Need More Sleep? REST Could Help

Drew Branch, Security Analyst I
dbranch@securityevaluators.com

Introduction

Drew Branch, CEH
Associate Security Analyst @ ISE
B.S. Electrical/Computer Engineering
M.S. Cybersecurity
Hobbies: Learning about new tech and playing sports

Introduction

ISE

- Based in Baltimore
- High end custom security assessments
- Assess new web technologies

Overview

- REST Background
- REST vs SOAP
- REST Concepts
- Common Security Mistakes
- Hot to Fix

Background

- REpresentational State Transfer (REST)
- Defined by Roy Fielding in 2000
  - Also one of the main contributors of the HTTP specification

REST vs SOAP

REST

- No specification guidelines
- Easier to create documentation
- Scalable
- Makes use of HTTP Methods

SOAP

- Uses SOAP protocol to exchange data
- Bound by SOAP specification
  - Break one requirement, API is not SOAP
- Uses HTTP POST method

REST vs SOAP

Sample Requests

REST Request

```
POST /auth HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 49
Cache-Control: no-cache
Origin:
chrome-extension:
//fhbjgbiflinjbdggehcddcbncdddomop
User-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_11_3)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
Content-Type: application/json
Accept: */*
DNT: 1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

{
  "username":"admin",
  "password":"admin"
}
```

SOAP Request

```
POST /auth HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: nnn
Cache-Control: no-cache

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap=
"http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle=
"http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m=
"http://www.example.org/stock">

  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

HTTP Methods

- GET – Retrieve a resource
- POST – Create a new resource
- DELETE – Delete a resource at specified URI
- PATCH – Modify the resource; not replace
- PUT – Replace resource with a newly-updated representation
  - Can also create resources
- Other methods include: OPTIONS, HEAD, and TRACE

Status Codes

Code	Description	Code	Description
200	OK	401	Unauthorized
201	Created	403	Forbidden
202	Accepted	404	Not Found
400	Bad Request	500	Internal Server Error

Data Format

No rules on the format of data

Q. How does client/server know what format the data is in?

A. Header Content-Type

- text/xml
- application/json

Data Format

Content Negotiation

Accept Header

- Client sends Server header of preferred data type

Server returns data in preferred format or returns error if data type is not supported

Resource URIs

Static resource URI

- www.example.com/blogname
  - Each blog has a static webpage

RESTful URI

- www.example.com/blogs/{blogId}
  - URI stays the same if web application is changed
  - Independent of framework

Resource Relationships

Better to group resources that belong to another resource in a subfolder instead of its own folder

Makes it clear that a resource belongs to a particular resource

Resource Relationships

www.example.com/comments/{commentId}

- Treats both blogs and comments as separate entities
- Loses relationship between a blog and its comments

www.example.com/blogs/{blogId}/comments/{commentId}

- Keeps relationships

HATEOAS

Hypermedia

As

The

Engine

Of

Application

State

HATEOAS

Sample response from server containing URIs to:

- The blog's comments
- The authors profile

GET /blog/1

```
{
    "id": "1",
    "author": "dbr@n",
    "date": "15June2016",
    "commentsUri": "api/blogs/1/comments/",
    "authorProfileUri": "api/profiles/1"
}
```

RESTful API Classification

Is my API fully RESTful? Almost RESTful? Or not at all

Classification is based off of the Richardson Maturity Model

RESTful API Classification

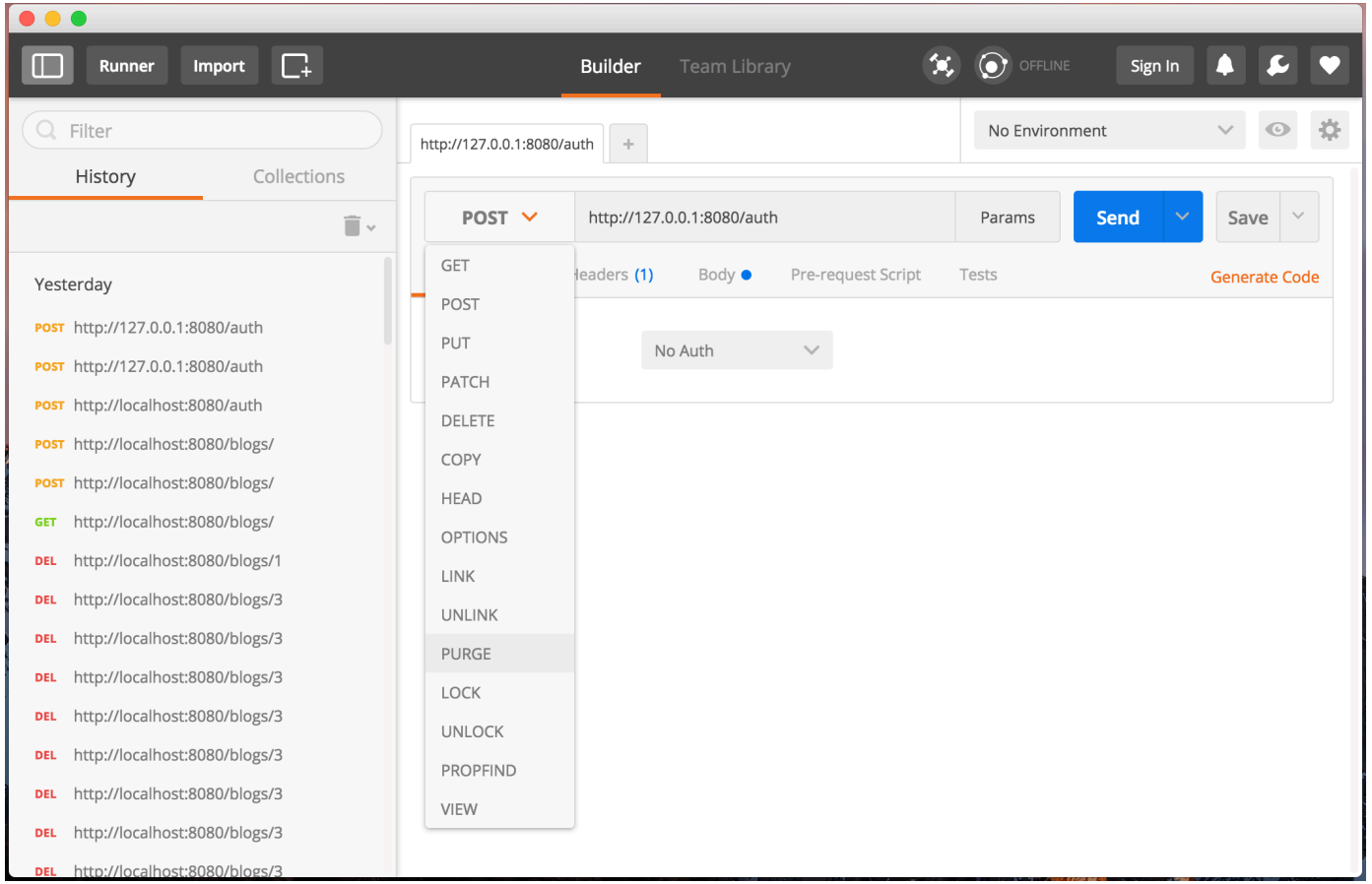
- Level 0 - Not RESTful at all
- Level 1 - Use of resources
  - Individual URI for each resource
    - /profiles/{id}
    - /blogs/{id}
    - /blogs/{id}/comments/{id}
  - Request will still contain operation
- Level 2 – Use of HTTP methods and status codes
- Level 3 - Hypermedia controls (HATEOAS)

Testing RESTful APIs

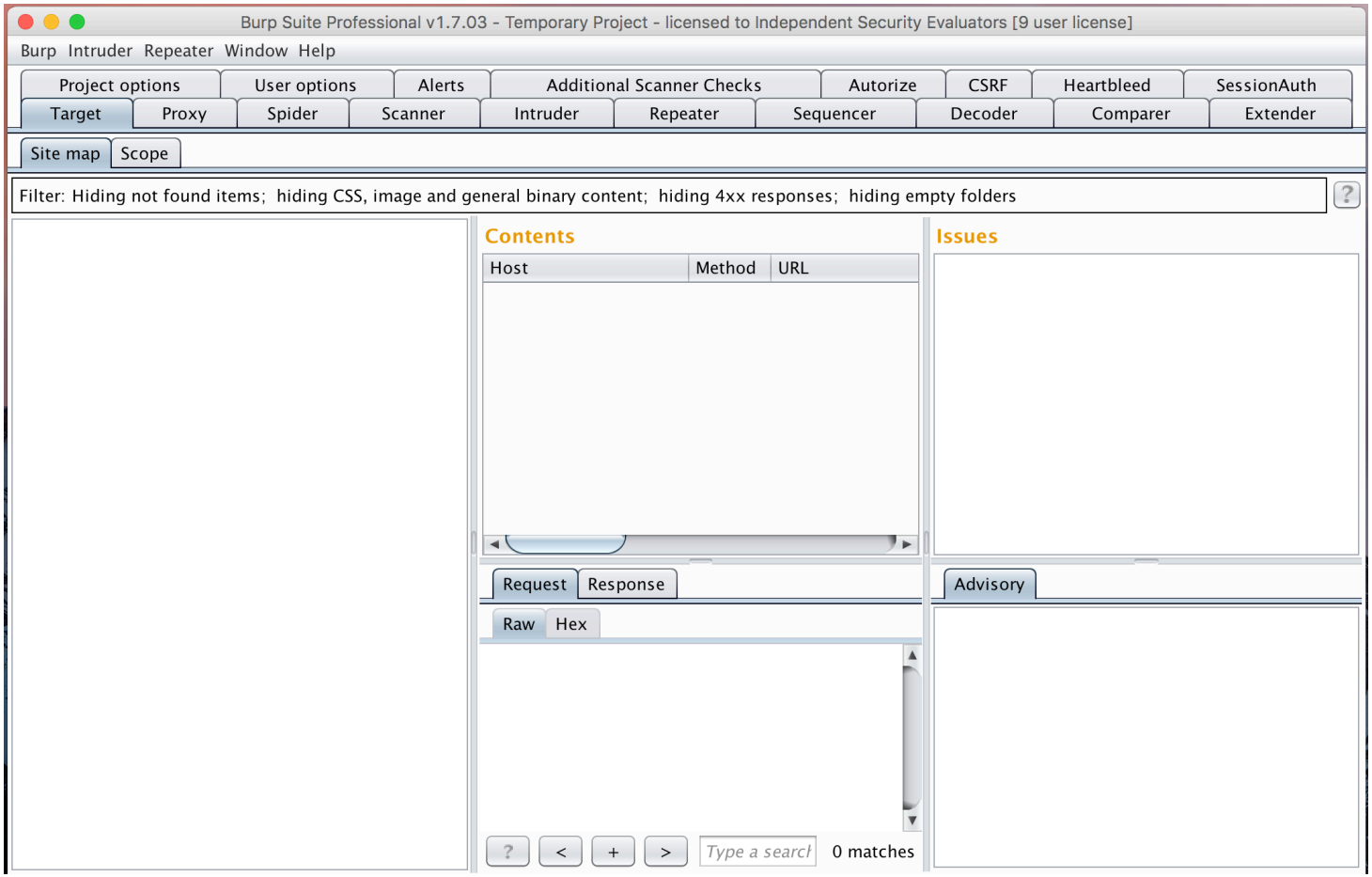
Useful Tools

- Use Postman to test during development
- Use Burp Suite to assess APIs during assessment

Postman


Runner
Import
Builder
Team Library
OFFLINE
Sign In
Filter
History
Collections
Yesterday
POST http://127.0.0.1:8080/auth
POST http://127.0.0.1:8080/auth
POST http://localhost:8080/auth
POST http://localhost:8080/blogs/
POST http://localhost:8080/blogs/
GET http://localhost:8080/blogs/
DEL http://localhost:8080/blogs/1
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
DEL http://localhost:8080/blogs/3
http://127.0.0.1:8080/auth
No Environment
POST
http://127.0.0.1:8080/auth
Params
Send
Save
GET
POST
PUT
PATCH
DELETE
COPY
HEAD
OPTIONS
LINK
UNLINK
PURGE
LOCK
UNLOCK
PROPFIND
VIEW
Headers (1)
Body
Pre-request Script
Tests
Generate Code
No Auth

Burp Suite


Burp Suite Professional v1.7.03 - Temporary Project - licensed to Independent Security Evaluators [9 user license]
Burp Intruder Repeater Window Help
Project options
User options
Alerts
Additional Scanner Checks
Autorize
CSRF
Heartbleed
SessionAuth
Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Site map
Scope
Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders
Contents
Host
Method
URL
Issues
Request
Response
Raw
Hex
Advisory
Type a searcl
0 matches

Security Concerns

- Unauthenticated/Unauthorized modification of “protected” assets
- Unauthenticated/Unauthorized access to “protected” assets
- Replay Attacks

Sample API

Created a simple blog RESTful API

- Eclipse (MARS.1)
- Spring
- Tomcat v8

Protect HTTP Methods

- Not every method is valid for every resource
- Whitelist allowable methods
- Do not allow delete for critical files

Protect HTTP Methods

Annotate endpoints with allowable methods

```
@RequestMapping(path ="/blogs", method = RequestMethod.GET)
public List<Blog> getAllBlogs(){
    return new ArrayList<Blog>(blogs.values());
}
```

Spring

Protect HTTP Methods

Annotate endpoints with allowable methods

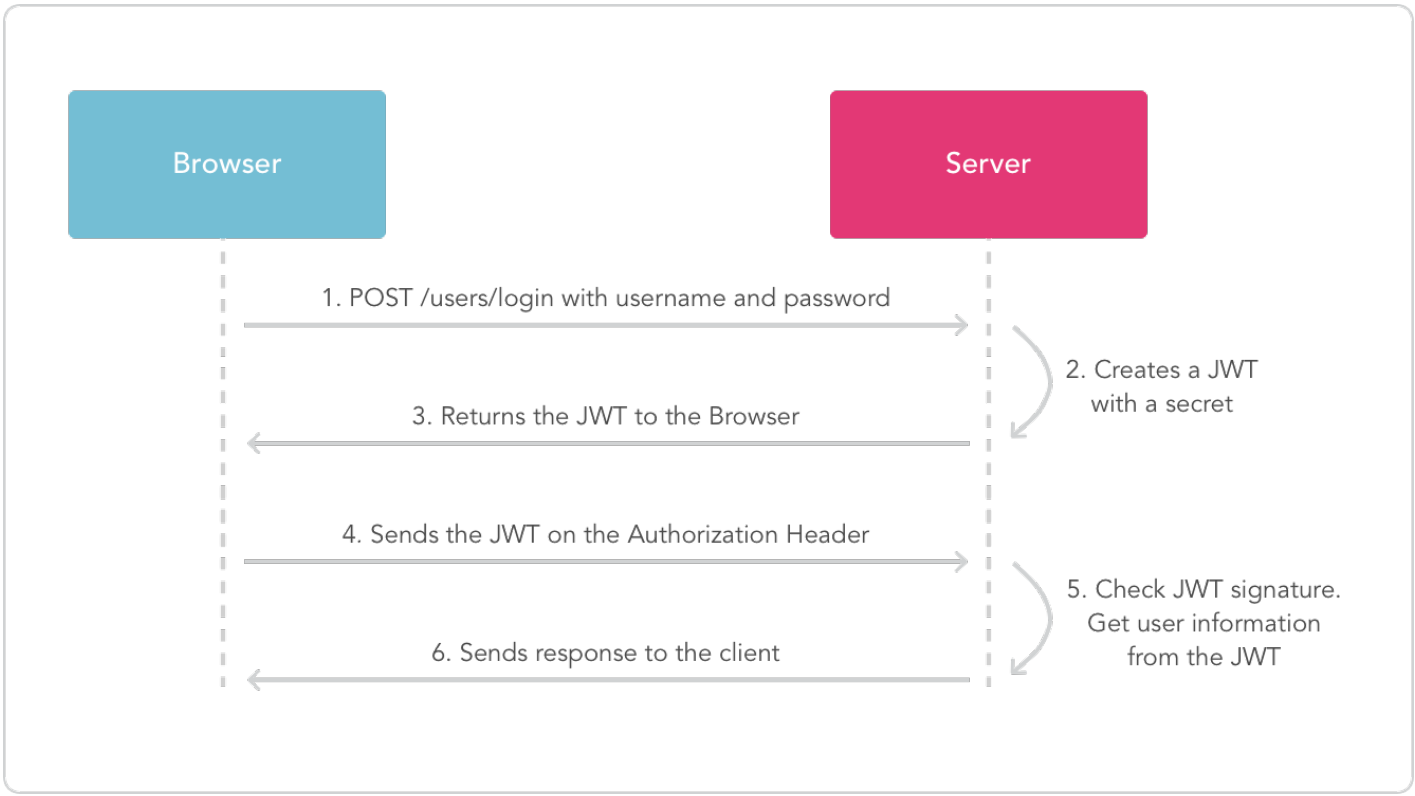
```
@Path("/blogs")
public class BlogResource {

      BlogService blogService = new BlogService();

      @GET
      @Produces(MediaType.APPLICATION_JSON)
      public List<Blog> getBlogs(){
            return blogService.getAllBlogs();
      }
}
```

Jersey

JSON Web Tokens (JWT)

- Used to verify the sender
- Self contained (Stateless)
- In JSON format
- Base64 encoded
- Cannot be secured using HTTP cookie flags
- Sent within request's Authorization header
  - Primarily used for authentication but could be utilized for authorization

JSON Web Tokens (JWT)



Reference: https://jwt.io/introduction/

JSON Web Tokens (JWT)

Made up of three parts:

- Header
- Payload
- Signature

Payload contains claims

- Statement about entity (usually a user)
- Metadata about token

JSON Web Tokens (JWT)

Sample JWT

header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

payload

```
{
  "iss": "dbr@n",
  "sub": "userName",
  "exp": 1426420800
}
```

signature = HS256(base64(header) + “.” + base64(payload))

Format: header.payload.signature

```
eyJhbGcxMiJ9.eyJzdWIiOiJ1XVka9.caf1R0kof9V5bZOl9
```

JSON Web Tokens (JWT)

- Should not contain sensitive information
  - not encrypted only base64 encoded
- JWT could be encrypted using JSON Web Encryption (JWE)
- Secret used to create signature should be secured server-side

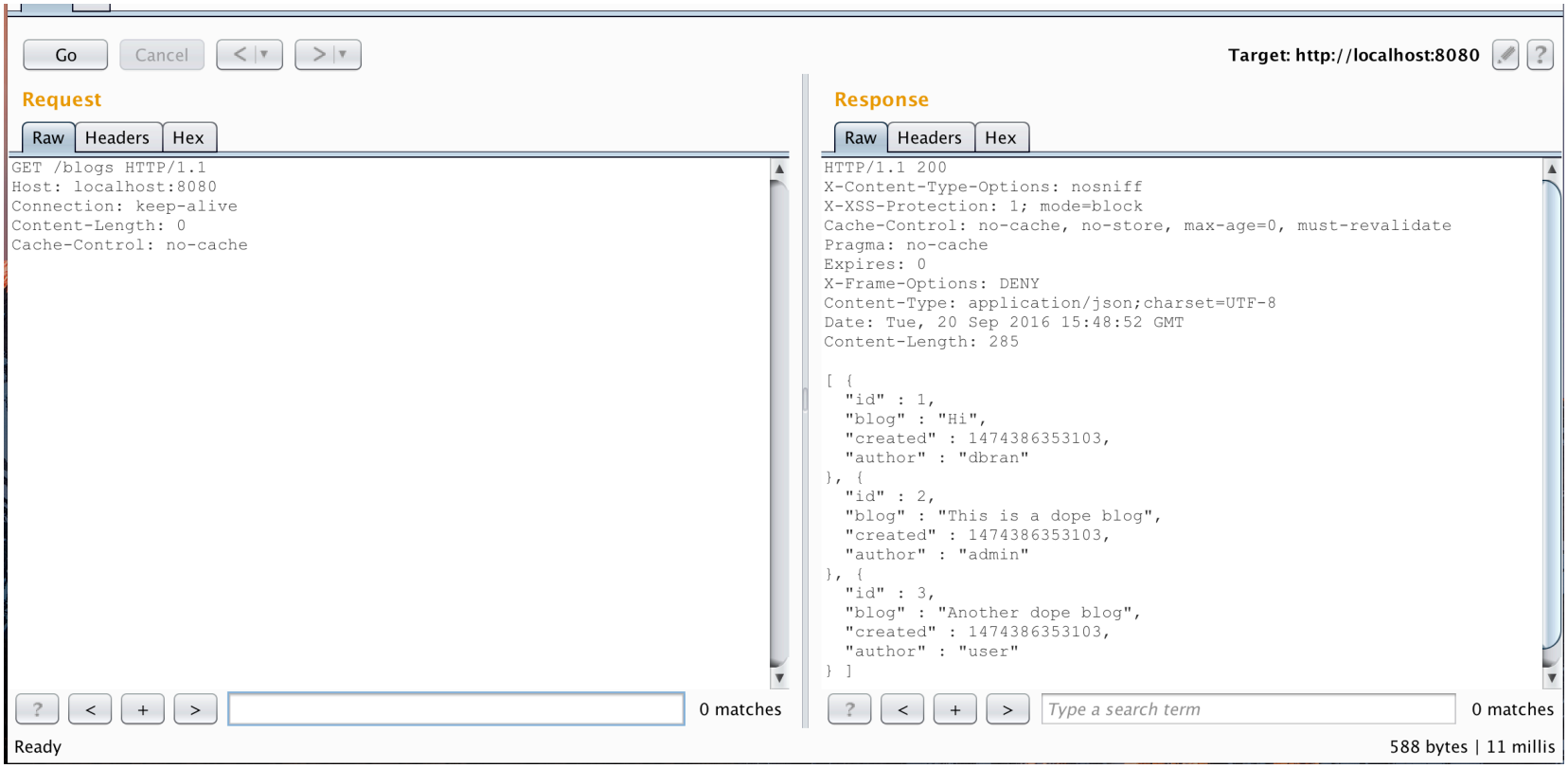
JSON Web Tokens (JWT)

Securing Endpoints with JWT

- Verify signatures
- Set short expiration dates
- Communicate over HTTPS
- Secure signing secret server side
- Do not include sensitive data in JWT
  - If so, use JWE

JSON Web Tokens (JWT)

Securing Endpoints with JWT



JSON Web Tokens (JWT)

Request Filter to intercept all requests and verify JWT token

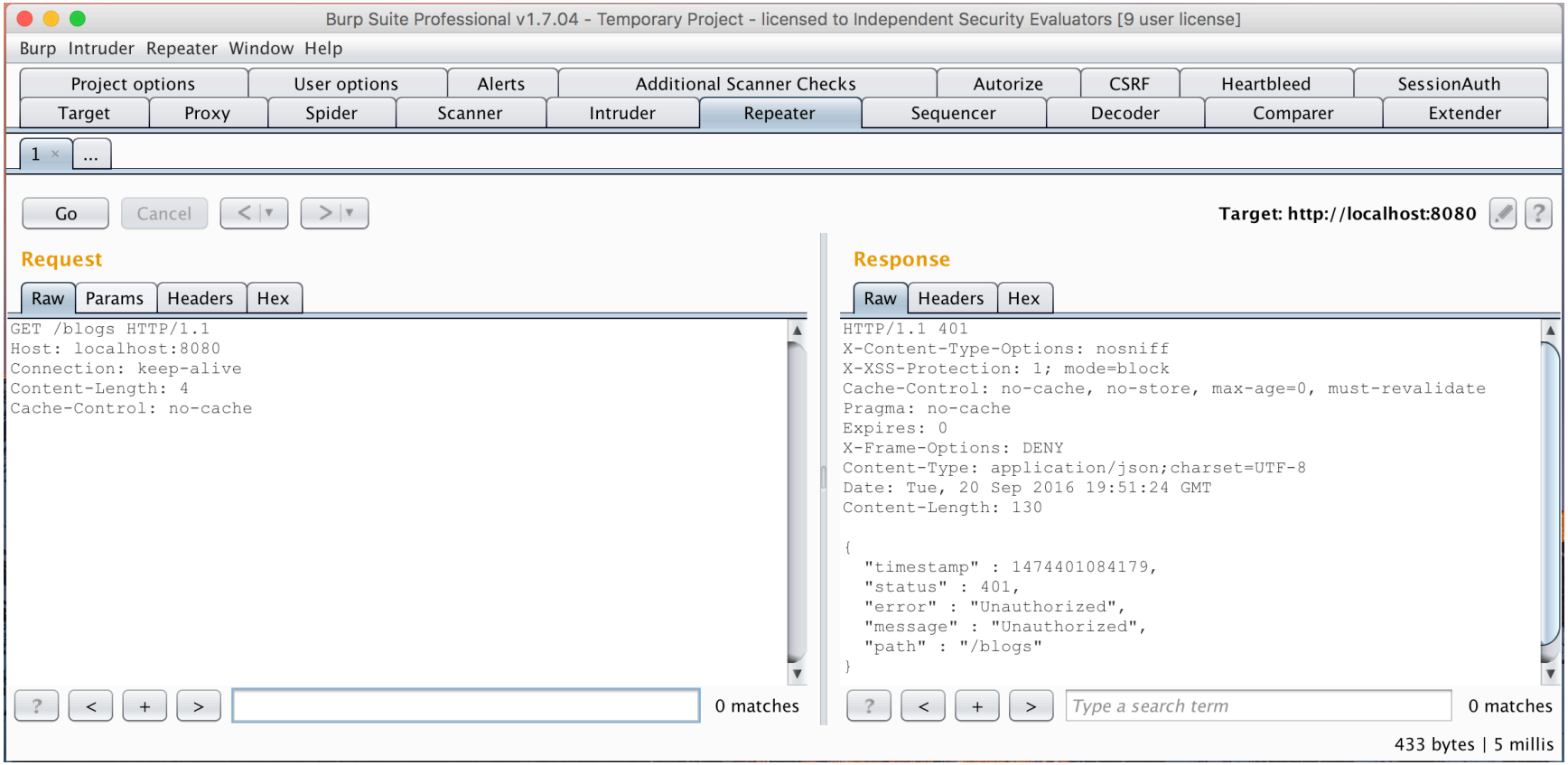
```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {

    HttpServletRequest httpRequest = (HttpServletRequest) request;
    String authToken = httpRequest.getHeader(this.tokenHeader);
    String username = jwtTokenUtil.getUsernameFromToken(authToken);

     if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
          UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);
          if (jwtTokenUtil.validateToken(authToken, userDetails)) {
               UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
               authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(httpRequest));
               SecurityContextHolder.getContext().setAuthentication(authentication);
    }
  }

  chain.doFilter(request, response);
}
```

JSON Web Tokens (JWT)

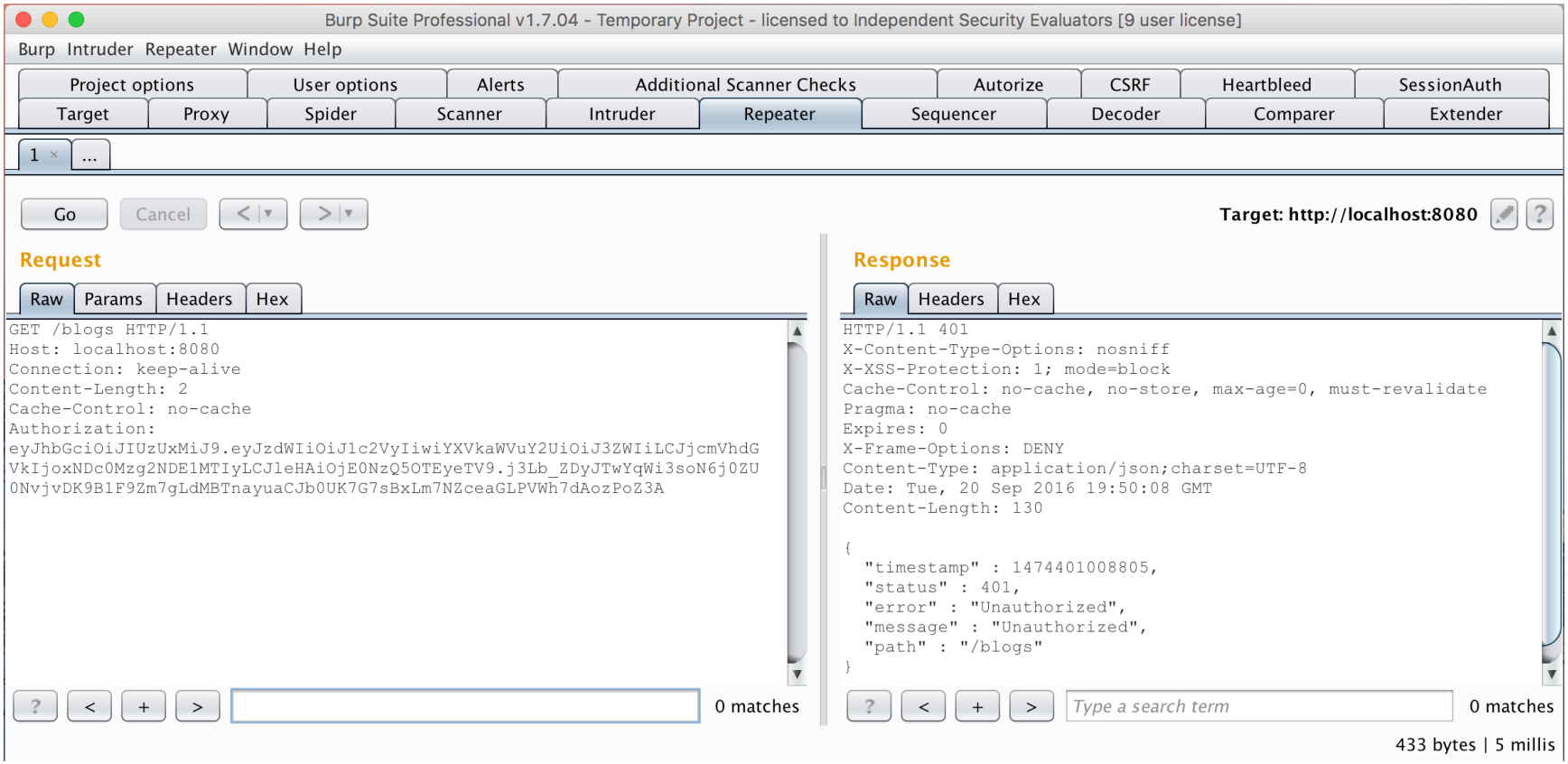
JSON Validate Method

```
public Boolean validateToken(String token, UserDetails userDetails) {
    JwtUser user = (JwtUser) userDetails;
    final String username = getUsernameFromToken(token);
    final Date created = getCreatedDateFromToken(token);

    return (
            username.equals(user.getUsername())
            && !isTokenExpired(token)
            && !isCreatedBeforeLastPasswordReset(created,
                    user.getLastPasswordResetDate()));
    }
```

JSON Web Tokens (JWT)



JSON Web Tokens (JWT)

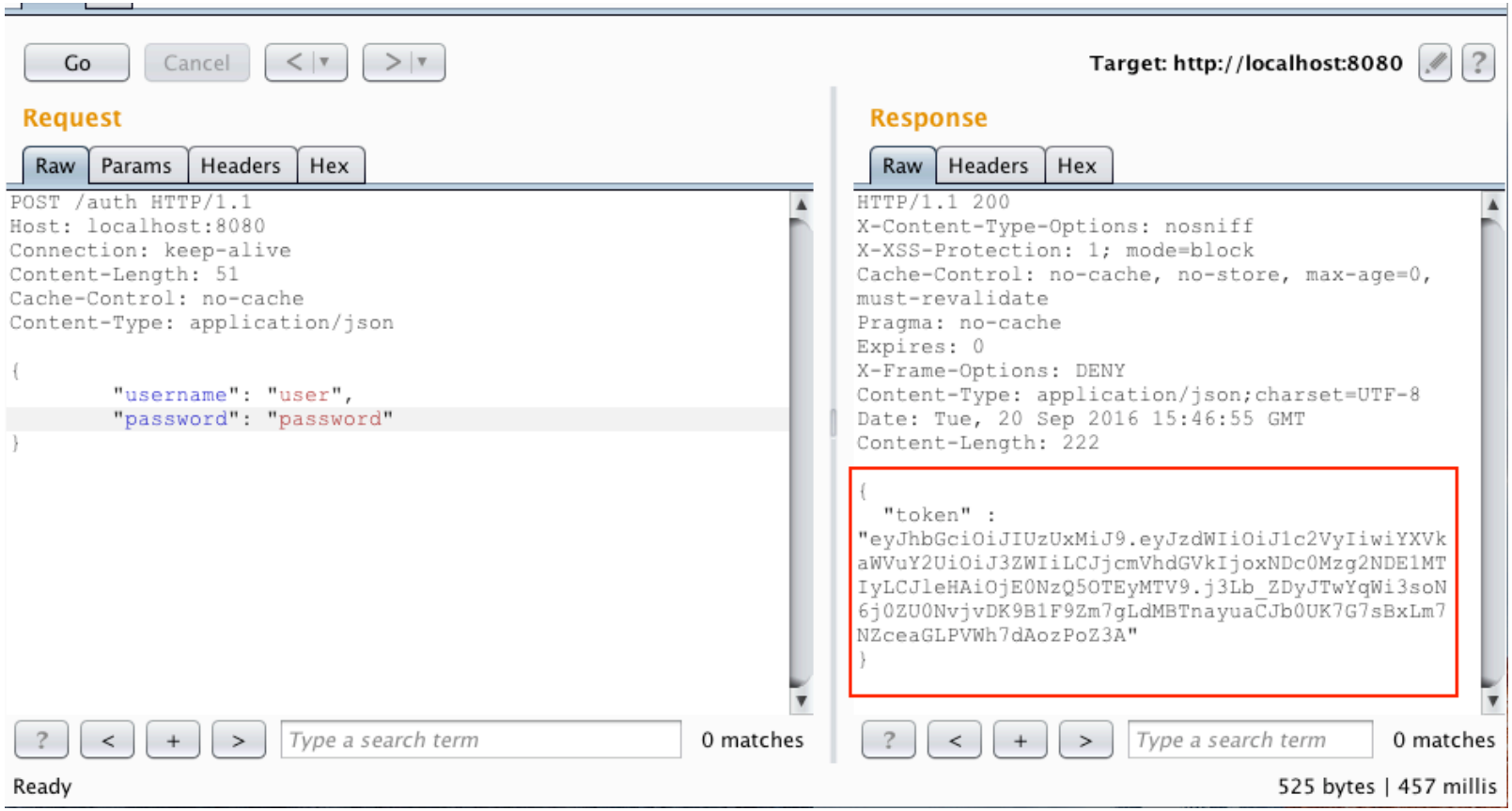


Protect Sensitive Resources

- Implement access controls
  - Role based and/or user-level controls

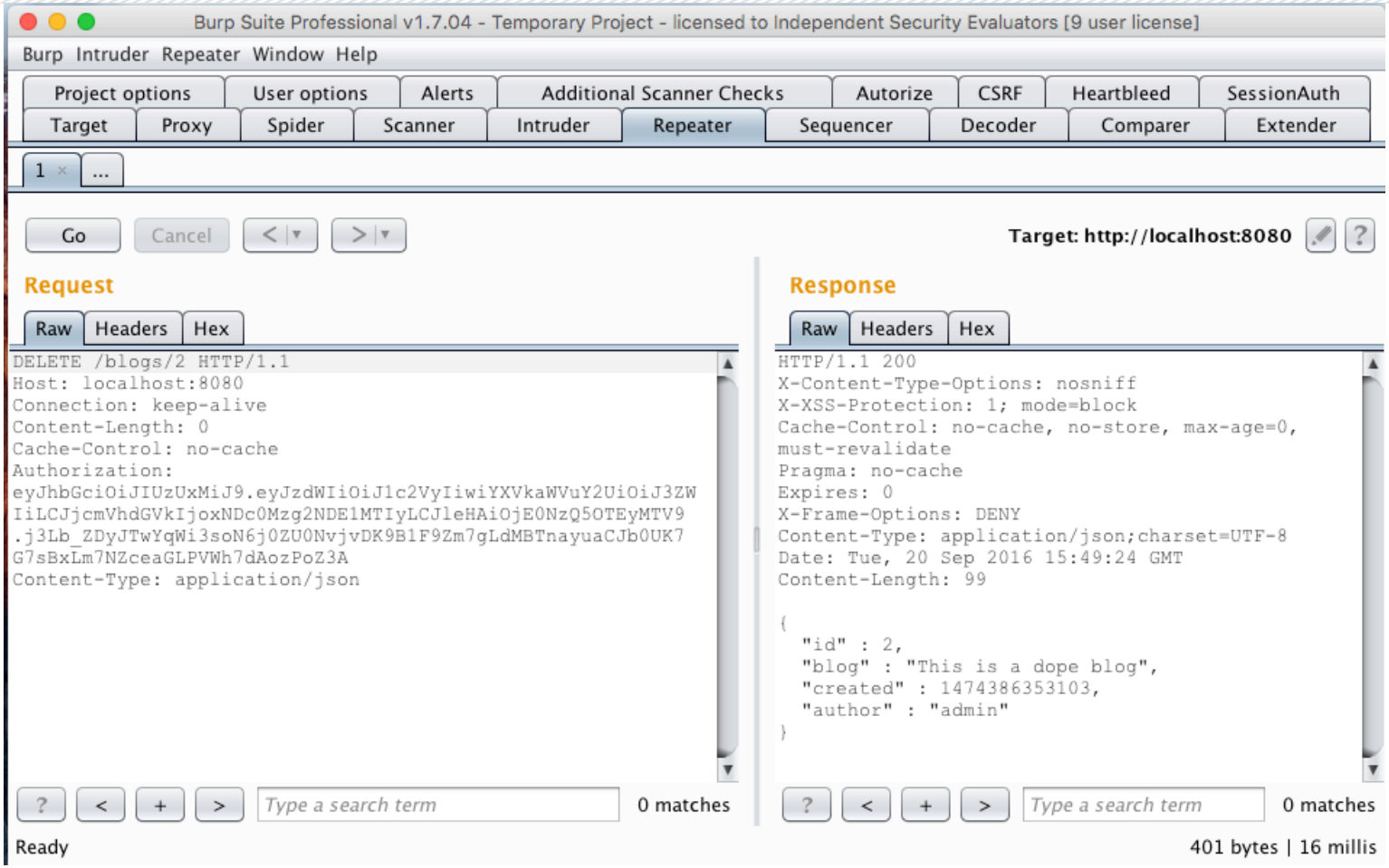
- Every user/tenant should not have access to every resource

Protect Sensitive Resources



Protect Sensitive Resources

Protect Sensitive Resources



Protect Sensitive Resources

What the code looks like…

```
@RequestMapping(path = "/blogs/{id}", method = RequestMethod.DELETE)
public Blog removeBlog(@PathVariable("id") long id, HttpServletRequest request) {

    blog = blogs.get(id);
    return blogs.remove(id);
}
```

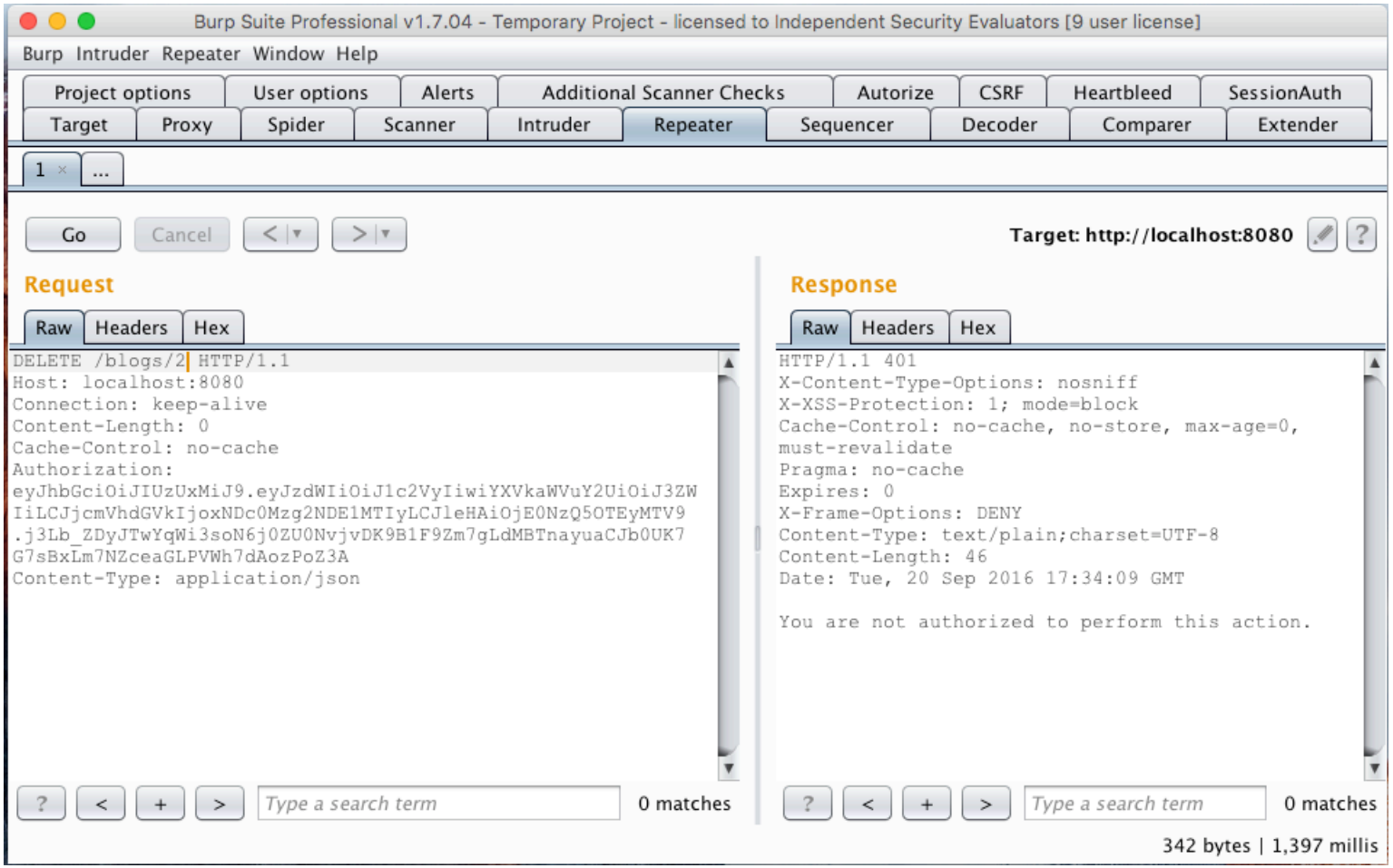
Access controls are not implemented!!

Protect Sensitive Resources

What the code should look like…

```
@RequestMapping(path = "/blogs/{id}", method = RequestMethod.DELETE)
public ResponseEntity<?> removeBlog(@PathVariable("id") long id, HttpServletRequest request) {

    String token = request.getHeader(tokenHeader);
    String username = jwtTokenUtil.getUsernameFromToken(token);
    JwtUser user = (JwtUser) userDetailsService.loadUserByUsername(username);
    Collection<? extends GrantedAuthority> authorities = user.getAuthorities();

    blog = blogs.get(id);

    if (blog.getAuthor().equalsIgnoreCase(username) ||
          authorities.contains(new SimpleGrantedAuthority("ROLE_ADMIN"))){

     blogs.remove(id);
          return new ResponseEntity<> ("Blog was successfully deleted.", HttpStatus.OK);
     }

    else
     return new ResponseEntity<>("You are not authorized to perform this action.",
          HttpStatus.UNAUTHORIZED);
}
```

Protect Sensitive Resources



Input Validation

- Validate Incoming Content-Types
  - Server should never assume the Content-type
  - Content-Type should be checked against the data
    - 400 level status code should be returned if header contains invalid types

Input Validation

Restricting content types

```
@RequestMapping(path = "/blogs",consumes= {"application/json"}, method = RequestMethod.POST)
public Blog addBlog(@RequestBody Blog blog, HttpServletRequest request){

      String token = request.getHeader(tokenHeader);
      String username = jwtTokenUtil.getUsernameFromToken(token);

      blog.setId(blogs.size() + 1);
      blog.setAuthor(username);
      blog.setBlog(blog.getBlog());
      blog.getCreated();
      blogs.put(blog.getId(), blog);
      return blog;
}
```

Spring

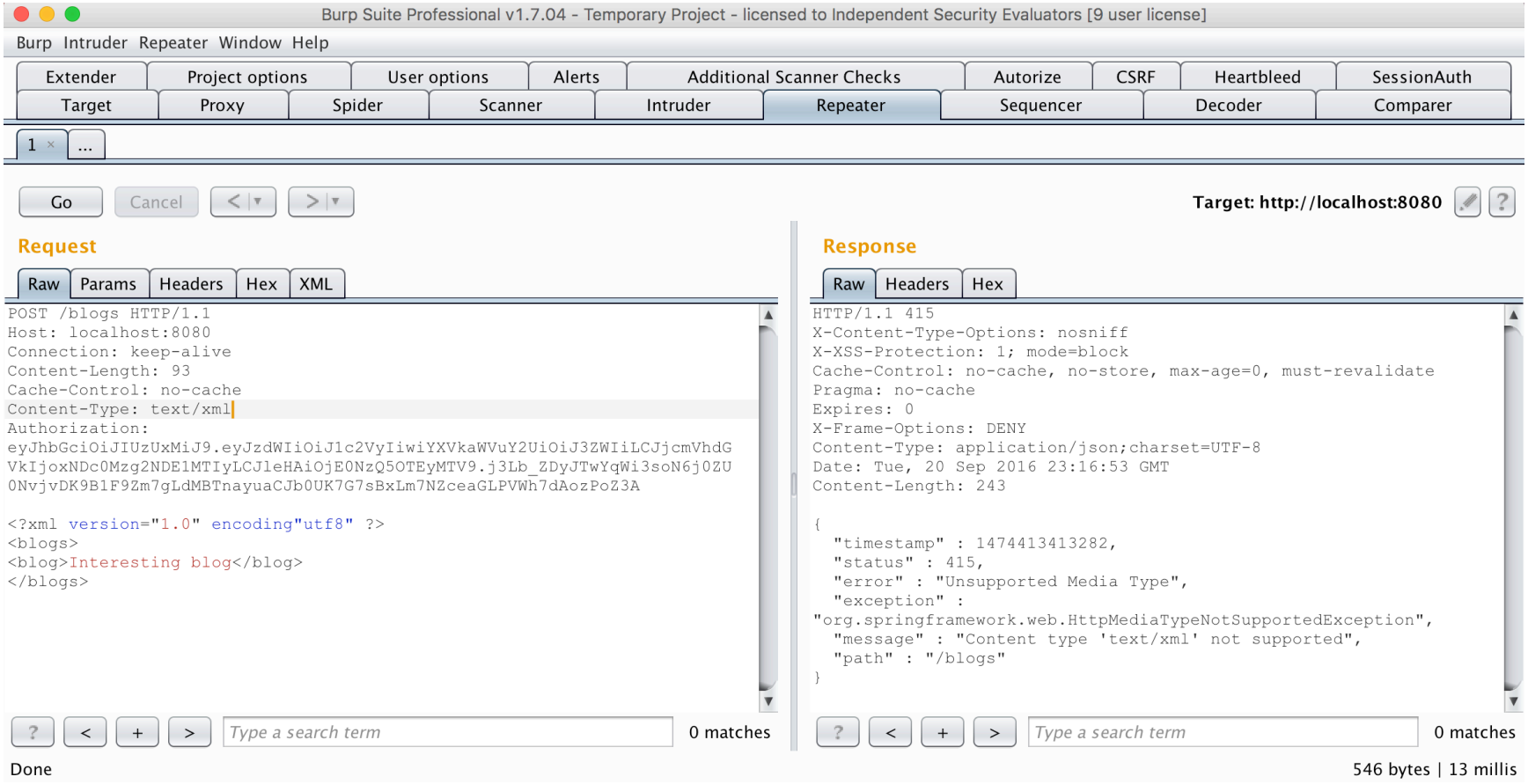
Input Validation

Restricting content types

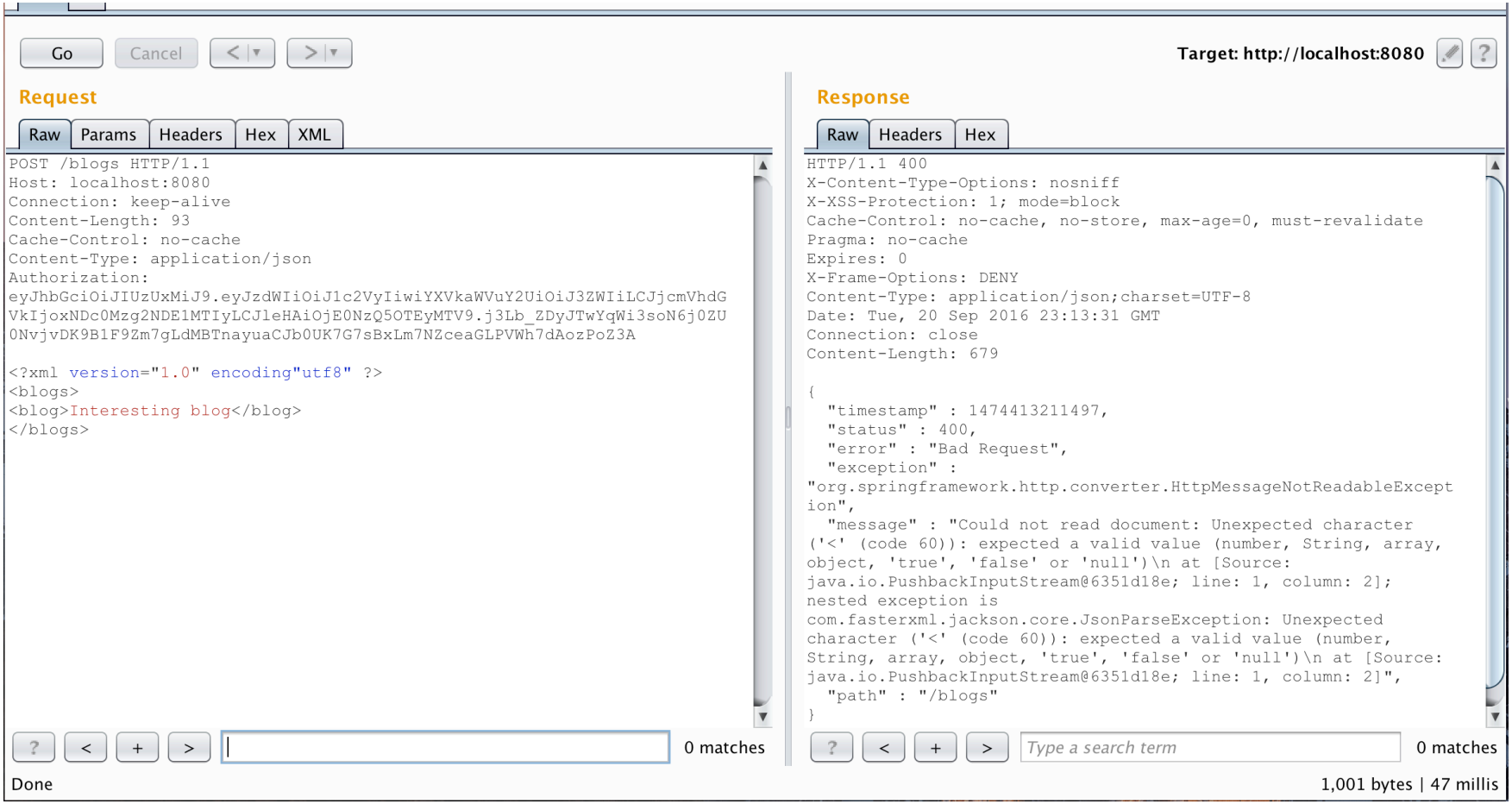
```
@POST
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public Blog addBlog(Blog blog){

    return blogService.addBlog(blog);
}
```

Jersey

Input Validation



Input Validation



Output Encoding

- Security headers should be sent within all responses
  - Content-Type
    - Should contain correct Content-Type
  - X-Content-Type-Options: nosniff
    - Ensures browsers wont attempt to detect a different Content-Type

QUESTIONS??

- Blog API
  - https://github.com/dbran9/jwt-spring-security
  - Based partially off
    - https://github.com/szerhusenBC/jwt-spring-security-demo
- Presentation
  - www.securityevaluators.com/knowledge/presentations

Send questions to dbranch@securityevaluators.com