Application Server Security

Overview

Even the most secure back-end infrastructures face risks if front-end setups don't support their high security standards. Yellowfin software can be further secured by tweaking some settings and disabling the ones you don't expect to use to help maintain very high security.

Set up HTTPS

HTTPS can be configured at a Tomcat level, allowing full control over available protocols and whether the service is accessible via HTTP. A detailed walk-through can be found here on the Yellowfin Community site.

Improve Yellowfin web.xml hardening configurations

There are several considerations when hardening at the application server level. Your needs will depend on different factors, such as public or private deployment, whether Yellowfin is embedded into another application, network configurations, and which APIs will be used.

The Yellowfin web.xml file is found at `<yellowfinInstall>/appserver/webapps/ROOT/WEB-INF/web.xml`.

Automatic redirection

When Yellowfin is configured to serve over HTTPS, it can also be configured to automatically redirect to the secure port.

```
<!-- Forward traffic to secure port -->
  <security-constraint>
          <web-resource-collection>
                  <web-resource-name>Automatic TLS Forwarding</web-resource-name>
                  <url-pattern>/*</url-pattern>
          </web-resource-collection>
          <user-data-constraint>
                  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
          </user-data-constraint>
  </security-constraint>
```

Secure session cookies

To set the secure attribute on JSESSION cookies, insert the following into your web.xml. This attribute declares that the cookie should be sent over secure channels only, such as HTTPS.

```
<!-- Secure cookie attribute -->
  <session-config>
          <cookie-config>
                  <secure>true</secure>
          </cookie-config>
  </session-config>
```

OWASP secure headers

Yellowfin has implemented an OWASP Secure Headers filter that can be used to configure security headers as per these standards. An example configuration can be found below; however, these values should be adjusted to meet organizational standards or policies. Note that for `Content-Security-Policy`, the set of values defined below in `<param-value>` is the minimum entry for Yellowfin functionality.

```
<!-- OWASP Secure Headers -->
  <filter>
          <filter-name>OWASPSecureHeaders</filter-name>
          <filter-class>com.hof.servlet.OWASPSecureHeaders</filter-class>
          <init-param>
                  <param-name>Strict-Transport-Security</param-name>
                  <param-value>max-age=315360000</param-value>
          </init-param>
          <init-param>
                  <param-name>X-Frame-Options</param-name>
                  <param-value>sameorigin</param-value>
          </init-param>
          <init-param>
                  <param-name>X-Content-Type-Options</param-name>
                  <param-value>nosniff</param-value>
          </init-param>
          <init-param>
                  <param-name>X-Permitted-Cross-Domain-Policies</param-name>
                  <param-value>none</param-value>
          </init-param>
          <init-param>
            <param-name>Content-Security-Policy</param-name>
            <param-value>default-src 'self' 'unsafe-inline' 'unsafe-eval'; font-src data: http:; img-src data:
http:</param-value>
          </init-param>
  </filter>
  <filter-mapping>
          <filter-name>OWASPSecureHeaders</filter-name>
          <url-pattern>/*</url-pattern>
  </filter-mapping>
```

CSRF filter

Enabling Yellowfin's CSRF filter adds a nonce to application requests, protecting your instance from cross-site request forgery attacks.

```
<!-- Yellowfin CSRF Filter -->
  <filter>
          <filter-name>CSRFFilter</filter-name>
          <filter-class>com.hof.servlet.CSRFFilter</filter-class>
          <init-param>
                  <param-name>AllowedEntry</param-name>
                  <param-value>/info.jsp,info_/threads.jsp</param-value>
          </init-param>
  </filter>
  <filter-mapping>
          <filter-name>CSRFFilter</filter-name>
          <url-pattern>*.i4</url-pattern>
  </filter-mapping>
  <filter-mapping>
          <filter-name>CSRFFilter</filter-name>
          <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
```

Of particular importance here is the `AllowedEntry` parameter. This comma-separated list of URLs will instruct Yellowfin to bypass the nonce check and allow entry to any of the pages from these URLs.

Referrer filter

The referrer filter is offered as an additional layer of security further to the CSRF filter. This validates the referrer field in a request against the `hostingdomainURL` parameter before processing the request. The `ignore` parameter should have the items listed below at a minimum, as well as any other entry points from external applications, such as when integrating Yellowfin.

```
<filter>
          <filter-name>RefererFilter</filter-name>
          <filter-class>com.hof.servlet.RefererFilter</filter-class>
          <init-param>
                  <param-name>hostingdomainURL</param-name>
                  <param-value>https://54.151.18.219</param-value>
          </init-param>
          <init-param>
                  <param-name>ignore</param-name>
                  <param-value>/RunDashboard.i4,/RunReport.i4,/*.js</param-value>
          </init-param>
  </filter>
  <filter-mapping>
          <filter-name>RefererFilter</filter-name>
          <url-pattern>/*</url-pattern>
  </filter-mapping>
```

Disable or restrict access to unused APIs

Yellowfin 9.3 and later releases feature the new REST API. The legacy SOAP Web Services API can be disabled if REST is to be used. Search for `AxisServlet` — the snippet that enables the legacy API — and comment out this block to turn this off.

```
<!-- Web Services Servlet
 <servlet>
   <servlet-name>AxisServlet</servlet-name>
   <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
 </servlet>
-->
```

Or for the REST and JS API, if not in use, add the following:

```
<security-constraint>
   <web-resource-collection>
   <web-resource-name>UnusedAPI's</web-resource-name>
     <url-pattern>/api</url-pattern>
     <url-pattern>/JsAPI</url-pattern>
   </web-resource-collection>
   <auth-constraint>
     <role-name>denyaccess</role-name>
   </auth-constraint>
 </security-constraint>
```

Informational pages

A collection of informational pages is available for support and analyses. These pages may contain system information such as OS version, RDBMS types, or running application threads.

URL	Description
`https://<YellowfinHostURL>/info.jsp`	Provides details of the operating environment including system, application, locale, license, server-side software details and library versions.
`https://<YellowfinHostURL>/info_cache.jsp`	Displays the Yellowfin cache status and capacity information.
`https://<YellowfinHostURL>/info_cluster.jsp`	Displays information on Yellowfin cluster configuration, where applicable.
`https://<YellowfinHostURL>/info_language.jsp`	Displays available and default locales.
`https://<YellowfinHostURL>/info_threads.jsp`	Outputs JVM threads and their status.

`https://<YellowfinHostURL>/info_threads_enhanced.jsp`	Expands on the above by including thread ownership, when applicable, and CPU time.

You can disable these by adding the snippet below.

```
 <security-constraint>
    <web-resource-collection>
    <web-resource-name>server-info</web-resource-name>
      <url-pattern>/info.jsp</url-pattern>
      <url-pattern>/info_browser.jsp</url-pattern>
      <url-pattern>/info_cache.jsp</url-pattern>
      <url-pattern>/info_threads.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>denyaccess</role-name>
    </auth-constraint>
  </security-constraint>
```

Keep in mind that these pages may be requested by Support when encountering an issue. There are alternatives to disabling these pages that will allow you to restrict access based on different factors. One example is the `RemoteAddrFilter`, which will restrict access by IP address.

```
 <filter>
    <filter-name>IPFilter</filter-name>
    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
    <init-param>
        <param-name>allow</param-name>
        <param-value>127\.\d+\.\d+\.\d+</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>IPFilter</filter-name>
      <url-pattern>/info.jsp</url-pattern>
      <url-pattern>/info_browser.jsp</url-pattern>
      <url-pattern>/info_cache.jsp</url-pattern>
      <url-pattern>/info_threads.jsp</url-pattern>
</filter-mapping>
```

This will allow the page to be accessed from any 127.x.x.x address. See more in the Tomcat Documentation under the Access Control heading.

Deployment and Hardening Guide

Back to the Overview

- General Security Infrastructure Considerations
- Application Server Security
- Yellowfin UI Security Settings

top