Java School

Web Security – Part I

Basics

T-Systems

Voronezh, 2018

Что такое Security

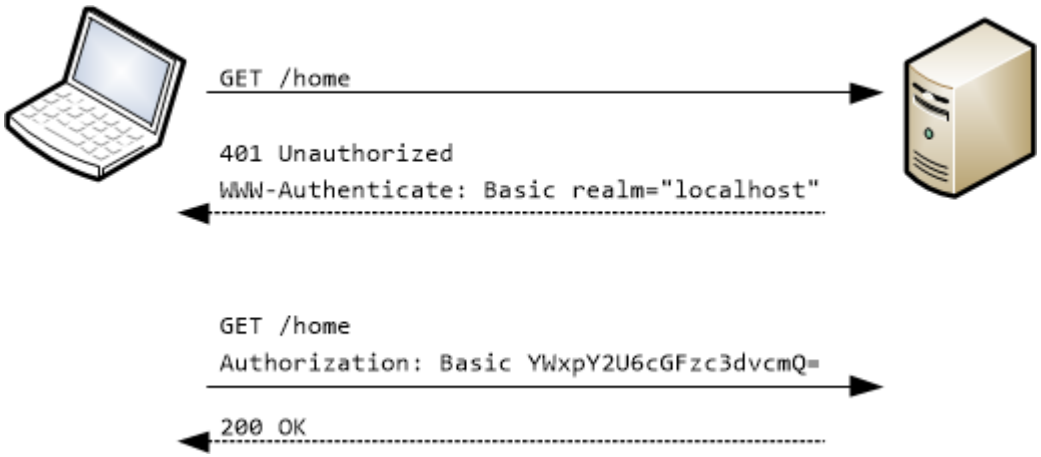
Информационная безопасность (англ. Information *Security*, а также — англ. *InfoSec*) — практика предотвращения несанкционированного доступа, использования, раскрытия, искажения, изменения, исследования, записи или уничтожения информации.

Основные понятия

- **Авториза́ция** (англ. *authorization* «разрешение; уполномочивание») — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.
- **Аутентифика́ция** (англ. *Authentication* — процедура проверки подлинности, например:
  - проверка подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных пользовательских логинов;
  - подтверждение подлинности электронного письма путём проверки цифровой подписи письма по открытому ключу отправителя;
  - проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла.
- **Идентифика́ция** в информационных системах — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно идентифицирующий этого субъекта в информационной системе.

HTTP Basic Authentication (Обычная проверка подлинности)

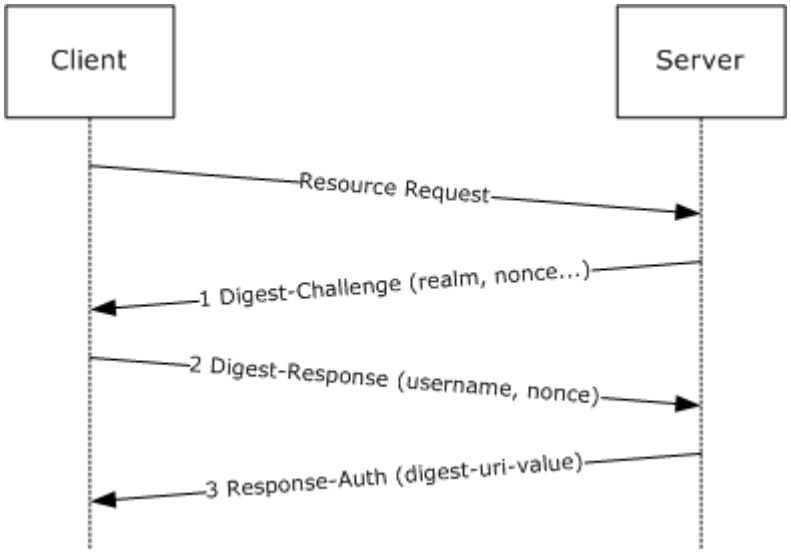
- Клиент обращается к защищенному ресурсу
- Так как запрос требует проверки подлинности, сервер возвращает 401 (не санкционировано). Ответ включает заголовок WWW-Authenticate, указывающий, что сервер поддерживает обычную проверку подлинности.
- Клиент отправляет другой запрос, с помощью учетных данных клиента в заголовке авторизации. Учетные данные форматируются в виде строки «login: password», с кодировкой base64. **Учетные данные не шифруются**.



T · · Systems

HTTP Digest Authentication (Аутентификация по дайджесту)

- Технически, аутентификация по дайджесту представляет собой применение криптографической хеш-функции MD5 к секрету пользователя с использованием случайных значений для затруднения криптоанализа и предотвращения replay-атак. Работает на уровне протокола HTTP.

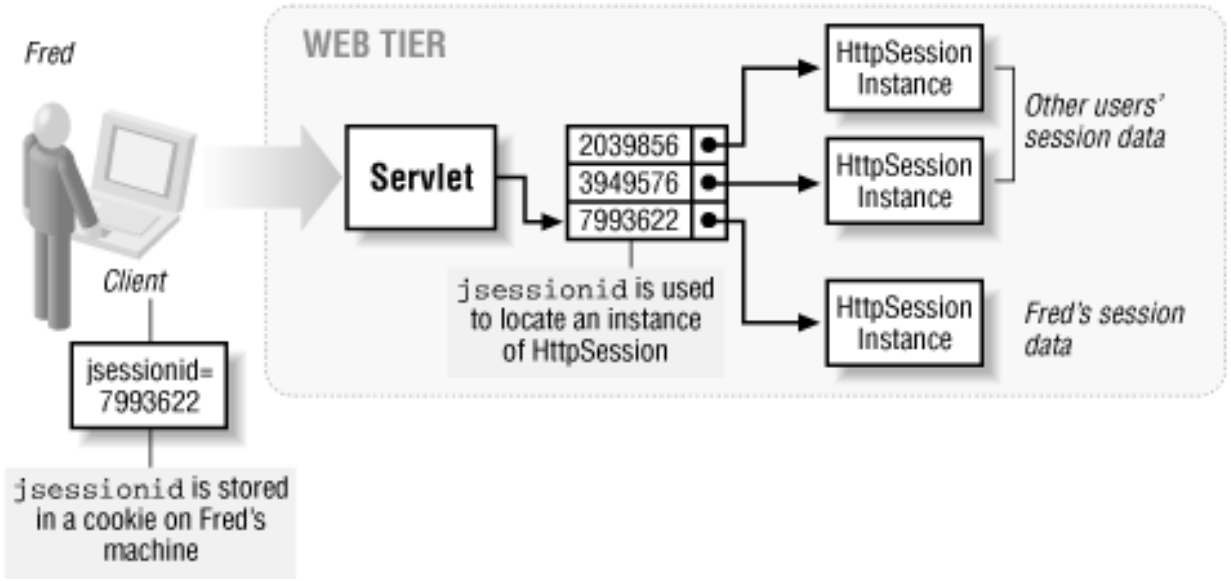


HTTP Basic и Digest в Tomcat/Java Servlet API

- **Realm** - «база данных» логинов и паролей для идентификации пользователей одного или нескольких веб-приложений. Каждому пользователю также соответствует список «ролей» (или иначе – групп).
- Самый простой **Realm** в tomcat – файл *tomcat-users.xml*. Поддерживаются также *JDBCRealm*, *DataSourceRealm*, *JNDIRealm* и другие.
- Пример web.xml для Basic Http Authentication.
- С HTTP-запросом прошедшим авторизацию можно выполнять:
  - *getRemoteUser* – возвращает имя пользователя
  - *getUserPrincipal* – возвращает экземпляр javax.security.Principal
  - *isUserInRole(String role)* – проверяет отношение текущего пользователя к заданной роли (группе)

Web-сессии в Java Servlets API

- Протокол HTTP не имеет сессии, при каждом запросе от клиента создаётся отдельное соединение. Сервер не хранит никакой информации о предыдущих запросах от данного клиента.
- Для создания и поддержки сессии как правило используются Cookie.
- В Java Servlet API таким Cookie выступает JSESSIONID.
- К текущей сессии можно обратится через *httpRequest.getSession*
- С сессией можно ассоциировать пользовательские данные в атрибутах



T · · Systems

Виды и атрибуты Cookie

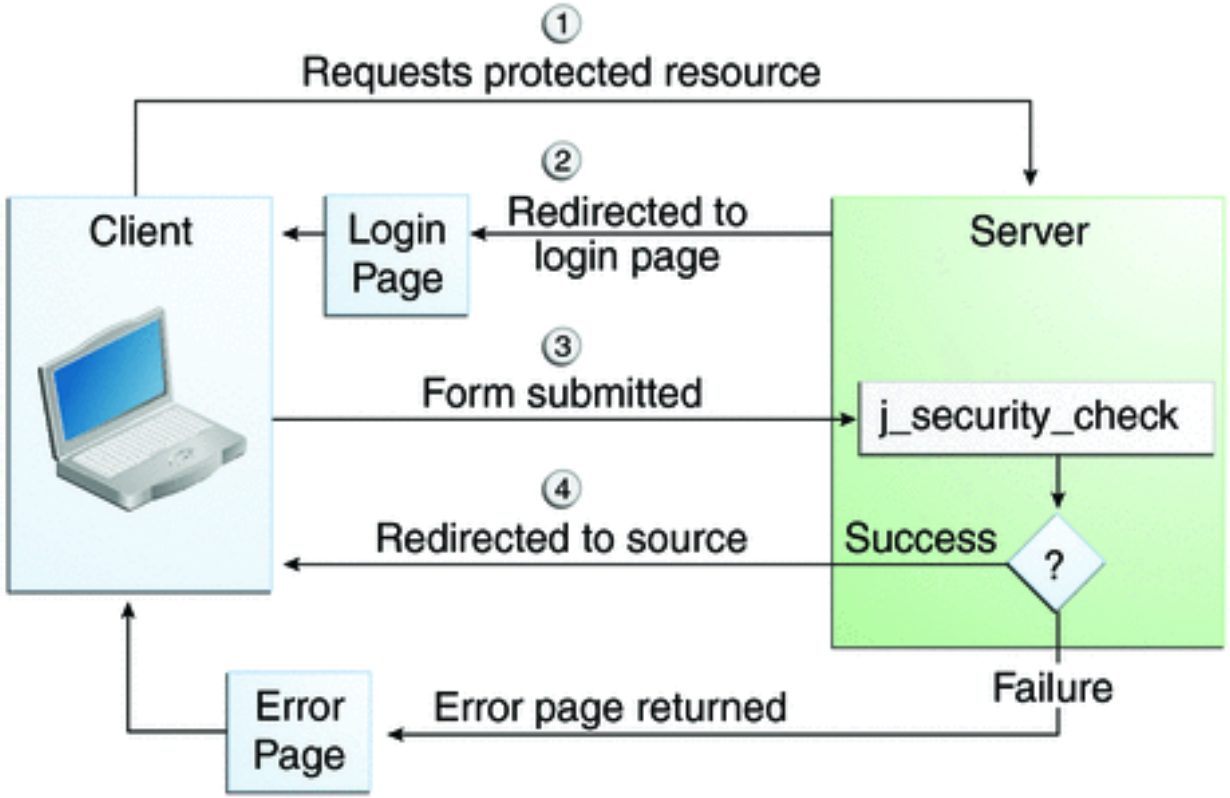
- **Сессионные куки.** Существуют только во временной памяти, пока пользователь находится на странице веб-сайта. Браузеры обычно удаляют сессионные куки после того, как пользователь закрывает окно браузера.
- **Постоянные куки.** Удаляются в определённую дату или через определённый промежуток времени. Это означает, что информация о куки будет передаваться на сервер каждый раз, когда пользователь посещает веб-сайт, которому эти куки принадлежат.
- **Защищенные куки**. Могут быть переданы только через шифрованное соединение (то есть HTTPS). Они не могут передаваться по незащищенным соединениям.
- **HttpOnly-куки.** К файлу HttpOnly-куки нельзя обращаться с помощью API на стороне клиента, таких как JavaScript.
- **Атрибуты куки:**
  - Домен (Domain)
  - Путь (Path)

```
Set-Cookie: attribute=myvalue; expires=Fri, 31 Dec 2010 23:59:59 GMT; path=/;
domain=.example.net
```

T · · Systems

Аутентификация на основе формы

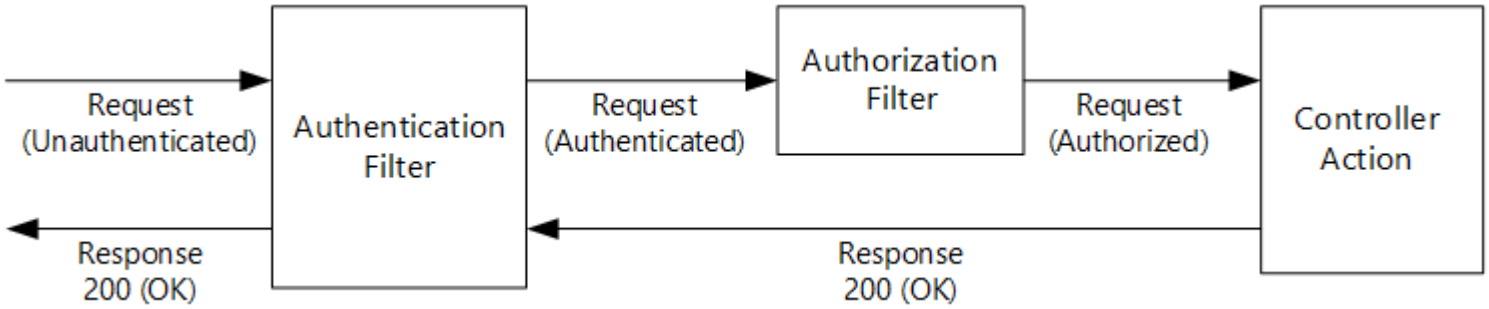
- Позволяет выполнить форму для авторизации в общем стиле (дизайне) приложения
- Позволяет выполнить автоматическое перенаправление (redirect) с формы авторизации на защищенный ресурс



T · · Systems

Создание своих схем авторизации и аутентификации

- При желании можно реализовать свою схему проверки подлинности и разграничения прав доступа на основе Servlet Filters.
- Например, реализовать аутентификацию на основе «токена».
- Каждый HTTP запрос будет попадать сначала в обработчик фильтра.
- Фильтр производит действия аутентификации и авторизации. При успешном итоге передает обработку далее по цепочке. При неуспешном – прерывает выполнение возвращая коды 401 Unauthorized или 403 Forbidden.



Аутентификация на основе токена (JWT)

- **JSON Web Token** (**JWT**) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON.
- Токен JWT состоит из трех частей: заголовок (header), полезная нагрузка (payload) и подпись или данные шифрования.
- Access-токен — это токен, который предоставляет доступ его владельцу к защищенным ресурсам сервера. Обычно он имеет короткий срок жизни.
- Refresh-токен — это токен, позволяющий клиентам запрашивать новые access-токены по истечении их времени жизни. Данные токены обычно выдаются на длительный срок.

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5c
CI6IkpXVCJ9.eyJzdWIiOiIxM
jM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiYWRtaW4iOnR
ydWV9.TJVA95OrM7E2cBab30R
MHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded

```
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Header

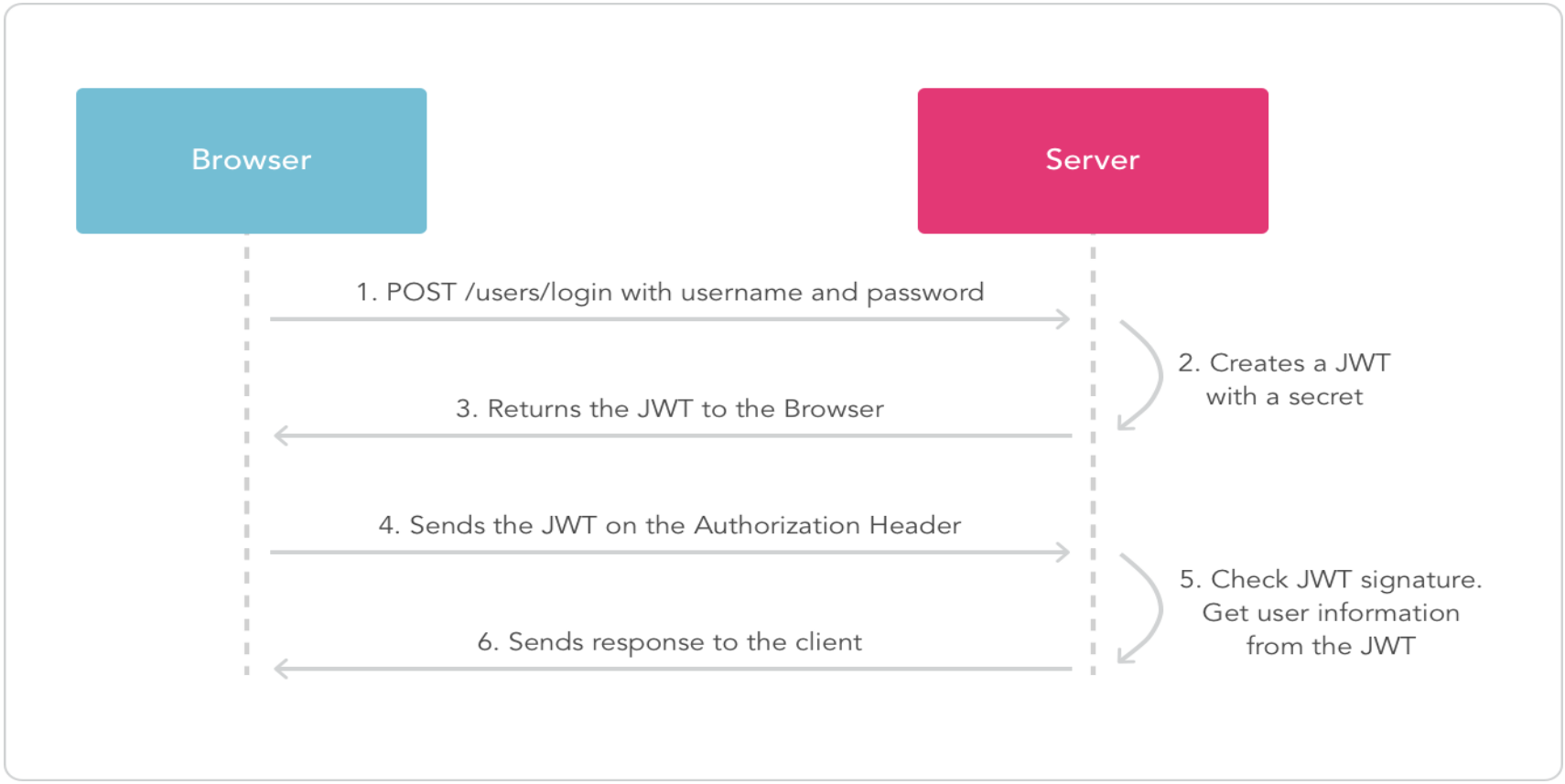
Payload

Signature

T · · Systems

Процесс аутентификации и авторизации на основе JWT

Диаграмма взаимодействия клиента и сервера при использовании JWT-токена



T · · Systems

Правовая основа

- Статья 272 УК РФ. Неправомерный доступ к компьютерной информации
- Статья 273 УК РФ. Создание, использование и распространение вредоносных программ для ЭВМ
- Статья 274 УК РФ. Нарушение правил эксплуатации средств хранения, обработки или передачи компьютерной информации и информационно-телекоммуникационных сетей

OWASP Top 10

- Open Web Application Security Project (OWASP) — это открытый проект обеспечения безопасности веб-приложений. Сообщество OWASP включает в себя корпорации, образовательные организации и частных лиц со всего мира. Сообщество работает над созданием статей, учебных пособий, документации, инструментов и технологий, находящихся в свободном доступе.
- OWASP Top 10 – список наиболее актуальных угроз составленный и поддерживаемый организацией.

OWASP

Open Web Application Security Project

T · · Systems

Уязвимости веб приложения

- Данные передаваемые по нешифрованному (HTTP) каналу могут быть перехвачены.
- Передача конфиденциальных данных в GET-запросах небезопасна из-за логирования таких запросов. Более подходит запрос POST.
- XSS. Межсайтовый скриптинг. Внедрение вредоносного скрипта в код страницы. Тем или иным образом. Простейший пример – ошибка экранирования.

```
http://example.com/search.php?q=<script>DoSomething();</script>
```

- CSRF (*Cross Site Request Forgery* — «межсайтовая подделка запроса»). С вредоносного сайта выполняется запрос на другой сайт, где пользователь уже авторизован (и имеет куки, например).

```
Боб: Привет, Алиса! Посмотри, какой милый котик:
<img
src="http://bank.example.com/withdraw?account=Alice&amount=1000000&for=Bob">
```

Broken Access Control

- Ошибки разграничения доступа к чувствительным ресурсам
- Неправильная организация ограничения доступа к данным

Insufficient Attack Protection

- Отсутствие средств:
  - Обнаружения вторжений
  - Предотвращения вторжений
  - Логгирования
- Невозможность быстрого закрытия обнаруженных уязвимостей

Криптография. Шифрование. Виды алгоритмов.

- Ассиметричные алгоритмы – для шифрования и расшифрования используются разные ключи (закрытый и открытый соотвественно). При этом из закрытого ключа можно получить открытый но не наоборот.
  - RSA (Rivest-Shamir-Adleman)
  - Diffie-Hellman (Обмен ключами Диффи - Хелмана)
  - ECDSA (Elliptic Curve Digital Signature Algorithm) - алгоритм с открытым ключом для создания цифровой подписи.
  - ГОСТ Р 34.10-2012
- Симметричные алгоритмы – для шифрования и расшифрования используется один и тот же ключ.
  - AES (Advanced Encryption Standard) - американский стандарт шифрования
  - ГОСТ 28147-89 - советский и российский стандарт шифрования
  - DES (англ. Data Encryption Standard) - стандарт шифрования данных в Европе и США (3DES (Triple-DES))

Криптография. Хэширование

- Требования:
  - Необратимость или стойкость к восстановлению прообраза
  - Стойкость к коллизиям первого рода или восстановлению вторых прообразов: для заданного сообщения M должно быть вычислительно невозможно подобрать другое сообщение N, для которого H(N)=H(M)
  - Стойкость к коллизиям второго рода: должно быть вычислительно невозможно подобрать пару сообщений (M,M') имеющих одинаковый хеш
- Виды:
  - Ключевая
  - Бесключевая

Дополнительная информация

- Общая картина: The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws
- Механизмы защиты браузера: The Tangled Web: A Guide to Securing Modern Web Applications
- Полигон для практических занятий:
  - OWASP Vulnerable Web Applications Directory Project
  - google gruyere
  - Metasploitable
- Каталоги уязвимостей:
  - https://cve.mitre.org
  - http://www.securityfocus.com

T · · Systems

Java School

Web Security: Part 2

Spring Security

T · · Systems

Spring security

- Spring security – фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для Spring-based приложений.
- Безопасность приложения базируется на двух основных проблемах:
  - Authentication (who you are?)
  - Authorization (what are you allowed to do?)
- Spring security имеет архитектуру которая разработана таким образом что отделяет процесс аутентификации от авторизации и позволяет расширять их.

Ключевые объекты контекста Spring Security:

- **SecurityContextHolder**, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением.
- **SecurityContext**, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.
- **Authentication** представляет пользователя (Principal) с точки зрения Spring Security.
- **GrantedAuthority** отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE_ANONYMOUS, ROLE_USER, ROLE_ADMIN.
- **UserDetails** предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности.
- **UserDetailsService**, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса:

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

Authentication

- Spring security поддерживает много способов аутентификации, таких как:
  - Simple Form-Based
  - HTTP Basic and Digest
  - LDAP
  - OpenID
  - etc.

- Основной интерфейс для аутентификации **AuthenticationManager**
- Имеет имплементацию **ProviderManager,** который делегирует процесс цепочке объектов имплементирующих **AuthenticationProvider**

Customizing Authentication Managers

- **AuthenticationManagerBuilder**

Часто используемый конфиг-хелпер для быстрой настройки in-memory, JDBC или LDAP user-details.

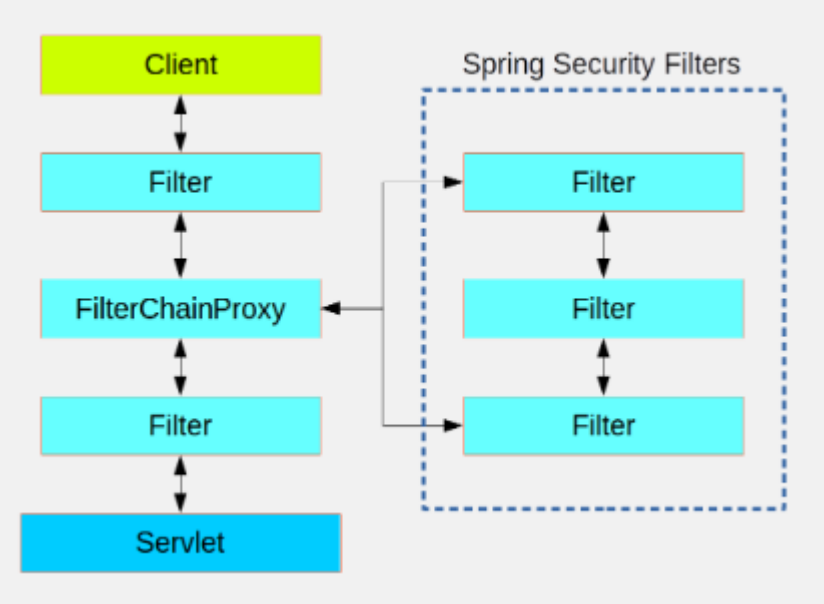
Пример настройки in-memory Authentication:

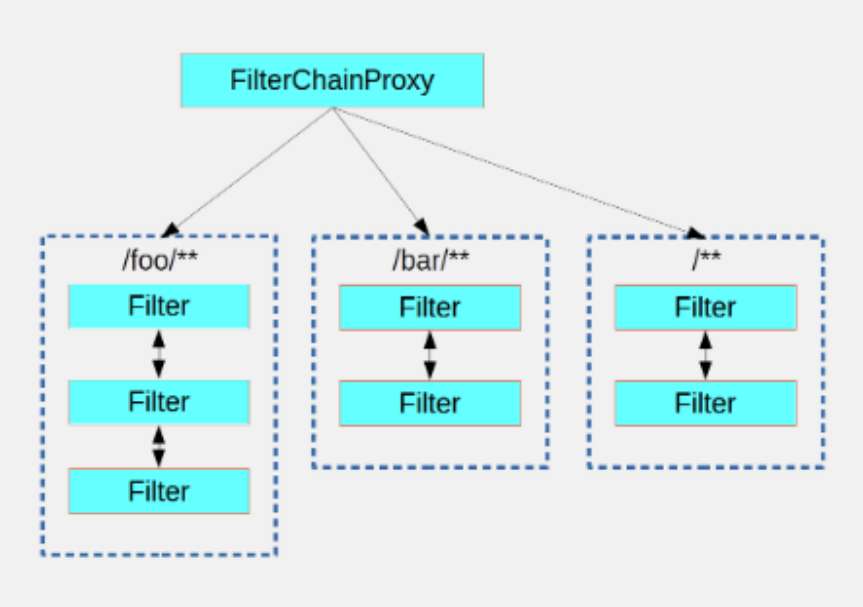
```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
        …
        @Autowired
        public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
                auth.inMemoryAuthentication().withUser(“root").password(“root").roles("ADMIN");
                auth.inMemoryAuthentication().withUser(“user").password(“user").roles("USER");
        }
}
```

T · · Systems

Web Security

- Spring security в контексте веб-приложения основано на фильтрах сервлета.





T · · Systems

Spring Method Security

- ***@Secured***, ***@RoleAllowed (*****JSR-250's equivalent*****)**
- ***@PreAuthorize***
- ***@PostAuthorize***
- ***@PreFilter***
- ***@PostFilter***

@Secured Annotation

- @Secured используется для определения роли или списка ролей каким доступен вызов метода

```
@Secured("ROLE_VIEWER")
public String getUsername() {
    SecurityContext securityContext = SecurityContextHolder.getContext();
    return securityContext.getAuthentication().getName();
}


@Secured({ "ROLE_VIEWER", "ROLE_EDITOR" })
public boolean isValidUsername(String username) {
    return userRoleRepository.isValidUsername(username);
}
```

@PreAuthorize and *@PostAuthorize* Annotations

- @PreAuthorize и @PostAuthorize аннотации предоставляют управление доступом на основе выражений (SpEL (Spring Expression Language)).
- @PreAuthorize проверяет выражение перед вызовом метода

```
@PreAuthorize("hasRole('ROLE_VIEWER')")
public String getUsernameInUpperCase() {
    return getUsername().toUpperCase();
}
```

Можно использовать аргументы метода в выражении

```
@PreAuthorize("#username == authentication.principal.username")
public String getMyRoles(String username) {
    //...
}
```

- @PostAuthorize проверяет выражение после того как метод выполнен

```
@PostAuthorize
  ("returnObject.username == authentication.principal.nickName")
public CustomUser loadUserDetail(String username) {
    return userRoleRepository.loadUserByUserName(username);
}
```

T · · Systems

@PreFilter and *@PostFilter* Annotations

- @PreFilter позволяет фильтровать аргумент коллекцию перед выполнением метода

```
@PreFilter("filterObject != authentication.principal.username")
public String joinUsernames(List<String> usernames) {
    return usernames.stream().collect(Collectors.joining(";"));
}
```

- @PostFilter Фильтрует возвращаемую методом коллекцию

```
@PostFilter("filterObject != authentication.principal.username")
public List<String> getAllUsernamesExceptCurrent() {
    return userRoleRepository.getAllUsernames();
}
```

Spring Method Security

- Можно аннотировать весь класс:

```
@Service
@PreAuthorize("hasRole('ROLE_ADMIN')")
public class SystemService {

    public String getSystemYear(){
        //...
    }

    public String getSystemDate(){
        //...
    }
}
```

- Можно использовать несколько аннотаций сразу

```
@PreAuthorize("#username == authentication.principal.username")
@PostAuthorize("returnObject.username == authentication.principal.nickName")
public CustomUser securedLoadUserDetail(String username) {
    return userRoleRepository.loadUserByUserName(username);
}
```

Spring Method Security

- **Важные замечания:**
  - По умолчанию для method-security используется Spring AOP.

  Если защищенный метод A вызвается другим методам из одного класса то аннотация на методе A игнорируется. Также это применимо к приватным методам

  - ***SecurityContext*** связан с текущим потоком и по умолчанию не распространяется на дочерние потоки

Securing URLs Using Spring Security

Задачи для защиты URL:

- Разрешить доступ всем
- Защитить URL на основании роли и списка ролей
- Защитить URL на основнаии IP адреса

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
      http
        .authorizeRequests()
          .antMatchers("/static","/register").permitAll()
          .antMatchers("/user/**").hasRoles("USER", "ADMIN") // can pass multiple roles
          .antMatchers("/admin/**").access("hasRole('ADMIN') and hasIpAddress('123.123.123.123')") // pass SPEL using
access method
          .anyRequest().authenticated()
          .and()
        .formLogin()
          .loginUrl("/login")
          .permitAll();
    }
```

JSP Tag Library

- Spring security предоставляет JSP tag library для доступа к SecurityContex на страницах JSP
- Declaration in JSP:

```
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
```

- *authorize* tag

```
<sec:authorize access="!isAuthenticated()">
  Login
</sec:authorize>
<sec:authorize access="isAuthenticated()">
  Logout
</sec:authorize>
```

- *authentication* Tag используется для доступа к текущему Authentication оъъекту в SecurityContext

```
<sec:authorize access="isAuthenticated()">
   Welcome Back, <sec:authentication property="name"/>
</sec:authorize>
```