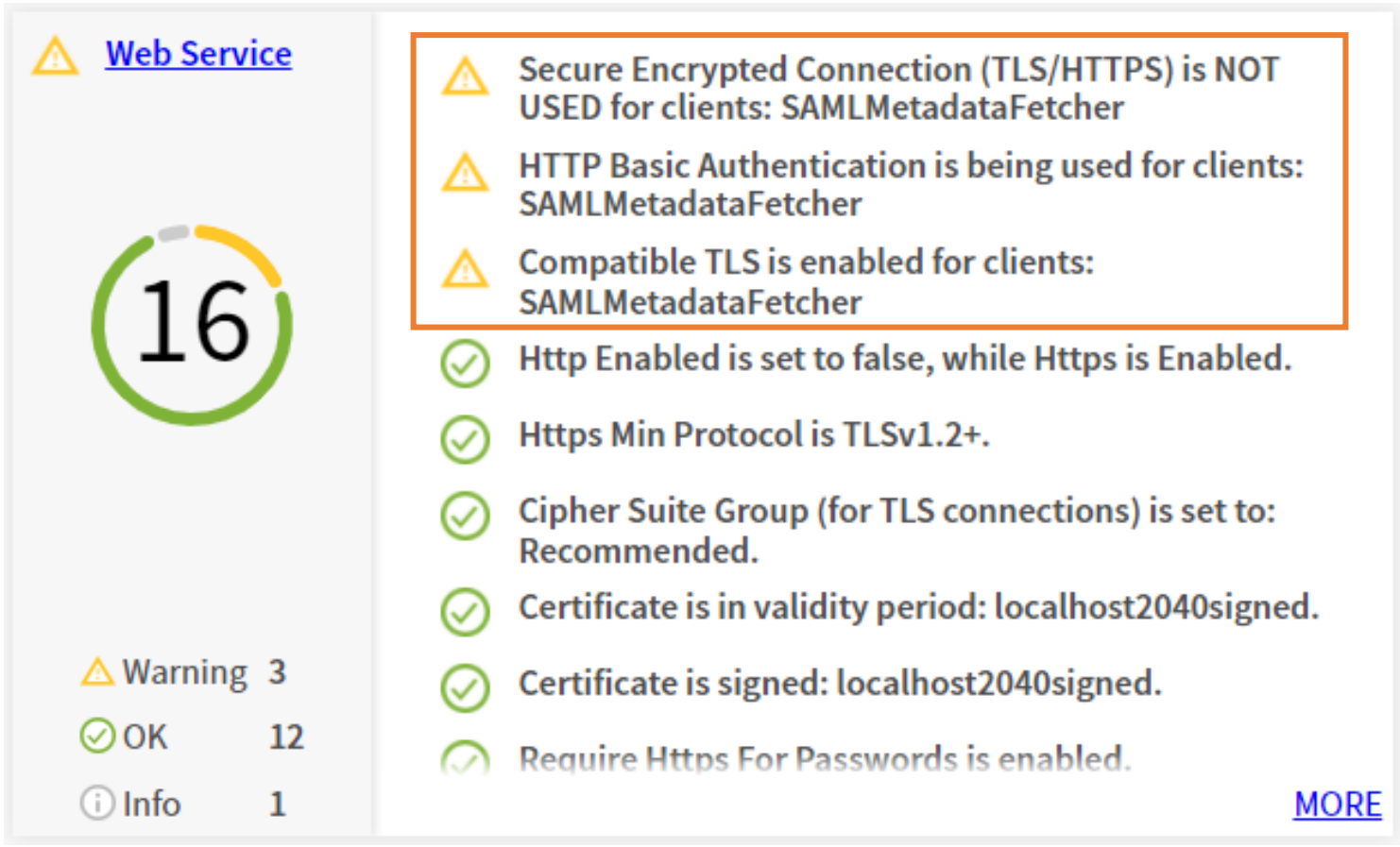
```
@NiagaraType(agent = @AgentOn(types = "web:WebService"))
public class BHttpClientSecurityDashboardItemProvider
  extends BObject
  implements BISecurityDashboardItemProvider, BIAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    webService = object.as(BWebService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNI
    checkClientsForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BA
    checkClientsForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION,
    checkDevicesForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);

    BHttpClientService service = BHttpClientService.service();
    if (service != null && service.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_CLIENTS_SUMMARY_OK, NON_DRIVER_CLIENTS_SUMMARY_WARNING,
        NON_DRIVER_CLIENTS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
```

```
private void checkClientsForWarning(final List<SecurityDashboardItem> items, String okSummaryKey,
                                    String warningSummaryKey, String descriptionKey,
                                    Predicate<BIHttpCommClient> secCheck, Context cx)
{
  List<String> warningClients = HttpClientRegister.getClients().stream()
    .filter(secCheck)
    .map(commClient -> getClientName(commClient, cx))
    .sorted()
    .collect(Collectors.toList());

  if (!warningClients.isEmpty())
  {
    items.add(builder.makeWarning()
      .withSummary(warningSummaryKey, String.join(", ", warningClients))
      .withDescription(descriptionKey));
  }
  else
  {
    items.add(builder.makeOk()
      .withSummary(okSummaryKey, String.join(", ", warningClients))
      .withDescription(descriptionKey));
  }
}
```

BISecurityDashboardItemProvider

```
@NiagaraType(agent = @AgentOn(types = "web:WebService"))
public class BHttpClientSecurityDashboardItemProvider
  extends BObject
  implements BISecurityDashboardItemProvider, BIAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    webService = object.as(BWebService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkClientsForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkClientsForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);
    checkDevicesForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUMMARY, COMPAT_TLS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isUsingCompatibleTls, cx);

    BHttpClientService service = BHttpClientService.service();
    if (service != null && service.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_CLIENTS_SUMMARY_OK, NON_DRIVER_CLIENTS_SUMMARY_WARNING,
        NON_DRIVER_CLIENTS_DESCRIPTION, BHttpClientSecurityDashboardItemProvider::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
```

The Result


Web Service
16
Warning 3
OK 12
Info 1
Secure Encrypted Connection (TLS/HTTPS) is NOT USED for clients: SAMLMetadataFetcher
HTTP Basic Authentication is being used for clients: SAMLMetadataFetcher
Compatible TLS is enabled for clients: SAMLMetadataFetcher
Http Enabled is set to false, while Https is Enabled.
Https Min Protocol is TLSv1.2+.
Cipher Suite Group (for TLS connections) is set to: Recommended.
Certificate is in validity period: localhost2040signed.
Certificate is signed: localhost2040signed.
Require Https For Passwords is enabled.
MORE

Create a New HttpClient Card

BHttpClient as a BISecurityDashboardProvider

TRIDIUM

BISecurityDashboardProvider

Extends **`BISecurityDashboardItemProvider`**

Is not a **`BIAgent`**

Specifies a hyperlink ord and header name

BISecurityDashboardProvider

```
@NiagaraType
public interface BISecurityDashboardProvider
  extends BISecurityDashboardItemProvider
{
  /**
   * Get the display name of the dashboard section.
   * @param cx The context, used for localization
   * @return localization info for the dashboard section display name
   */
  LexiconFormatInfo getSecurityDashboardSectionHeader(Context cx);

  /**
   * Returns the ORD of the service/security item that should be hyperlinked to
   * for more details. This ORD will be considered relative to the
   * {@code BISecurityDashboardProvider} instance, so in most cases,
   * {@code relativizeToSession()} should be used.
   *
   * @return the ORD to the service/security item. This may also return
   * {@code null} or {@code BOrd.DEFAULT} to indicate that no hyperlink should
   * be shown.
   */
  BOrd getSecurityDashboardSectionHyperlinkOrd();
}
```

Must also implement **`getSecurityDashboardItemsVersion()`**, **`getSecurityDashboardItems()`**, but not **`setSecurityDashboardItemsSource()`**

BISecurityDashboardProvider

```
public class BHttpClient
  extends BComponent
  implements BIHttpCommClient, BIHttpCommClientTransport, BIHttpCommClientWrite, BISecurityDashboardProvider
{
  @Override
  public LexiconFormatInfo getSecurityDashboardSectionHeader(Context cx)
  {
    return LexiconFormatInfo.make(TYPE, SECTION_HEADER, getClientName(this, cx));
  }

  @Override
  public BOrd getSecurityDashboardSectionHyperl
  {
    return getNavOrd().relativizeToSession();
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();
    checkClientForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, isInsecure(), cx);
    checkClientForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, isUsingBasicAuth(), cx);
    checkClientForWarning(items, COMPAT_TLS_SUMMARY_OK, COMPAT_TLS_SUMMARY_WARNING, COMPAT_TLS_DESCRIPTION, isUsingCompatibleTls(), cx);

    return Collections.unmodifiableList(items);
  }
```

```
private static final String SECTION_HEADER = "securityDashboard.httpClient.sectionHeader.client";
```

module.lexicon

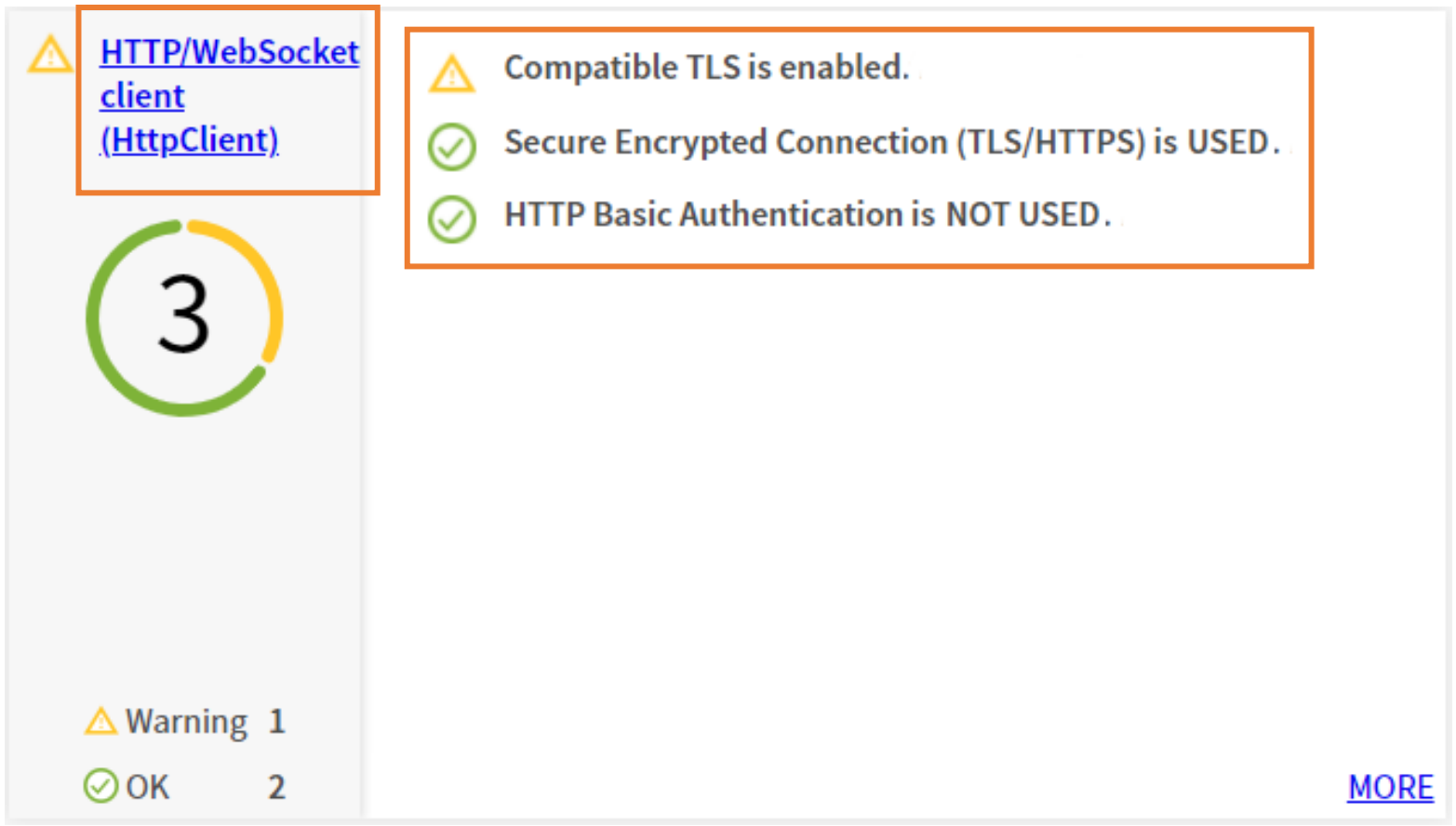
```
securityDashboard.httpClient.sectionHeader=HTTP/WebSocket client ({0})
```

BISecurityDashboardProvider

```
public class BHttpClient
  extends BComponent
  implements BIHttpCommClient, BIHttpCommClientTransport, BIHttpCommClientWrite, BISecurityDashboardProvider
{
  @Override
  public LexiconFormatInfo getSecurityDashboardSectionHeader(C
  {
    return LexiconFormatInfo.make(TYPE, SECTION_HEADER, getCli
  }

  @Override
  public BOrd getSecurityDashboardSectionHyperlinkOrd()
  {
    return getNavOrd().relativizeToSession();
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();
    checkClientForWarning(items, INSECURE_SUMMARY_OK, INSECURE_SUMMARY_WARNING, INSECURE_DESCRIPTION, isInsecure(), cx);
    checkClientForWarning(items, BASIC_SUMMARY_OK, BASIC_SUMMARY_WARNING, BASIC_DESCRIPTION, isUsingBasicAuth(), cx);
    checkClientForWarning(items, COMPAT_TLS_SUMMARY_OK, COMPAT_TLS_SUMMARY_WARNING, COMPAT_TLS_DESCRIPTION, isUsingCompatibleTls(), cx);

    return Collections.unmodifiableList(items);
  }
```

```
private void checkClientForWarning(final List<SecurityDashboardItem> items,
String okSummaryKey, String warningSummaryKey, String descriptionKey,
boolean isInsecure, Context cx)
{
  if (isInsecure)
  {
    items.add(builder.makeWarning()
      .withSummary(warningSummaryKey)
      .withDescription(descriptionKey));
  }
  else
  {
    items.add(builder.makeOk()
      .withSummary(okSummaryKey)
      .withDescription(descriptionKey));
  }
}
```

The Result


HTTP/WebSocket client (HttpClient)
3
Warning 1
OK 2
Compatible TLS is enabled.
Secure Encrypted Connection (TLS/HTTPS) is USED.
HTTP Basic Authentication is NOT USED.
MORE

Create a New HttpClient Card

BHttpClientService as a BISecurityDashboardProviderAgent

TRIDIUM

BISecurityDashboardProviderAgent

Extends **`BISecurityDashboardProvider`**

Is a **`BIAgent`**

Specifies a hyperlink ord, a header name, and an item source

BISecurityDashboardProviderAgent

```
@NiagaraType
public interface BISecurityDashboardProviderAgent
  extends BIAgent, BISecurityDashboardProvider
{
//region /*+ ------------ BEGIN BAJA AUTO GENERATED CODE ------------ +*/

  Type TYPE = Sys.loadType(BISecurityDashboardProviderAgent.class);

//endregion /*+ ------------ END BAJA AUTO GENERATED CODE -------------- +*/
}
```

Must implement
getSecurityDashboardItemsVersion(),
getSecurityDashboardSectionHeader(),
getSecurityDashboardSectionHyperlinkOrd(),
getSecurityDashboardItems(), and
setSecurityDashboardItemsSource()

BISecurityDashboardProviderAgent

```
@NiagaraType(agent = @AgentOn(types = "httpClient:HttpClientService"))
public final class BHttpClientSecurityDashboardAgent extends BObject implements BISecurityDashboardProviderAgent
{
  @Override
  public void setSecurityDashboardItemsSource(BIObject object)
  {
    httpClientService = object.as(BHttpClientService.class);
    builder = new SecurityDashboardItemBuilder(this);
  }

  @Override
  public LexiconFormatInfo getSecurityDashboardSectionHeader(Context cx)
  {
    return LexiconFormatInfo.make(TYPE, SECTION_HEADER);
  }

  @Override
  public BOrd getSecurityDashboardSectionHyperlinkOrd()
  {
    return httpClientService.getNavOrd().relativizeToSession();
  }

  @Override
  public int getSecurityDashboardItemsVersion()
  {
    return SECURITY_DASHBOARD_ITEMS_VERSION;
  }

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    ...
  }
}
```

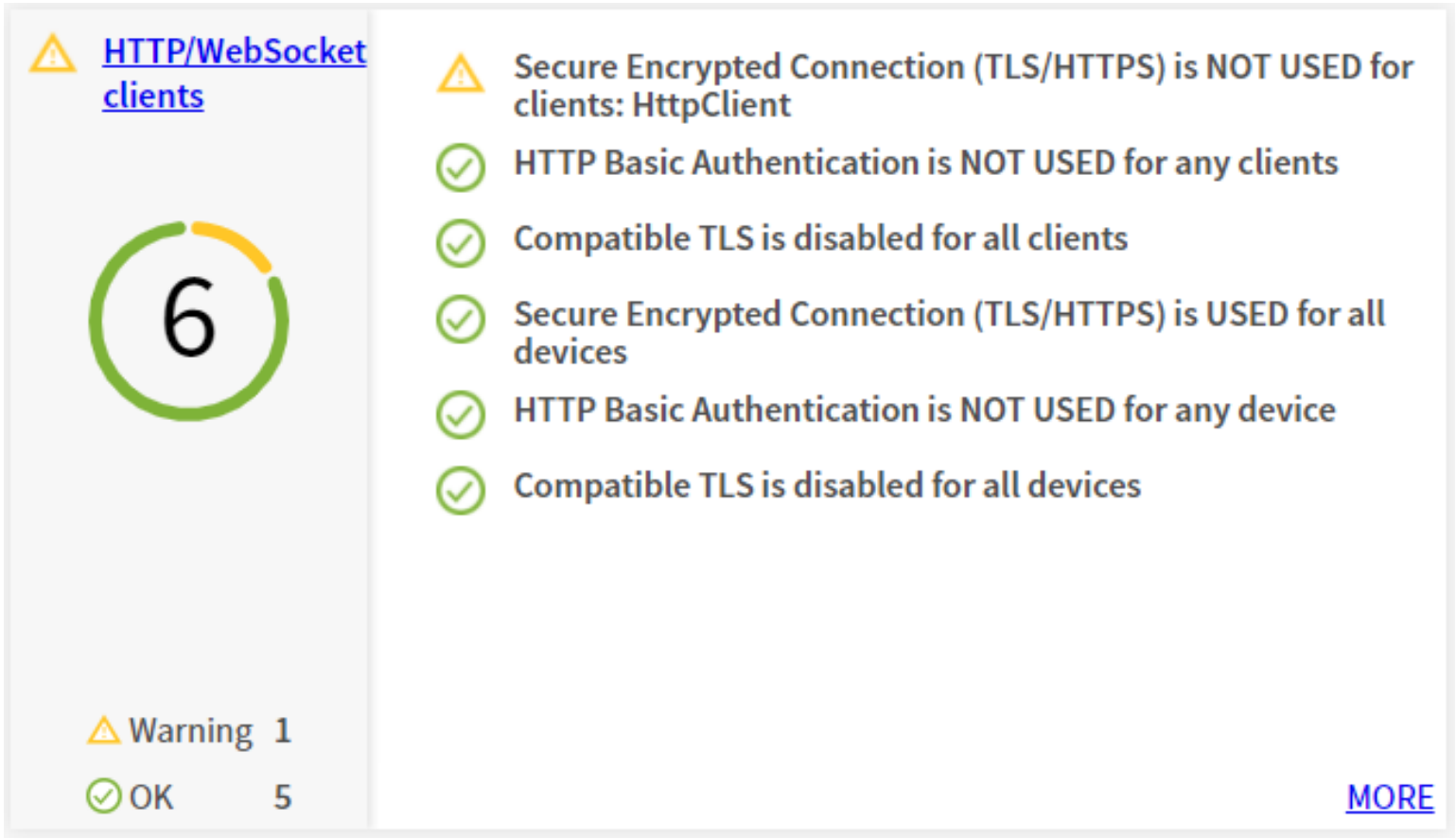
object is a **BHttpClientService**
because we're an agent on that type

```
private static final String SECTION_HEADER = "securityDashboard.httpClient.sectionHeader";

module.lexicon
securityDashboard.httpClient.sectionHeader=HTTP/WebSocket clients
```

BISecurityDashboardProviderAgent

```
@NiagaraType(agent = @AgentOn(types = "httpClient:HttpClientService"))
public final class BHttpClientSecurityDashboardAgent extends BObject implements BISecurityDashboardProviderAgent
{
  ...

  @Override
  public List<SecurityDashboardItem> getSecurityDashboardItems(Context cx)
  {
    List<SecurityDashboardItem> items = new ArrayList<>();

    checkClientsForWarning(items, INSECURE_SUM_OK, INSECURE_SUM_WARNING, INSECURE_DESC, BHttpClientSecurityDashboardAgent::isInsecure, cx);
    checkClientsForWarning(items, BASIC_SUM_OK, BASIC_SUM_WARNING, BASIC_DESC, BHttpClientSecurityDashboardAgent::isUsingBasicAuth, cx);
    checkClientsForWarning(items, COMPAT_TLS_SUM_OK, COMPAT_TLS_SUM_WARNING, COMPAT_TLS_DESC, BHttpClientSecurityDashboardAgent::isUsingCompatibleTls, cx);
    checkDevicesForWarning(items, INSECURE_SUM_OK, INSECURE_SUM_WARNING, INSECURE_DESC, BHttpClientSecurityDashboardAgent::isInsecure, cx);
    checkDevicesForWarning(items, BASIC_SUM_OK, BASIC_SUM_WARNING, BASIC_DESC, BHttpClientSecurityDashboardAgent::isUsingBasicAuth, cx);
    checkDevicesForWarning(items, COMPAT_TLS_SUM_OK, COMPAT_TLS_SUM_WARNING, COMPAT_TLS_DESC, BHttpClientSecurityDashboardAgent::isUsingCompatibleTls, cx);

    if (httpClientService.getEnableNonDriverClients())
    {
      checkClientsForWarning(items, NON_DRIVER_SUM_OK, NON_DRIVER_SUM_WARNING, NON_DRIVER_DESC, BHttpClientSecurityDashboardAgent::isNonDriverClient, cx);
    }

    return Collections.unmodifiableList(items);
  }
}
```

The Result


HTTP/WebSocket clients
6
Warning 1
OK 5
Secure Encrypted Connection (TLS/HTTPS) is NOT USED for clients: HttpClient
HTTP Basic Authentication is NOT USED for any clients
Compatible TLS is disabled for all clients
Secure Encrypted Connection (TLS/HTTPS) is USED for all devices
HTTP Basic Authentication is NOT USED for any device
Compatible TLS is disabled for all devices
MORE

Which Interface Should I Use?

TRIDIUM

Summary

`BISecurityDashboardItemProvider`

Adds items to existing card

`BISecurityDashboardProvider`

Adds a new card

`BISecurityDashboardProviderAgent`

Adds a new card, new class

How to Choose

How do you want the dashboard to look?

How many cards will it create?

What component will it link back to?

What do the individual items look like?

Questions?

TRIDIUM