Security Frameworks for Scala

by Domingo Valera • November 03, 2015 • scala • security • frameworks • functional programming

In our continuous effort to improve and make our software more secure, we've been exploring some of the security features found in the most popular Scala frameworks. Here are our top picks and a brief explanation of their security features:

Framework	Authentication	Authorization	CSRF	XSS	SQL Injection
Play	✓	✓			
Authentication and Authorization module	✓	✓			
Authenticity Token module			✓		
Deadbolt 2		✓			
Play-pac4j	✓				
Scala-oauth2-provider	✓				
SecureSocial	✓				
Silhouette	✓				
Stateless client authentication	✓				
Lift	✓	✓	✓	✓	✓
Akka (Akka-http)	✓	✓			
Akka-http-sessions	✓		✓		
Akka-http-extensions	✓	✓			
Spray	✓	✓			

Play:

The Play framework is one of the most frequently used web frameworks in Scala. From a security point-of-view, Play 2 provides a Security trait that can be used to create secure actions.

If you need more advanced security options, you're encouraged to use the Authentication and Authorization and the Authenticity Token modules, the Deadbolt 2 plugin, Secure Social, Silhouette or the Stateless client authentication modules. We'll talk a little bit more about these in a moment.

In the Play 2 documentation you can find a few guides to increment the security of various parts of your application such as:

- Configuring Security Headers
- Protecting against Cross Site Request Forgery
- Configuring WS SSL

Finally, information on the Security Vulnerabilities found in Play through the present day is also available. It's important to visit this page often to make sure that the version of Play you're using is not affected by a certain vulnerability. You can deduce by this that it is also extremely important to make sure you upgrade to the latest version of Play.

Website: https://www.playframework.com

Let's take a look at some of the modules we can use to increase the security of our applications:

Authentication and Authorization module:

This module provides more secure authentication and authorization features in Play2.x than the original trait Security available in Play 2. It creates a unique SessionID using a secure random number generator that allows the user to invalidate the session even if the sessionId cookie was intercepted.

Website: https://github.com/t2v/play20-auth

Authenticity Token module:

This module is useful in protecting your Play 2 application against Cross-Site Request Forgery (CSRF) attacks. It adds a hidden parameter with a signed uuid on every form post. The signature is stored in a cookie session, and when the form is submitted, along with the rest of the form inputs, it gets the uuid and the signature. If the session.sign is equal to the uuid.sign then the validation passes.

Website: https://github.com/orefalo/play2-authenticitytoken

Deadbolt 2:

Deadbolt gives you the option to add an authorization mechanism to your Play application restricting access to certain controller methods or what users can or cannot view. This allows you to define the requirements for access to each resource and which user groups are allowed to view and edit them. Deadbolt includes template-level restrictions that rely on template tags to hide or show DOM elements. Although this won't increase the security of the code on the server side, it will give you a cleaner UI and controller-level restrictions and also will allow you to customize the response to unauthorized access.

Website: http://deadbolt.ws

Play-pac4j:

This authentication library provides six different authentication mechanisms on the client side. You have to extend your Application from the ScalaController trait and define the clients you need to support. You can get the profile of the user with getUserProfile and protect actions using the RequiresAuthentication function included in Play-pac4j.

Website: https://github.com/leleuj/play-pac4j

Scala-oauth2-provider:

This framework will give OAuth 2.0 server-side functionality to your Play 2 app. You'll have to implement the DataHandler trait and use it with a User class of your own. This DataHandler will return an AuthInfo. To work with Play, you need to define a controller with the OAuth2Provider to issue an access token, assign a route to the controller, and access to an authorized resource.

Website: https://github.com/nulab/scala-oauth2-provider

SecureSocial:

This is an authentication module that can be used to add OAuth, OAuth2, OpenID, Username/Password, and custom

authentication schemes to your application. Its modular design allows you to include only the plugins that you need, or even change them if it's necessary for your project. With the SecuredAction class, you can check on each request and if there is an authenticated user. If so, then the code is invoked. If there isn't an authenticated user, it'll redirect to a login page or send an unauthorized error if it was an ajax call. It's also possible to add authorization logic to the controllers using the Authorization object that will check if an authenticated user is allowed to perform a certain action.

Website: http://www.securesocial.ws

Silhouette:

Silhouette is authentication library forked from SecureSocial that supports several authentication methods including OAuth1, OAuth2, OpenID, Credentials or custom authentication schemes. It needs a configured Environment composed by an IdentityService, an AuthenticatorService, a RequestProvider, and an EventBus implementation. From this, you must implement the IdentityService extending the trait IdentityService. The AuthenticatorService will manage the authentication, verifying the user's session. For the RequestProvider, you can use one of the implementations of the services to identify the user that Silhouette includes, or you can implement your own. Finally, the EventBus, based on the Akka's Event Bus, will handle the events in your app.

With your Environment already configured, you'll be able to use SecuredAction to restrict the access to actions that should be executed only by certain authenticated users.

Silhouette follows the best practices described by the OWASP Authentication Cheat Sheet such as Password Strength Controls, SSL Client Authentication, and use of authentication protocols that require no password.

Website: http://silhouette.mohiva.com

Stateless client authentication:

With this module, you can add authentication to your app without storing any state on the server side. For maintaining the session information, it stores the data on a cookie on the client side and to prevent tampering, it cryptographically signs them. To use it, you'll have to extend the StatelessSecurityBase and define a User model. After doing this, you'll be able to use the StatelessSecurity in your controllers and actions and the currentUser val in your views. As is indicated in the website of the project, this is only for development purposes because it still has issues that restricts it's use in production.

Website: https://github.com/blendlabs/play20-stateless-auth

Lift:

Lift is a web framework that you can use to build your applications with a special focus on security.

The Seven Things that make Lift one of the most important web frameworks currently available are as follows:

- Lazy Loading.
- Parallel page rendering.
- Comet and Ajax.
- Wiring - declare interdependencies between page elements.
- Designer-friendly templates.
- Wizard - multipage input screens with full back-button support.
- Security.

It includes safeguards to prevent many of the OWASP Top 10 vulnerabilities like Injection, Cross Site Scripting (XSS), Broken Authentication and Session Management, Insecure Direct Object References, Cross Site Request Forgery (CSRF), and Failure to Restrict URL Access. You can read how Lift fights these vulnerabilities in Seven Things - Security.

Website: http://liftweb.net

Akka:

Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM. Akka is designed to work in a distributed setting and uses the actor model. Actors are objects with state and behavior that communicate passing messages asynchronously and can be configured to enable location transparency.

The Akka Remoting module allows you to connect and communicate actor systems in peer-to-peer mode but it has some limitations working for client-server setups.

The security between remote nodes can be increased in two ways:

- Untrusted Mode, where you can enable the possibility of ignoring the following operations:
  - remote deployment, including no remote supervision
  - remote DeathWatch
  - system.stop(), PoisonPill, Kill
  - sending any message which extends from the PossiblyHarmful marker interface, including Terminated
  - messages sent with actor selection, unless destination defined in trusted-selection-paths.

- Security Cookie Handshake, with a secure cookie that can be generated cryptographically and that will be exchanged, and that has to be identical in the client and in the server to allow the connection to establish.

- SSL, implemented with Java Secure Socket Extension

(You can read more about this in the Remote Security section of Akka Remoting)

Akka HTTP is an experimental module for interacting with web services and clients. It is used to expose Actors to the web via HTTP and allows them to act as a client consuming HTTP services.

Currently, Akka HTTP only implements the Basic' HTTP Authentication Scheme but provides some predefined SecurityDirectives for authentication and authorization.

Website: http://akka.io

Akka-http-sessions

This project can be used to manage sessions with akka-http. Its most important features are the following:

- Client-side sessions - with the session data saved as a cookie signed by the server and stored on the client. It's possible to encrypt the session and establish a session timeout.
- CSRF protection - implemented assuming that GET requests are non-mutating, verifying requests using double-submit cookies, setting the token in a custom header, and generating a new token on the first GET request that doesn't have the token cookie set.
- Remember me cookies - used to implement persistent sessions. It stores the token hashes and a selector value used to look up those stored hashes and, to prevent timing attacks, compares tokens using a special constant-time comparison method. It allows the use of expiring client sessions apart from the "remember me" session. Finally, it's possible to create conditional persistent sessions depending on user choice.
- Customizing cookie parameters - calling the appropriate methods on the `SessionConfig`. It is recommended to set the `secure` attribute of the cookies and that all sites use HTTPS.
- Creating a SessionConfig using Typesafe config
- Custom cryptography - providing a custom `Crypto` implementation to change the signing algorithm, or the session data encryption/decryption code.

Website: https://github.com/softwaremill/akka-http-session

Akka-http-extensions

With Akka-http-extensions you can easily add some useful directives and utils for your akka-http projects. Including:

- Registration - using bcrypt to create password hashes.
- Authentication - storing the tokens in a cookie.
- Authorization - using the `allow` directive to restrict the access by users.
- P-JAX - adding the P-JAX header to your ajax requests, it will allow to the backend to render the whole html page if there is no P-JAX header or only a specific template if it is an AJAX request.
- UI controls - the Binding-controls subproject (independent from akka-http-extension) includes some predefined views you can use for login, for example.

Website: http://akka.io/community/

Spray:

This is a toolkit for building REST/HTTP-based integration layers on top of Scala and Akka. It's asynchronous, actor-based, fast, lightweight, modular, and testable.

Spray includes SecurityDirectives (update - Spray is no longer maintained) that you can use to add authentication and authorization to your application.

- Authenticate works using a `Future[Authentication[T]]` or a `ContextAuthenticator[T] = RequestContext ⇒ Future[Authentication[T]]` which will return a value of type Authentication[T] that will be the authenticated principal. This will be supplied to the inner route or a rejection if authentication fails. As both of these return a `Future` longer-running authentication tasks can be run without blocking the HttpService actor. The `BasicHttpAuthenticator` implements HTTP Basic Authentication with `BasicAuth()`, although this basic authentication should be used only over SSL because credentials are transferred in plaintext.
- Authorization defined by user is provided with a `⇒ Boolean` value or a `RequestContext ⇒ Boolean` function, if it returns true, the request is passed on to the inner route, otherwise an `AuthorizationFailedRejection` is created, triggering a 403 Forbidden response by default.

Website: http://spray.io/

Conclusion:

As you can see, there are many frameworks that can help you to improve the security of your applications in many different aspects and, nowadays, this is a must have.

This is only a list of the tools we have found with an introduction to them from a security point of view. You can learn more about each one on their own website. And, of course, if you know any others that we have missed, please let us know in the comments section below.

Combining data from a database and a web service with Fetch

scala , http4s , fetch , doobie

Amar Patel joins 47 Degrees as UK's Business Development Director

team , news

GitHub4s 0.10.0 Released

scala , open source , github4s

General

North America

321 3rd Ave S.
Suite 205
Seattle, WA 98104
United States

United Kingdom

41 Corsham Street
London N1 6DR
United Kingdom

Spain

Heroes de Baleares 2
Entreplanta - 11100
San Fernando, Cadiz, España

JOIN OUR NEWSLETTER