Security

Tapestry has a number of **security** features designed to harden your application against unwanted intrusion and denial of service.

Related Articles

- Security
- Integrating with Spring Framework
- HTTPS
- Security FAQ

HTTPS-only Pages

Main Article: HTTPS

Tapestry provides several annotations and configuration settings that you can use to ensure that all access to certain pages (or all pages) occurs only via the encrypted HTTPS protocol. See HTTPS for details.

Controlling Page Access

JumpStart Demo:
Protecting Pages

For simple access control needs, you can contribute a ComponentRequestFilter with your custom logic that decides which pages should be accessed by which users. The Tapestry Hotel Booking app demonstrates this approach with an `@AnonymousAccess` annotation along with a ComponentRequestFilter named `AuthenticationFilter.java`. The filter enforces security by intercepting all requests to pages that don't have that annotation, and it redirects those requests to the login page. JumpStart has a similar demo.

For more advanced needs see the Security Framework Integration section below.

White-listed Pages

Pages whose component classes are annotated with @WhitelistAccessOnly will only be displayed to users (clients) that are on the *whitelist*. By default the whitelist consists only of clients whose fully-qualified domain name is "localhost" (or the IP address equivalent, 127.0.0.1 or 0:0:0:0:0:0:0:1), but you can customize this by contributing to the ClientWhitelist service in your application's module class (usually AppModule.java):

AppModule.java (partial) – simple inline example

```
@Contribute(ClientWhitelist.class)
public static void provideWhitelistAnalyzer(OrderedConfiguration<WhitelistAnalyzer> configuration)
{
    configuration.add("MyCustomAnalyzer", new WhitelistAnalyzer()
    {
        public boolean isRequestOnWhitelist(Request request)
        {
            // add your custom logic here and return true or false
            return true;
        }
    }, "before:*");
}
```

Sometimes, in production, a firewall or proxy may make it look like the client web browser originates from localhost, with the consequence that whitelisted pages may be visible to all users. See the Security FAQ for how to deal with this.

Asset Security

Main Article: Security

Tapestry serves assets (static content such as CSS files, images, and JavaScript, many of which are on the classpath alongside your compiled class files) to the client. Because of this, great care has gone into ensuring that certain file types cannot be served to the client. By default, file ending with ".class', ". tml" and ".properties" can be served to the client only if the request includes the file's MD5 checksum. As you would expect, that blacklist can be extended. See Assets for more information.

Protecting Serialized Object Data on the Client

As of version 5.3.6, Tapestry integrates a hash-based message authentication code (HMAC) into serialized Java object data that it sends to the client (generally, this means the `t:formdata` hidden field used by the Form component). This ensures that the hidden binary object data is guaranteed to be unaltered when it returns to the server upon form (or AJAX) submission. The HMAC pass phrase is set using the tapestry.hmac-passphrase configuration symbol. If you don't set that value, you'll see a warning message in the browser, like this:

```
The symbol 'tapestry.hmac-passphrase' has not been configured. This is used to configure hash-based message
authentication of Tapestry data stored in forms, or in the URL. You application is less secure, and more
vulnerable to denial-of-service attacks, when this symbol is not configured.
```

The solution is to set the tapestry.hmac-passphrase to some value (any fixed, private string, such as 30 to 40 random-looking characters, will do) in your application's module class (usually AppModule.java).

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery is a type of security vulnerability in which legitimate, authorized users may be made to unwittingly submit malicious requests to your web application.

Tapestry-csrf-protection is a 3rd party module that has several features for preventing CSRF attacks. It protects all component event handlers (event links, forms, etc.) by adding a CSRF token to event links and adds a CSRF token as a hidden field to all forms. Tokens are generated on a per-session basis.

Security Framework Integration

Tapestry does not lock you into a specific authentication/authorization implementation. There are integration modules available for the more popular open source Java security frameworks. A popular choice among Tapestry users is tapestry-security (based on Apache Shiro) from Tynamo.org. It is always kept up-to-date with the latest Tapestry versions and offers several supporting security modules (e.g. tapestry-security-jpa, tynamo-federatedaccounts). There's also an integration module available for Spring Security but lately, it hasn't kept up with the latest versions of Tapestry 5.

Additional information:

- Tynamo-federatedaccounts is an add-on to the tapestry-security module, providing federated (third-party) authentication with Facebook, Twitter or Google.
- To include OpenID with Spring Security in your application, see the following Wiki entry: http://wiki.apache.org/tapestry/Tapestry5HowToSpringSecurityAndOpenId

Vulnerability Disclosures

CVE-2019-0195: File reading Leads to Java Deserialization Vulnerability.

Disclosure date: September 13th, 2019

Versions affected: all Apache Tapestry versions from 5.4.0 (including its betas) through 5.4.3

Description: Manipulating classpath asset file URLs, an attacker could guess the path to a known file in the classpath and have it downloaded. If the attacker found the file with the value of the tapestry.hmac-passphrase configuration symbol, most probably the webapp's AppModule class, the value of this symbol could be used to craft a Java deserialization attack, thus running malicious injected Java code. The vector would be the t:formdata parameter from the Form component.

Mitigation: Upgrade to Tapestry 5.4.5, which is a drop-in replacement for any 5.4.x version.

Credit: Ricter Zheng

CVE-2019-0207: Apache Tapestry 5.4.2 Path Traversal vulnerability

Disclosure date: September 13th, 2019

Versions affected: all Apache Tapestry versions from 5.4.0 (including its betas) through 5.4.4.

Description: Tapestry processes assets `/assets/ctx` using classes chain `StaticFilesFilter -> AssetDispatcher -> ContextResource`, which doesn't filter the character `\`, so attacker can perform a path traversal attack to read any files on Windows platform.

Mitigation: Upgrade to Tapestry 5.4.5, which is a drop-in replacement for any 5.4.x version.

Credit: Ricter Zheng

CVE-2019-10071: New Issue in Fix for CVE-2014-1972

Disclosure date: September 13th, 2019

Versions affected: all Apache Tapestry versions from 5.4.0 (including its betas) through 5.4.3

Description: The code which checks HMAC in form submissions used
String.equals() for comparisons, which results in a timing side channel vulnerability in
the comparison of the HMAC signatures. This could lead to remote code
execution if an attacker is able to determine the correct signature for
their payload. The comparison should have been done with a constant time algorithm instead.

Mitigation: Upgrade to Tapestry 5.4.5, which is a drop-in replacement for any 5.4.x
version.

Credit:

```
David Tomaschik of the Google Security Team
```

CVE-2019-10071: Bypass of the fix for CVE-2019-0195

Disclosure date: March 14th, 2021

Versions affected: all Apache Tapestry versions from 5.4.0 (including its betas) through 5.6.1, plus 5.7.0.

Description: A critical unauthenticated remote code execution vulnerability was found all recent versions of Apache Tapestry. The affected versions include 5.4.5, 5.5.0, 5.6.2 and 5.7.0.

The vulnerability I have found is a bypass of the fix for CVE-2019-0195.

Recap:

Before the fix of CVE-2019-0195 it was possible to download arbitrary class files from the classpath by providing a crafted asset file URL. An attacker was able to download the file `AppModule.class` by requesting the URL `http://localhost:8080/assets/something/services/AppModule.class` which contains a HMAC secret key. The fix for that bug was a blacklist filter that checks if the URL ends with `.class`, `.properties` or `.xml`.

Bypass:

Unfortunately, the blacklist solution can simply be bypassed by appending a `/` at the end of the URL: `http://localhost:8080/assets/something/services/AppModule.class/`

The slash is stripped after the blacklist check and the file `AppModule.class` is loaded into the response. This class usually contains the HMAC secret key which is used to sign

serialized Java objects. With the knowledge of that key an attacker can sign a Java gadget chain that leads to RCE (e.g. CommonsBeanUtils1 from ysoserial).

Solution for this vulnerability:

* For Apache Tapestry 5.4.0 to 5.6.1, upgrade to 5.6.2 or later.

* For Apache Tapestry 5.7.0, upgrade to 5.7.1 or later.

This issue is being tracked as TAP5-2663

Credit:

Apache Tapestry would like to thank Johannes Moritz for finding and notifying this vulnerability

CVE-2022-31781: Regular Expression Denial of Service (ReDoS) in ContentType.java. (GHSL-2022-022)

Disclosure date: July 12th, 2022

Versions affected:

This issue affects Apache Tapestry 5.8.1 and earlier.

Severity: low

Description:

Apache Tapestry up to version 5.8.1 is vulnerable to Regular Expression Denial of Service (ReDoS) in the way it handles Content Types. Specially crafted Content Types may cause catastrophic backtracking, taking exponential time to complete.

Specifically, this is about the regular expression used on the parameter of the org.apache.tapestry5.http.ContentType class.

Apache Tapestry 5.8.2 has a fix for this vulnerability.

Notice the vulnerability cannot be triggered by web requests in Tapestry code alone. It would only happen if there's some non-Tapestry codepath passing some outside input to the ContentType class constructor.

Acknowledgements:
CodeQL team members [@atorralba (Tony Torralba)](https://github.com/atorralba) and [@joefarebrother (Joseph Farebrother)](https://github.com/joefarebrother).