Security Tokens

Table of Contents

- 1 Security Tokens
  - 1.1 Turning on secure tokens.
    - 1.1.1 container.js configuration
- 2 Tokens on the client
  - 2.1 Container tokens
    - 2.1.1 Container token refresh
  - 2.1.2 Making sure the container token is valid before you do something.
  - 2.2 Gadget tokens
    - 2.2.1 Gadget token refresh
- 3 Tokens on the server
  - 3.1 Java Interfaces and Classes of Note
    - 3.1.1 SecurityToken
    - 3.1.2 AbstractSecurityToken
    - 3.1.3 SecurityTokenCodec
    - 3.1.4 BlobCrypter

Security Tokens

Security tokens are used by shindig to sign requests made by container pages and individual gadgets back to the shindig server. The token is typically a tuple of the container, the authenticated user, the gadget as well as an expiration time. There are two fundamental types of security tokens: secure and insecure. Insecure tokens in Shindig are unencrypted, plaintext strings and should only be used for development and testing. Secure tokens, on the other hand, are encrypted by the Shindig server and are suitable for production use.

The functioning of secure security tokens is similar to SSO systems such as Ltpa. The expiration time acts as a means to prevent request forgery by ensuring that an intercepted token is only valid for a set period of time. "secure" tokens are encrypted server side so that new tokens can only be generated by attackers with knowledge of the security key and algorithm employed by an instance of shindig.

Turning on secure tokens.

container.js configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gadgets.securityTokenType</td>
<td>String</td>
<td>insecure</td>
<td>Set to &quot;secure&quot; to enable secure tokens. The default value is &quot;insecure&quot;</td>
</tr>
<tr>
<td>gadgets.securityTokenKey</td>
<td>String</td>
<td>N/A</td>
<td>Only applicable when gadgets.securityTokenType is set to &quot;secure&quot;, this value determines the key to use for token encryption. The value of this field can be the encryption key itself, a &quot;res://&quot; url that points to a file containing the key on the classpath, or a &quot;file://&quot; url that points to a file containing the key on the file system. One can generate an encryption key using:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dd if=/dev/random bs=32 count=1</td>
</tr>
<tr>
<td>gadgets.securityTokenTTL</td>
<td>int</td>
<td>3600</td>
<td>The time-to-live for security tokens in seconds. The default value is one hour (3600 seconds). Set this configuration value to change how long security tokens will live in your container.</td>
</tr>
</tbody>
</table>

This mail thread is outdated but may still be useful as a supplement to the information on this page.

Tokens on the client

Container tokens

Container tokens are held by the container page and authenticate container transactions to the shindig server. Since shindig tokens can expire, the container will probably have to periodically refresh its token in order to keep it valid.
Container token refresh

Until v2.5, container token refresh was left up to the container owner to implement. As of 2.5 there are features in the CommonContainer feature to help you manage the expiration of the container's security token.

It will require the container implementation to specify a function in the container config (GET_CONTAINER_TOKEN) to do fetch an updated token.

The built-in refresh mechanism will typically call this fetch function when the container's token reaches 80% of it's time-to-live(ttl). This is an asynchronous operation, so the fetch function should call the result function with the appropriate parameters when it has a result to report (including errors).

The way the result function is called determines is important to the functionality of the refresh function.

1. Successfully got a new token! Report the new token and the token's ttl to the result function.

```
result(token, ttl);
```

2. There was a temporary error in getting the token, call the result function so that it will call us again in about 5 seconds to try again.

```
var undef;
result(undef, 5);
```

3. There was a fatal error. We're probably not going to be able to successfully refresh the security token any time soon (try again in 1 hour). The error message here is important, it triggers the fatal error behavior.

```
var undef;
result(undef, 3600, 'Fatal Error!');
```

Making sure the container token is valid before you do something.

Within container code, there are places where we perform an operation based on having a valid container security token. We typically do something like this:

```
myContainer.updateContainerSecurityToken(function(error) {
    // Do something that requires a security token, like fetching gadget meta-data.
});
```

This will cause the provided callback to wait until there is a valid security token if the current token is no longer valid. If and when the container entered a situation where it is maintaining a queue of callbacks to run, case 2 above will continue to hold all callbacks until case 1 can be achieved. In the event you find yourself in case 3, reporting the fatal error will cause all queued callbacks to execute with the provided error message so that the container can show updated user experience and give the user an indication that something went wrong and they should try again in 1 hour.

The provided callback will be called with no parameters (error will be undefined) if everything went according to plan. If error is defined, there was a fatal error and error is the message the container provided. The callback should probably handle or report the error appropriately so that the user will know what's going on.

In some cases, you never really care if there's an error or not, you just always want to make this call with a valid token no matter what. Using the lazy param, we requeue our function in a way that will not force an immediate attempt to refresh the token but only add our callback to the queue to be notified when we finally do have a token.
function myCallback(error) {
    var undef;
    if (error == undef) {
        myContainer.updateContainerSecurityToken(myCallback, true);
    } else {
        // Do my important work
    }
}
myContainer.updateContainerSecurityToken(myCallback);

Your callback may be called multiple times with a fatal error condition before it is called with success. All callbacks should re-queue themselves with the lazy param (set to true) if they need to be re-queued.

When a call to updateContainerSecurityToken comes in, the container will immediately try to fetch a new token if the following conditions are all met:

1. The current token is expired or about to expire.
2. The container is not currently fetching a new token.
3. The lazy param is not true.

Gadget tokens

Container tokens are tracked the CommonContainer feature code and authenticate container transactions to the shindig server for the gadget they belong to. Since shindig tokens can expire, the container will probably have to periodically refresh gadget tokens in order to keep them valid.

Gadget token refresh

The gadget token refresh implementation is older than the container token one, and is not quite as capable. Future efforts to improve the functionality of the gadget token refresh are planned for post 2.5.

Tokens on the server

Java Interfaces and Classes of Note

SecurityToken

AbstractSecurityToken

SecurityTokenCodec

BlobCrypter