MANICODE
SECURE CODING EDUCATION

Webservice Security

JIM MANICO Secure Coding Instructor *www.manicode.com*

A little background dirt…

jim@manicode.com

@manicode

- Former OWASP Global Board Member
- Project manager of the OWASP Cheat Sheet Series and several other OWASP projects
- 20+ years of software development experience
- Author of "Iron-Clad Java, Building Secure Web Applications" from McGraw-Hill/Oracle-Press
- Kauai, Hawaii Resident

ORACLE
Iron-Clad Java: Building Secure Web Applications
Java
Best Practices for Secure Java Web Application Development
Jim Manico
August Detlefsen
Contributing Author, Kevin Kenan
Technical Editor, Milton Smith
Oracle Senior Principal Security Product Manager, Java
Oracle Press

WARNING: Please do not attempt to hack any computer system without legal permission to do so. Unauthorized computer hacking is illegal and can be punishable by a range of penalties including loss of job, monetary fines and possible imprisonment.

ALSO: The *Free and Open Source Software* presented in these materials are examples of good secure development techniques. You may have unknown legal, licensing or technical issues when making use of *Free and Open Source Software*. You should consult your company's policy on the use of *Free and Open Source Software* before making use of any software referenced in this material.

"API security is going to be a much bigger topic in 2018. So many companies think their attack surface is the website and that 2FA solves everything but API access is done via tokens and secrets. API security is at least a couple of years behind other types of web security."

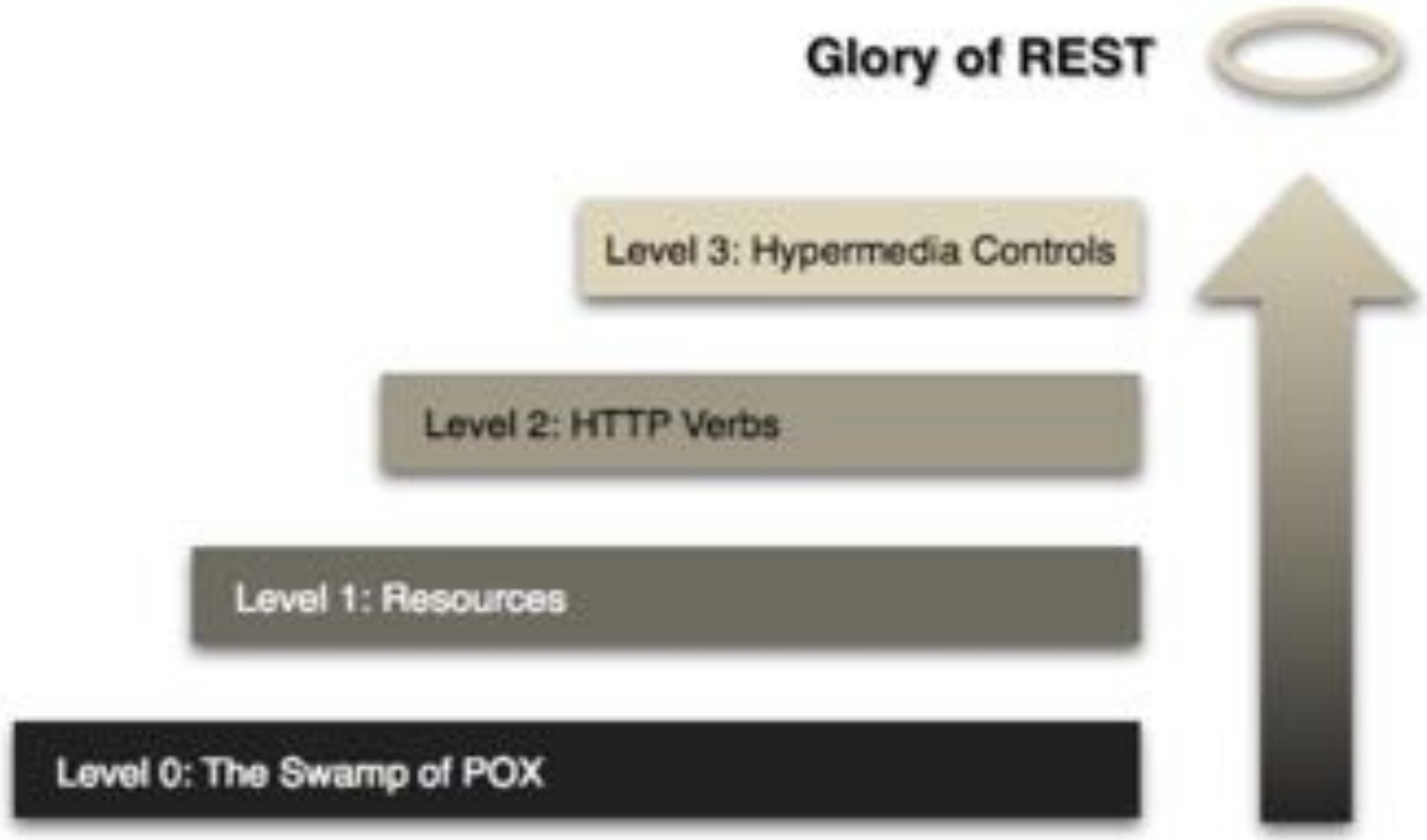
Daniel Miessler
https://danielmiessler.com/podcast/

REST History

- Introduced to the world in a PHD dissertation by Roy Fielding in 2000.
- Promoted HTTP methods (PUT, POST, GET, DELETE) and the URL itself to communicate additional metadata as to the nature of an HTTP request.

Http Method	Database Operation
PUT	Update
POST	Insert
GET	Select
DELETE	Delete

The Glory of REST



http://martinfowler.com/articles/richardsonMaturityModel.html

Level 0 – RPC/POX

```
POST /appointmentService HTTP/1.1
<openSlotRequest date="2010-01-04" doctor="mjones"/>

HTTP/1.1 200 OK
<openSlotList>
  <slot start="1400" end="1450">
    <doctor id="mjones"/>
  </slot>
  <slot start="1600" end="1650">
    <doctor id="mjones"/>
  </slot>
</openSlotList>
```

http://martinfowler.com/articles/richardsonMaturityModel.html

Level 1 – Resources

```
POST /doctors/mjones HTTP/1.1

<openSlotRequest date="2010-01-04"/>

HTTP/1.1 200 OK

<openSlotList>
  <slot id="1234" doctor="mjones" start="1400" end="1450"/>
  <slot id="5678" doctor="mjones" start="1600" end="1650"/>
</openSlotList>
```

http://martinfowler.com/articles/richardsonMaturityModel.html

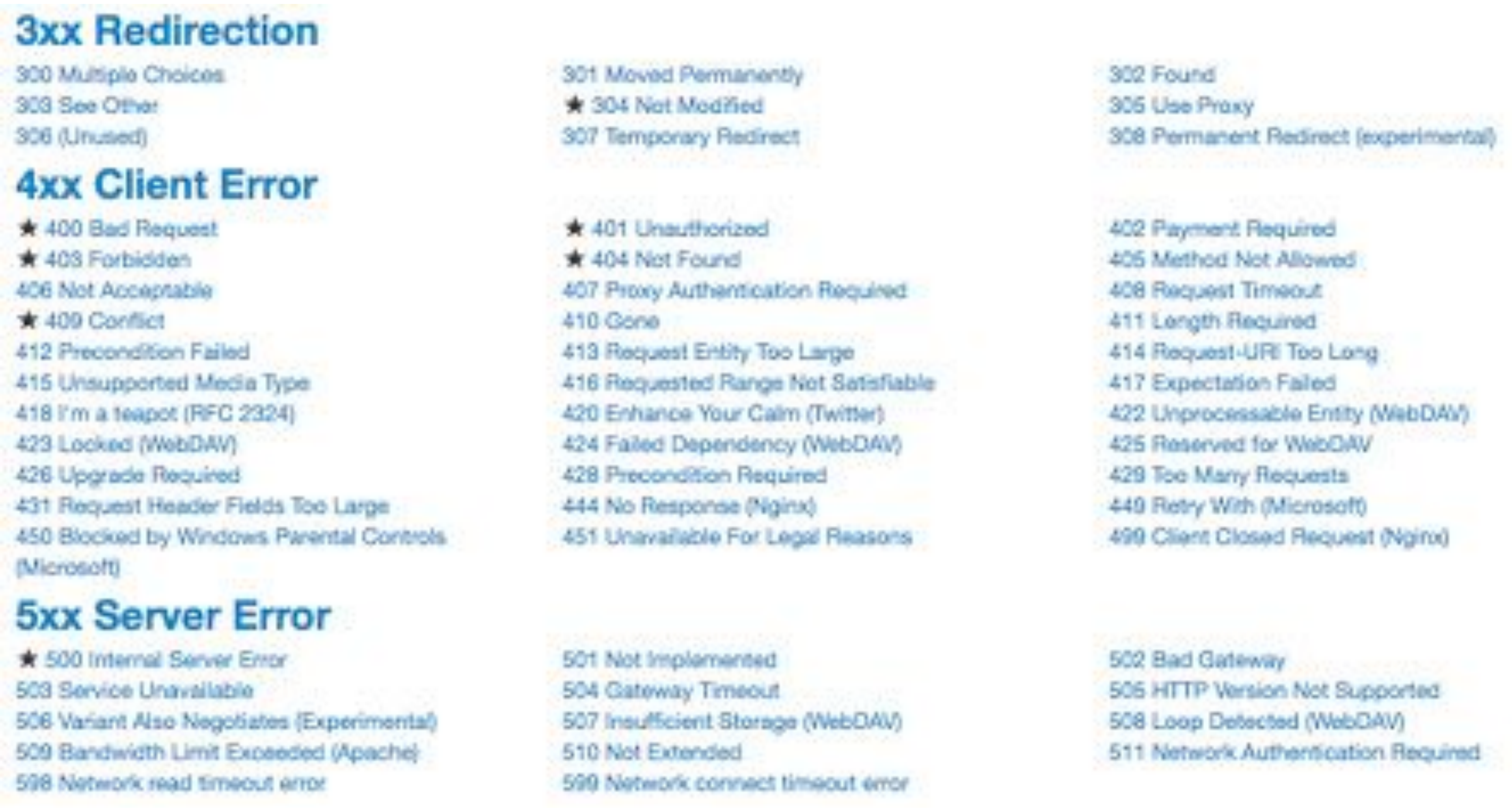
Level 2 – HTTP Verbs

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1

HTTP/1.1 200 OK

<openSlotList>
  <slot id="1234" doctor="mjones" start="1400" end="1450"/>
  <slot id="5678" doctor="mjones" start="1600" end="1650"/>
</openSlotList>
```

Http Method	Database Operation
PUT	Update
POST	Insert
GET (DANGER)	Select
DELETE	Delete

http://martinfowler.com/articles/richardsonMaturityModel.html

Level 2 – HTTP Response Codes



3xx Redirection

300 Multiple Choices	301 Moved Permanently	302 Found
303 See Other	★ 304 Not Modified	305 Use Proxy
306 (Unused)	307 Temporary Redirect	308 Permanent Redirect (experimental)

4xx Client Error

★ 400 Bad Request	★ 401 Unauthorized	402 Payment Required
★ 403 Forbidden	★ 404 Not Found	405 Method Not Allowed
406 Not Acceptable	407 Proxy Authentication Required	408 Request Timeout
★ 409 Conflict	410 Gone	411 Length Required
412 Precondition Failed	413 Request Entity Too Large	414 Request-URI Too Long
415 Unsupported Media Type	416 Requested Range Not Satisfiable	417 Expectation Failed
418 I'm a teapot (RFC 2324)	420 Enhance Your Calm (Twitter)	422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)	424 Failed Dependency (WebDAV)	425 Reserved for WebDAV
426 Upgrade Required	428 Precondition Required	429 Too Many Requests
431 Request Header Fields Too Large	444 No Response (Nginx)	449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)	451 Unavailable For Legal Reasons	499 Client Closed Request (Nginx)

5xx Server Error

★ 500 Internal Server Error	501 Not Implemented	502 Bad Gateway
503 Service Unavailable	504 Gateway Timeout	505 HTTP Version Not Supported
506 Variant Also Negotiates (Experimental)	507 Insufficient Storage (WebDAV)	508 Loop Detected (WebDAV)
509 Bandwidth Limit Exceeded (Apache)	510 Not Extended	511 Network Authentication Required
598 Network read timeout error	599 Network connect timeout error	

Level 2 – HTTP Response Error Codes

```
POST /slots/1234 HTTP/1.1

<appointmentRequest>
  <patient id="jsmith"/>
</appointmentRequest>

HTTP/1.1 201 Created (or) HTTP/1.1 409 Conflict
Location: slots/1234/appointment

<appointment>
  <slot id="1234" doctor="mjones" start="1400" end="1450"/>
  <patient id="jsmith"/>
</appointment>
```

http://martinfowler.com/articles/richardsonMaturityModel.html

Level 3 – Hypermedia (take action part 2)

```
HTTP/1.1 201 Created (or) HTTP/1.1 409 Conflict
Location: slots/1234/appointment
```

```
<appointment>
  <slot id="1234" doctor="mjones" start="1400" end="1450"/>
  <patient id="jsmith"/>
  <link rel="/linkrels/appointment/cancel"
     uri="/slots/1234/appointment"/>
   <link rel="/linkrels/appointment/addTest"
     uri="/slots/1234/appointment/tests"/>
   <link rel="self"
     uri="/slots/1234/appointment"/>
   <link rel="/linkrels/appointment/changeTime"
     uri="/doctors/mjones/slots?date=20100104&status=open"/>
</appointment>
```

http://martinfowler.com/articles/richardsonMaturityModel.html#ILoveKirk

Why?

Level 1 tackles the question of handling complexity by using divide and conquer, breaking a large service endpoint down into multiple resources.

Level 2 introduces a standard set of verbs and other HTTP artifacts so that we handle similar situations in the same way, removing unnecessary variation.

Level 3 introduces discoverability, providing a way of making a protocol more self-documenting.

Why do Webservice Bugs Happen?

- Location in the "**trusted**" network of your data center gives false sense of security
- SSRF (**Server Side Request Forgery**) to Internal REST APIs
- Self describing and **predicable nature** (hypermedia) of REST
- Complete lack of **HTTPS** or placement of sensitive data in URL's
- Complete lack of **Authentication** or use of weak authentication
- Complete lack of **Authorization** or weak authorization design

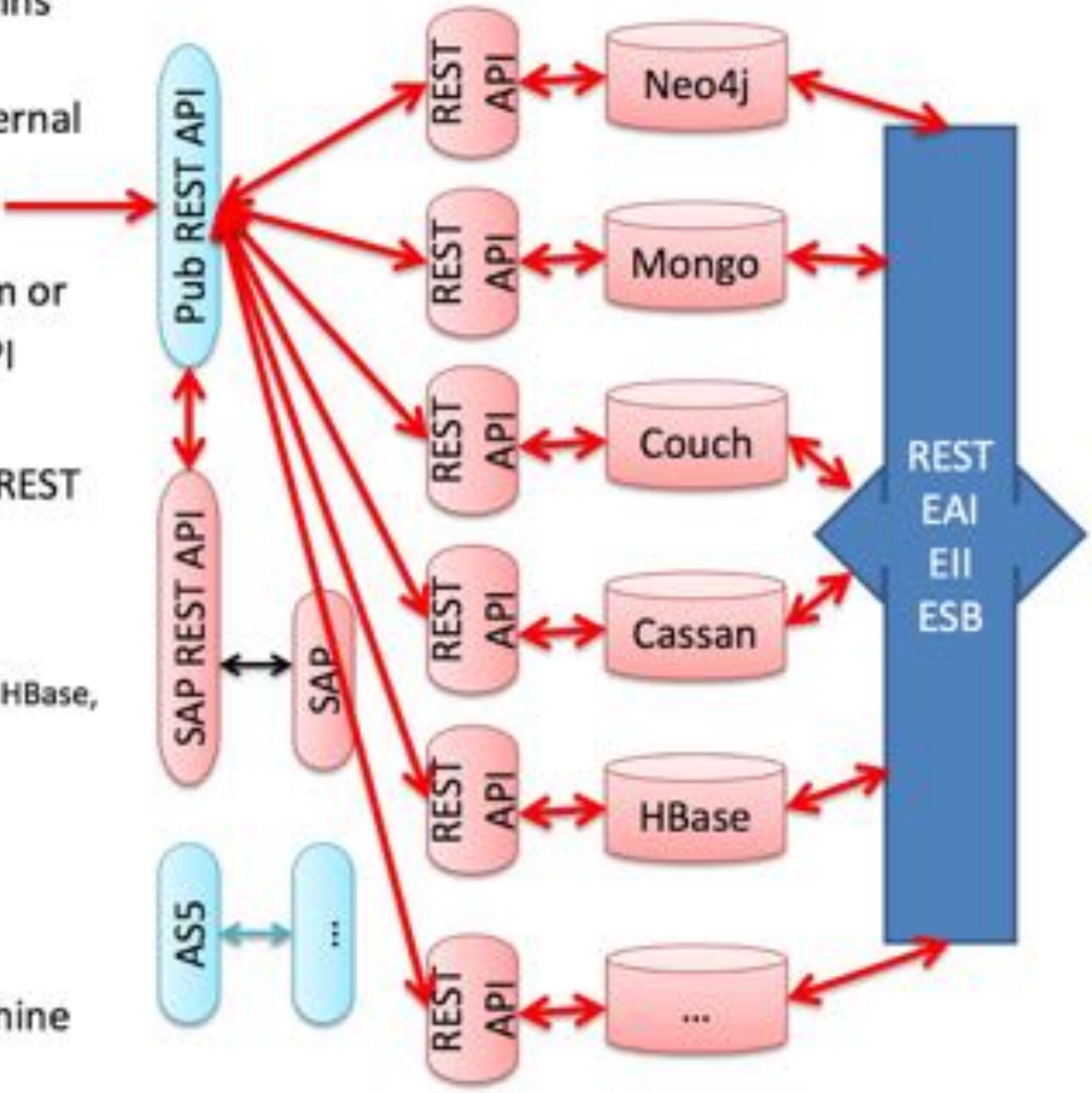
Server Side Request Forgery (SSRF)

Courtesy of Alvaro Munoz
@pwntester

Attacking An Internal Network (REST style)

- Find an HTTP REST proxy w/ vulns
- Figure out which REST based systems are running on the internal network
- Exfiltrate data from the REST interface of the backend system or
- Get RCE on an internal REST API

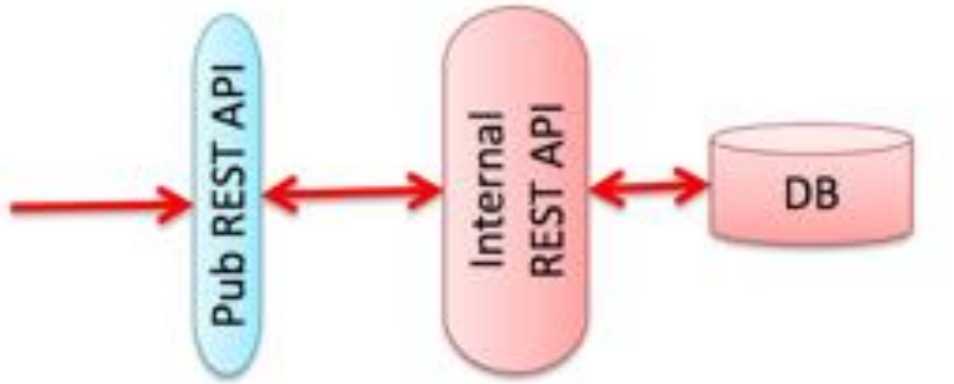
- What backend systems have a REST API that we can attack:
  - ODATA in MS SQL Server
  - Beehive and OAE RESTful API
  - Neo4j, Mongo, Couch, Cassandra, HBase, your company, and many more



X	Non-compromised machine
Y	Affected machine

URLs to backend REST APIs are built with concatenation instead of URIBuilder (Prepared URI)

- Most publically exposed REST APIs turn around and invoke internal REST APIs using URLConnections, Apache HttpClient or other REST clients. If user input is directly concatenated into the URL used to make the backend REST request then the application could be vulnerable to Extended HPPP.



What to Look For

- new URL (“http://yourSvr.com/value” + var);
- new Redirector(getContext(), urlFromCookie, **MODE_SERVER_OUTBOUND**);
- HttpGet(“http://yourSvr.com/value” + var);
- HttpPost(“http://yourSvr.com/value” + var);
- restTemplate.postForObject(”http://localhost :8080/Rest/user/” + var, request, User.class);
- …

Safe URL Construction

http://blog.palominolabs.com/2013/10/03/creating-urls-correctly-and-safely/index.html

```
UrlBuilder.forHost("http", "foo.com")
    .pathSegment("with spaces")
    .pathSegments("path", "with", "varArgs")
    .pathSegment("&=?/")
    .queryParam("fancy + name", "fancy?=value")
    .matrixParam("matrix", "param?")
    .fragment("#?=")
    .toUrlString()
```

Additional SSRF resources

SSRF Testing Resources

- https://github.com/cujanovic/SSRF-Testing/blob/master/README.md

Nicolas Gregoire talk at AppSecEU of SSRF

- http://www.agarri.fr/docs/AppSecEU15-Server_side_browsing_considered_harmful.pdf
- https://www.youtube.com/watch?v=8t5-A4ASTIU

Great talk by Orange Tsai at BlackHat and Defcon

- https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf
- http://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html
- https://www.youtube.com/watch?v=D1S-G8rJrEk

Faking Out Security Filters (Bypass)

User

- Hacker
- "_method" parameter
- "X-HTTP-Method-Override" header

Security Filter/Servlet

- Looks like a GET **but turns into PUT, POST, or DELETE**
- creditInfo?_method=PUT

TLS

Transport Layer Security

“Cryptography is only truly useful if the rest of the system is also sufficiently secure against the attackers.”

Bruce Schneier
Security Engineering

HTTPS / TLS: When and How

Where should HTTPS be used at minimum?

EVERYWHERE

Webservice Authentication

Webservice Authentication and Session Management

- First identify the server via TLS and a certificate authority of some kind.

- **Single Server Consumer Apps**: Web Sessions
- **Federated Consumer Apps**: OpenID Connect
- **Stateless Microservices**: JWT
- **Machine Acting on Behalf of Users**: OAuth 2 (Delegation)
- **Strict Machine to Machine Communication:** Mutual TLS

Webservice Access Control

INSECURE OBJECT REFERENCE

Request

```
GET https://api.example.com/users/1234/private-messages
```

Controller: Attack

```
## PYTHON

class PrivateMessagesView(APIView):
  def get(self, request, user_id):
    """Get the private messages for a specific user"""
    msgs=private_messages(user_id)
    return Response(data=msgs, status=200)
```

Controller: Remediation

```
class PrivateMessagesView(APIView):
  def get(self, request, user_id):
    """Get the private messages for a specific user"""
    if request.user.id != user_id:
      return Response(data={'msg': 'forbidden'}, status=403)
    msgs=private_messages(user_id)
    return Response(data=msgs, status=200)
```

INSECURE OBJECT REFERENCE: DEFENSES

- Verify that data being accessed is owned by by current authenticated user
- Consider lookup maps between object ids and user ids or user group ids
- Verify user authorization to objects using a modern access control design such as capabilities

HTTP METHODS PROTECTION

Ensure that a requesting user is authorized to use a given method

- Anonymous user cannot DELETE a blog article
- Anonymous user can GET a blog article
- Admin User can POST, PUT, DELETE, and GET a blog article

UNAUTHORIZED PRIVILEGED ACTIONS: EXAMPLES

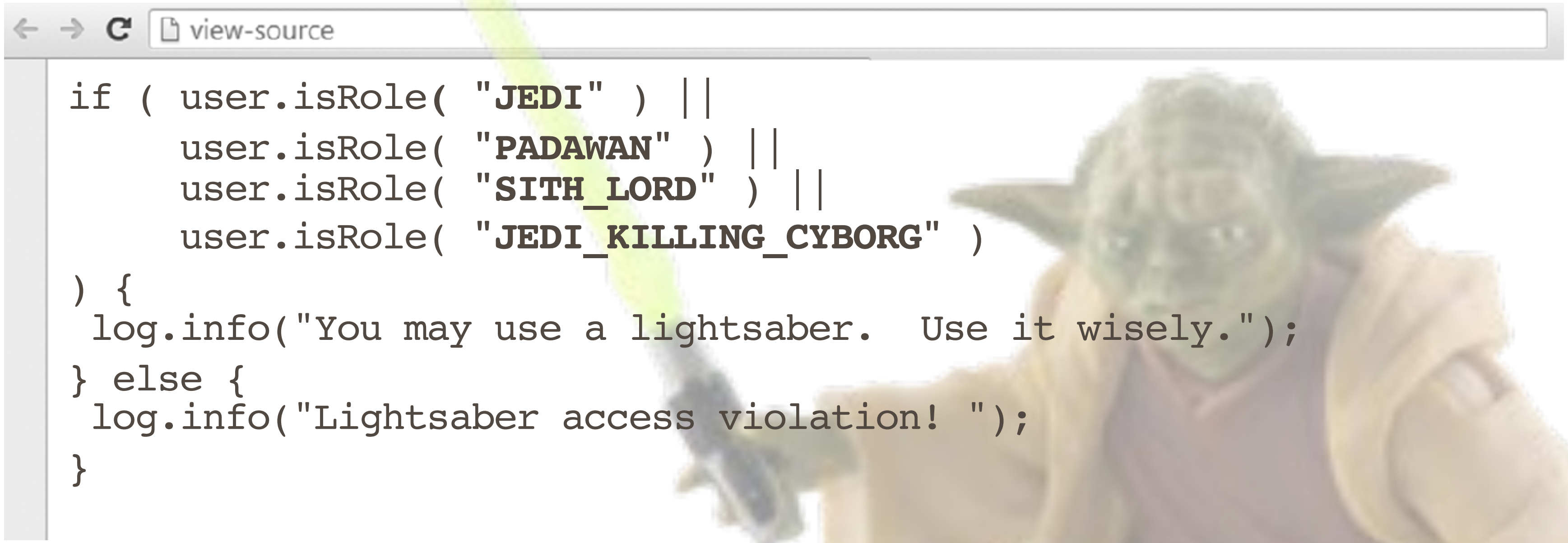
Controller: Vulnerable

```
#PYTHON

class AdminCommentsView(APIView):
  def delete(self, request, comment_id):
    """Allow an Admin to delete a comment"""
    comment=get_comment(comment_id)
    comment.delete()
    return Response(status=204)
```

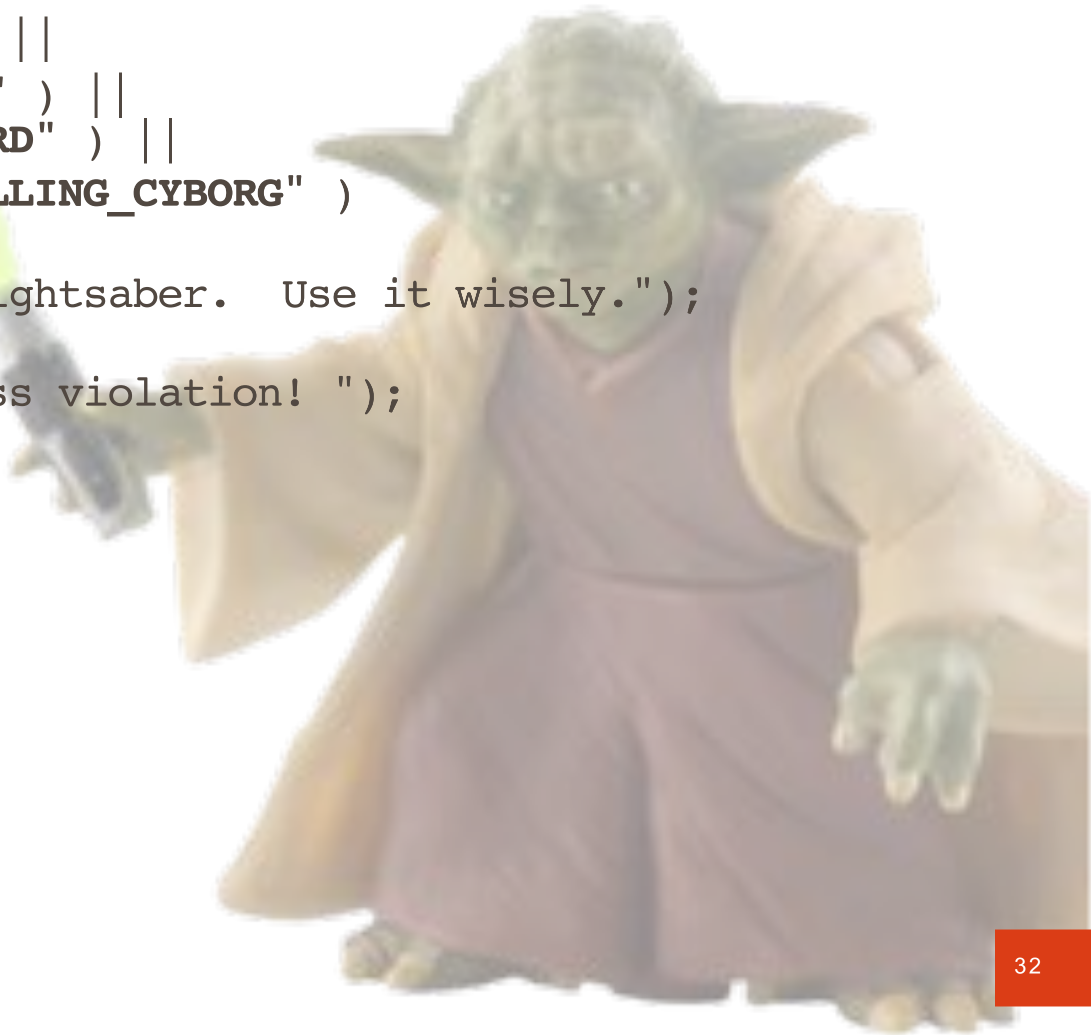
Controller: Defense

```
  def delete(self, request, comment_id):
    """Allow an Admin to delete a comment"""
    comment=get_comment(comment_id)
    ## Does `request.user.id` have permission to "delete" a "comment"
    ## where the "comment_id" is `comment_id`?
    perm=\
      has_permission(
        request.user.id,
        'comment',
        'comment_id',
        comment_id,
        'delete')
    if not perm:
      return Response(data={'msg': 'forbidden'}, status=403)
    comment.delete()
    return Response(status=204)
```

Role Based Example
Do not or **do not do this**!



```
if ( user.isRole( "JEDI" ) ||
     user.isRole( "PADAWAN" ) ||
     user.isRole( "SITH_LORD" ) ||
     user.isRole( "JEDI_KILLING_CYBORG" )
) {
 log.info("You may use a lightsaber.  Use it wisely.");
} else {
 log.info("Lightsaber access violation! ");
}
```

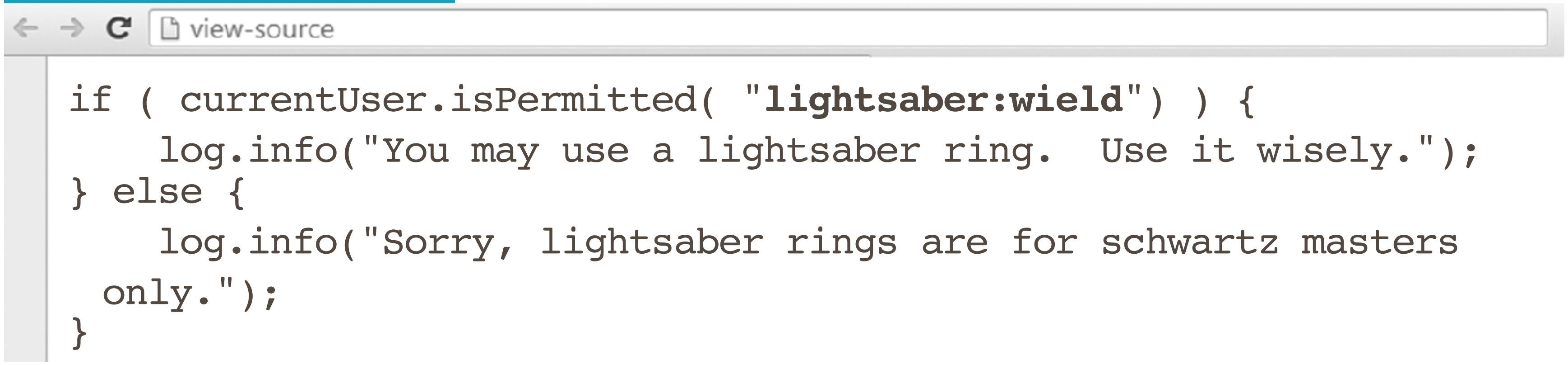


Permission (Claims) Based Access Control Enforcement Points

The Problem

Web Application needs secure access control mechanism

The Solution



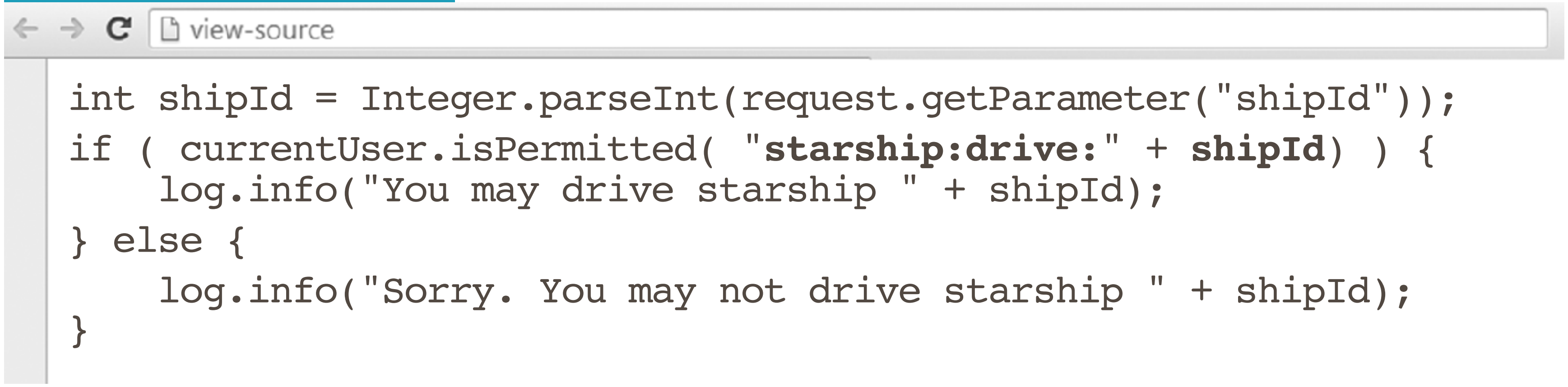
```
if ( currentUser.isPermitted( "lightsaber:wield") ) {
    log.info("You may use a lightsaber ring.  Use it wisely.");
} else {
    log.info("Sorry, lightsaber rings are for schwartz masters
  only.");
}
```

Permission (Claims) Based Access Control Enforcement Points

The Problem

Web Application needs secure access control mechanism

The Solution



```
int shipId = Integer.parseInt(request.getParameter("shipId"));
if ( currentUser.isPermitted( "starship:drive:" + shipId) ) {
    log.info("You may drive starship " + shipId);
} else {
    log.info("Sorry. You may not drive starship " + shipId);
}
```

Basic Data Contextual Access Control Schema

Permission / Feature

Permission ID	Permission Name	Data Check T/F	Data Type ID	Customer ID
15	lightsaber:wield	F		1
16	lightsaber:repair	T	11	1
25	starship:drive	T	10	1

Data Type

Data Type ID	Data Name
10	Starship
11	Lightsaber

User / User Group

UID	User Name
1	Luke Skywalker
2	Han Solo

Entitlements

User ID	Permission ID	Role/Group ID	Data Element ID	Data Group Id
1	15			
2	25		1138	
	15	5 (Jedi)		

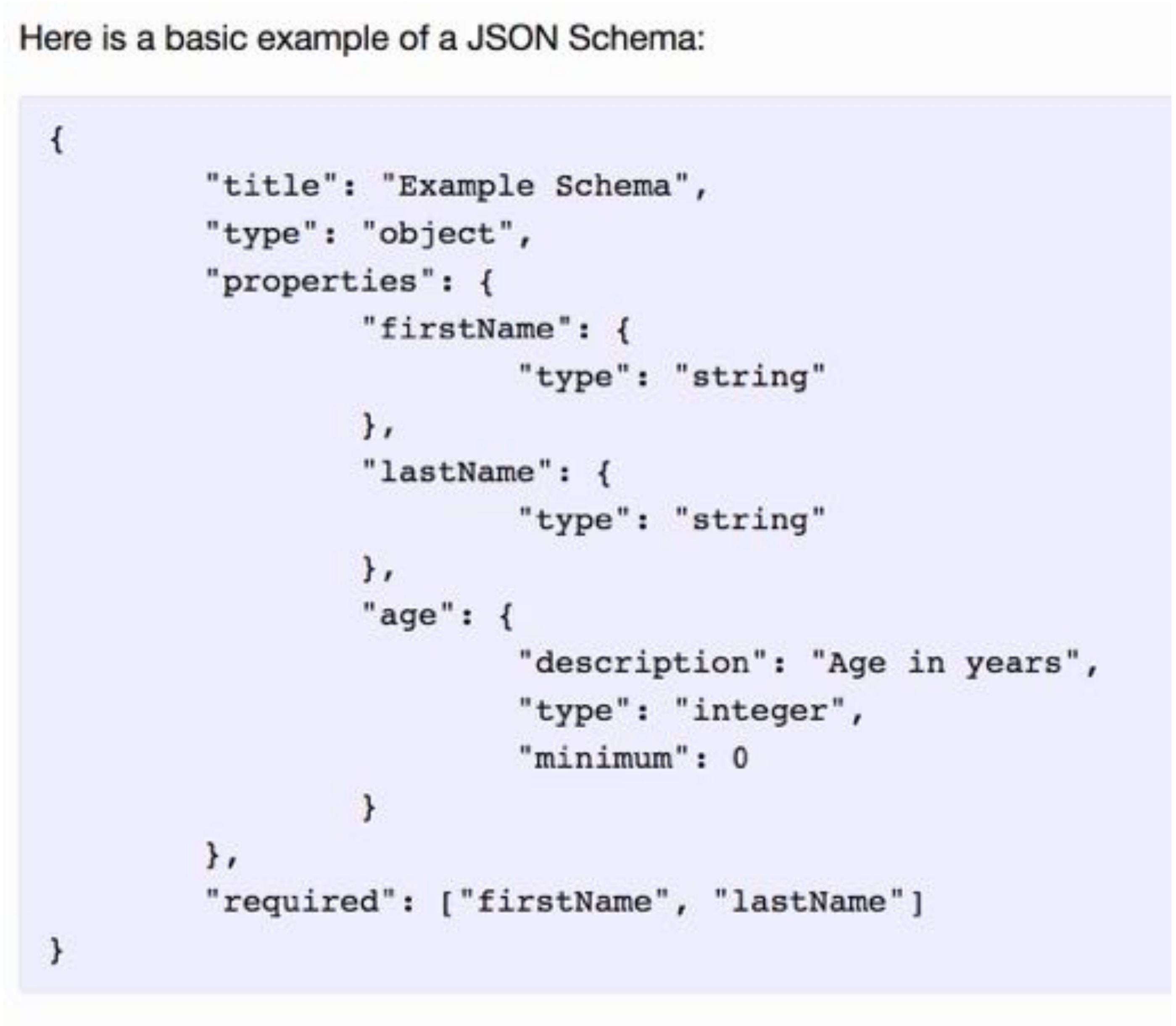
Server Side JSON Issues

Should you trust all JSON? (no)

```
{
    "first_name":
     "' or 1=1-- ",
    "homepage":
     "http://www.bad.com/packx1/cs.jpg?&cmd=uname%20-a",
    "username":
     "*)(uid=*))(|(uid=*",
    "email":
     "woot'or'1'!='ing@manico.net",
    "profile_image":
     "../../../../../../../etc/passwd",
    "location":
     "(function() { alert('XSS 1!'); return 'somewhere'})()",
    "bio":
     "<script>document.body.innerHTML='<h1>TomWazHere';</script>"
}
```

JSON SERVER-SIDE INPUT VALIDATION

Validate that the JSON is actually correct, parseable JSON	Parse the JSON safely	Parseable JSON may contain dangerous data!
Start by ensuring that the JSON is of the correct format by validating against a **JSON Schema** for each webservice endpoint. http://json-schema.org/	Parse the JSON using a battle-tested and **updated** JSON parser. *JSON parsers have a history of security vulnerablities related to security problems with serialization and deserialization.*	Even if a JSON string is correct and parseable JSON, it can still be unsafe from wrong data types. *Use query parameterization in any SQL queries which use JSON input as input parameters* *Use proper XSS defense if JSON input is used as output to browser*

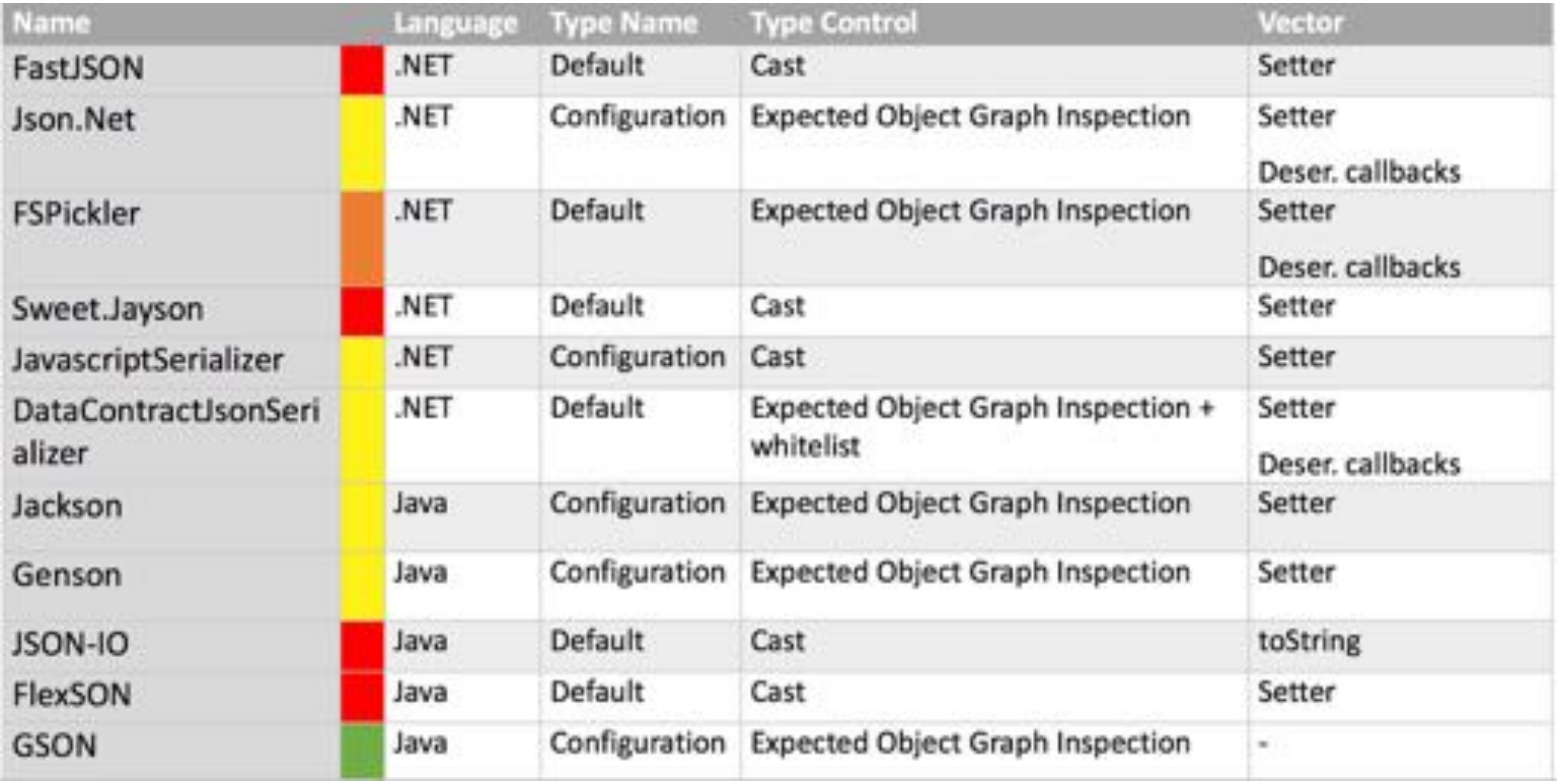


Here is a basic example of a JSON Schema:

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

http://json-schema.org/examples.html

JSON parsers are mostly insecure

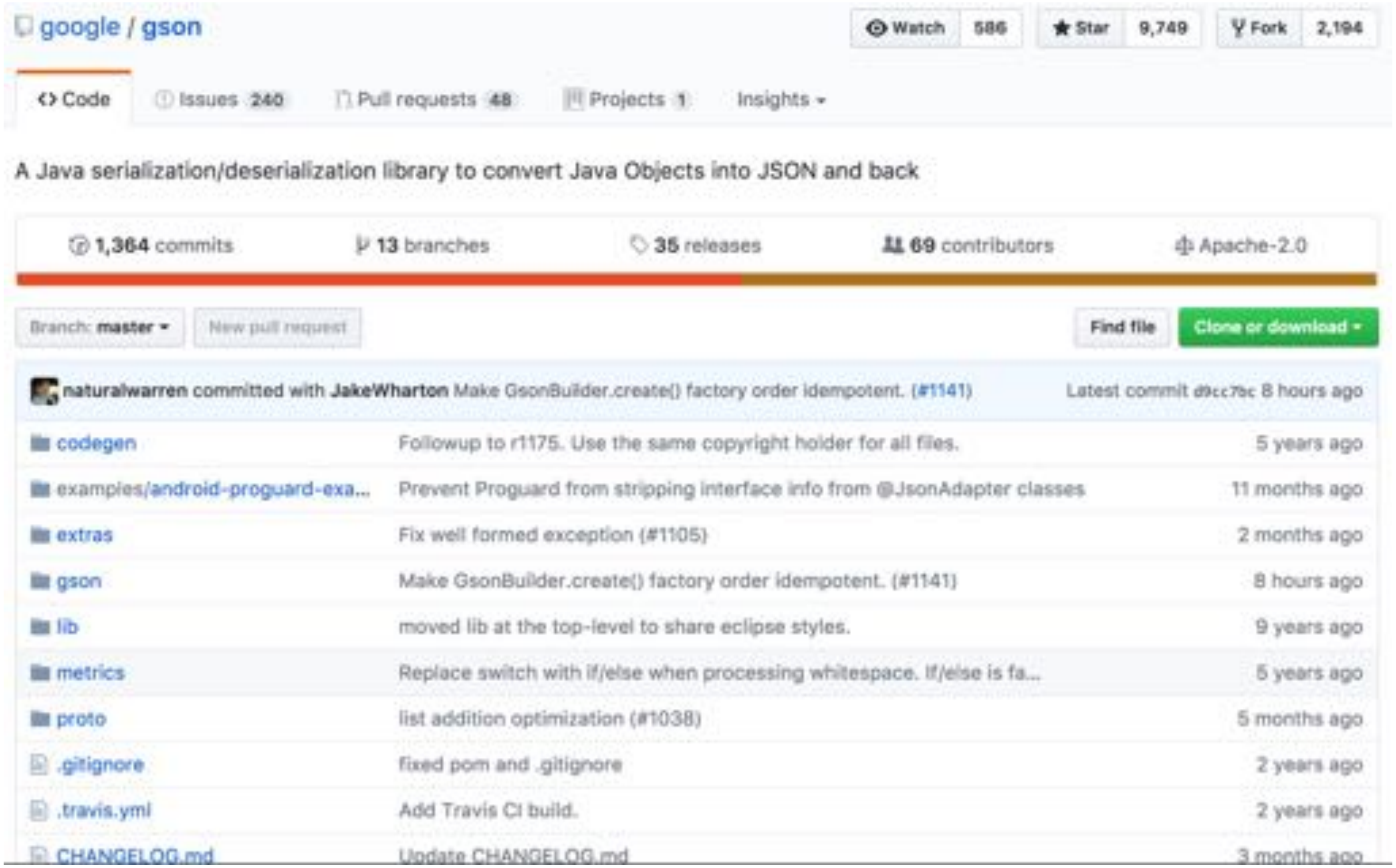


Name		Language	Type Name	Type Control	Vector
FastJSON	🟥	.NET	Default	Cast	Setter
Json.Net	🟨	.NET	Configuration	Expected Object Graph Inspection	Setter Deser. callbacks
FSPickler	🟧	.NET	Default	Expected Object Graph Inspection	Setter Deser. callbacks
Sweet.Jayson	🟥	.NET	Default	Cast	Setter
JavascriptSerializer	🟨	.NET	Configuration	Cast	Setter
DataContractJsonSerializer	🟨	.NET	Default	Expected Object Graph Inspection + whitelist	Setter Deser. callbacks
Jackson	🟨	Java	Configuration	Expected Object Graph Inspection	Setter
Genson	🟨	Java	Configuration	Expected Object Graph Inspection	Setter
JSON-IO	🟥	Java	Default	Cast	toString
FlexSON	🟥	Java	Default	Cast	Setter
GSON	🟩	Java	Configuration	Expected Object Graph Inspection	-

Alvaro Muñoz – July 2017 – Blackhat

Security Research with HPE @pwntester

https://github.com/google/gson



A Java serialization/deserialization library to convert Java Objects into JSON and back

1,364 commits | 13 branches | 35 releases | 69 contributors | Apache-2.0

naturalwarren committed with JakeWharton Make GsonBuilder.create() factory order idempotent. (#1141) Latest commit d9cc7bc 8 hours ago

codegen	Followup to r1175. Use the same copyright holder for all files.	5 years ago
examples/android-proguard-exa...	Prevent Proguard from stripping interface info from @JsonAdapter classes	11 months ago
extras	Fix well formed exception (#1105)	2 months ago
gson	Make GsonBuilder.create() factory order idempotent. (#1141)	8 hours ago
lib	moved lib at the top-level to share eclipse styles.	9 years ago
metrics	Replace switch with if/else when processing whitespace. If/else is fa...	5 years ago
proto	list addition optimization (#1038)	5 months ago
.gitignore	fixed pom and .gitignore	2 years ago
.travis.yml	Add Travis CI build.	2 years ago
CHANGELOG.md	Update CHANGELOG.md	3 months ago

JSON HIJACKING

JSON Endpoint

```
GET https://mybank.example/purchases.json
```

```
[{"vendor": "Acme Widgets", "amount": 4509.10}]
```

Attack: Code on Evil Site

```
<html>
  <body>
    <script type="text/javascript">
      Object.prototype.__defineSetter__('Id', function (obj) {
        console.log("Stolen Object", obj);
      });
    </script>
    <script src="https://mybank.example/purchases.json"></script>
  </body>
</html>
```

Remediation

```
while(1);[{"vendor": "Acme Widgets", "amount": 4509.10}]
```

Modern browsers are not affected, but it is still good to implement a defense in case of older browser or future browser regression

Credit: Phil Haack (http://haacked.com/archive/2009/06/25/json-hijacking.aspx/)

XML

XML Input Parsing Security Checklist

- Do not allow input documents to contain DTDs
- Do not expand entities
- Do not resolve external references
- Impose limits on recursive parse depth
- Limit total input size of document
- Limit parse time of document
- Use an incremental or stream parser such as SAX for large documents
- Validate and properly quote arguments to XSL transformations and XPath queries
- Do not use XPath expression from untrusted sources
- Do not apply XSL transformations that come untrusted sources

Credit: https://pypi.python.org/pypi/defusedxml#how-to-avoid-xml-vulnerabilities

XML Schema Validation

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="orderperson" type="xs:string"/>
   <xs:element name="shipto">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
```

http://www.w3schools.com/XML/schema_example.asp

XML EXTERNAL ENTITY PROCESSING

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<foo>&xxe;</foo>
```

Remediation

Specify the option to the XML parser to make sure it does not include external entities

https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

XEE Prevention in Java/JAXP

Disable all external entity references

```
// Document Builder
DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
dbf.setAttribute({{XMLConstants.ACCESS_EXTERNAL_DTD}}, "");
dbf.setAttribute({{XMLConstants.ACCESS_EXTERNAL_SCHEMA}}, "");
dbf.setAttribute({{XMLConstants.ACCESS_EXTERNAL_STYLESHEET}}, "");

// SAX Parser
SAXParserFactory spf=SAXParserFactory.newInstance();
SAXParser parser=spf.newSAXParser();
parser.setProperty({{XMLConstants.ACCESS_EXTERNAL_DTD}}, "");
parser.setProperty({{XMLConstants.ACCESS_EXTERNAL_SCHEMA}}, "");
parser.setProperty({{XMLConstants.ACCESS_EXTERNAL_STYLESHEET}}, "");

// XML Input
XMLInputFactory xif=XMLInputFactory.newInstance();
xif.setProperty({{XMLConstants.ACCESS_EXTERNAL_DTD}}, "");
xif.setProperty({{XMLConstants.ACCESS_EXTERNAL_SCHEMA}}, "");
xif.setProperty({{XMLConstants.ACCESS_EXTERNAL_STYLESHEET}}, "");

// Schema
SchemaFactory schemaFactory=SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
schemaFactory.setProperty({{XMLConstants.ACCESS_EXTERNAL_DTD}}, "");
schemaFactory.setProperty({{XMLConstants.ACCESS_EXTERNAL_SCHEMA}}, "");
schemaFactory.setProperty({{XMLConstants.ACCESS_EXTERNAL_STYLESHEET}}, "");

// Transformer
TransformerFactory factory=TransformerFactory.newInstance();
factory.setAttribute({{XMLConstants.ACCESS_EXTERNAL_DTD}}, "");
factory.setAttribute({{XMLConstants.ACCESS_EXTERNAL_SCHEMA}}, "");
factory.setAttribute({{XMLConstants.ACCESS_EXTERNAL_STYLESHEET}}, "");
```

https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

XML EXPONENTIAL ENTITY EXPANSION

"Billion Laughs Attack"

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Remediation

- Disable DTD inclusion in document
- Set depth limits on recursive parsing
- Set memory limits for parser

XSLT Injection

```
## Python

def get(self, request):
    xml=StringIO(request.POST['xml'])
    xslt=StringIO(request.POST['xslt'])
    xslt_root=etree.XML(xslt)
    transform=etree.XSLT(xslt_root)
    result_doc=transform(xml)
    res=etree.tostring(result_doc)
    return Response(res)
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="document('/etc/passwd')"/>
  </xsl:template>
</xsl:stylesheet>
```

Never process untrusted user XSLT transformations!

Credit: http://www.hpenterprisesecurity.com/vulncat/en/vulncat/php/xslt_injection.html

Tokens

A JWT is a base64-encoded data object

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkaXN0cmluZXQuY3Mua3VsZXV2ZW4uYmUiLCJleHAiOjI0MjUwNzgwMDAwMDAsIm5hbWUiOiJwaGlsaXBwZSIsImFkbWluIjp0cnVlfQ.dIi1OguZ7K3ADFnPOsmX2nEpF2Asq89g7GTuyQuN3so

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

Header

```
{
  "iss": ”distrinet.cs
          .kuleuven.be",
  "exp": 1425078000000,
  "name": "philippe",
  "admin": true
}
```

Payload

```
HMACSHA256(
    base64UrlEncode(header)
    + "." +
    base64UrlEncode(payload),
    “secret”
)
```

Signature

JSON WEB TOKEN

If you must use token authorization instead of a httpOnly cookie, make sure you transfer and store the token with a "JSON Web Token".

How a JWT is Built

```
## PYTHON

# secret is best as a per-user secret, such as the HMAC'd password
secret="MYSECRET!"
header=base64urlEncode(json.dumps({"alg": "HS256", "typ": "JWT"}))
claims=\
  base64urlEncode(json.dumps(
    {
      "iss": "Acme Bank",
      "iat": 1417414339,
      "exp": 1448950339,
      "aud": "acme.example",
      "sub": "1234",
      "username": "john.doe"
    }
  ))
signature=HMACSHA256(".".join([header, claims], secret)
token=".".join([header, claims, signature])
```

Generated Token

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJBY21lIEJhbmsiLCJpYXQiOjE0MTc0MTQzMzksIm
V4cCI6MTQ0ODk1MDMzOSwiYXVkIjoiYWNtZS5leGFtcGxlIiwic3ViIjoiMTIzNCIsInVzZXJuYW1lIjoiam9ob
i5kb2UifQ.9DttD6SC7VLoZnWhFAqbdmRm-LTgHzRjEpMUOamZT3I
```

https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-31

TOKEN ROTATION

An additional layer of security is to have the server issue a new token periodically.

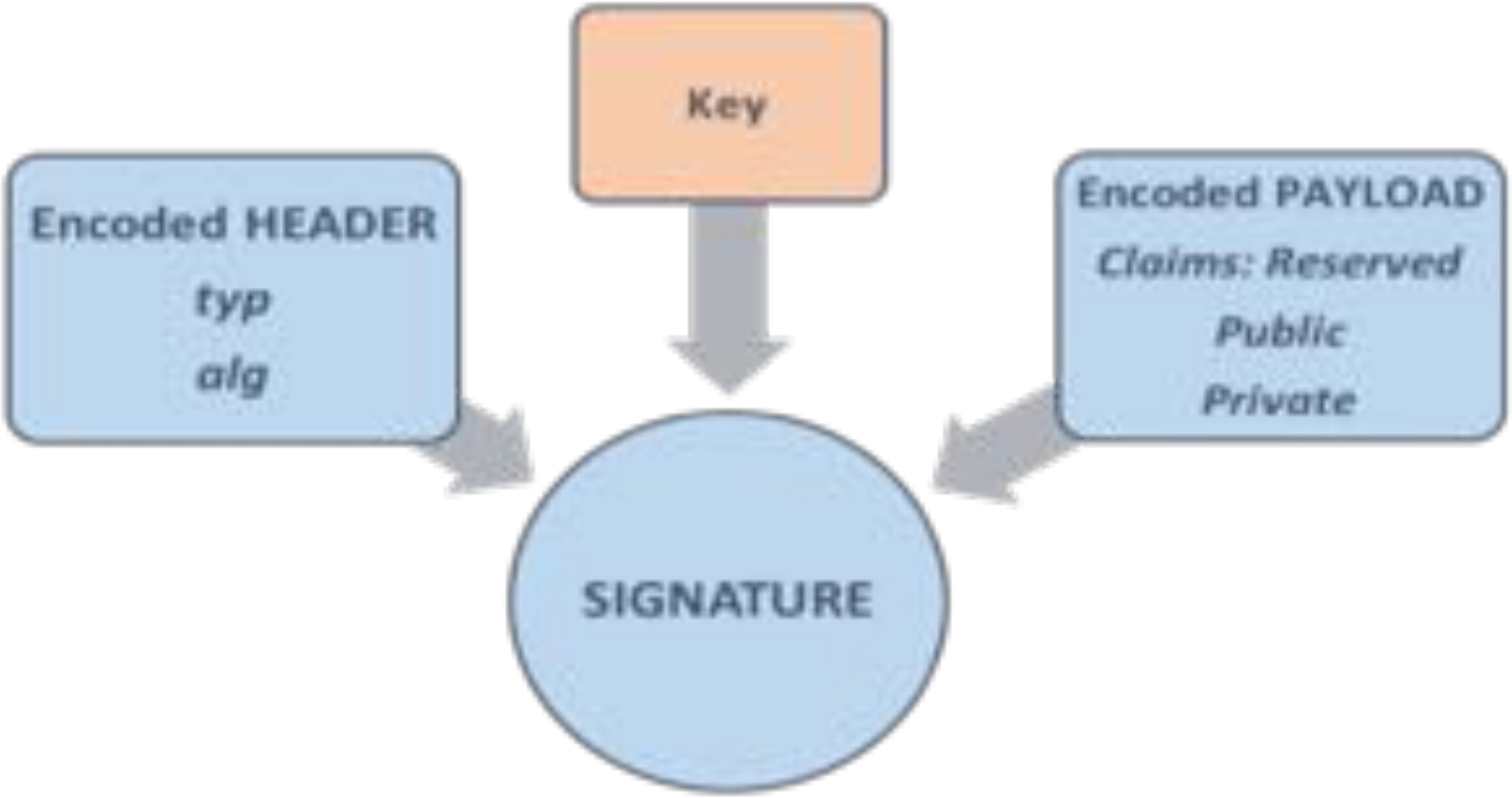
```
GET https://myblog.example/articles.json HTTP/1.0
Authorization: Bearer
 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJBY21lIEJhbmsiLCJpYXQiOjE0MTc0MTQzMzksI
 mV4cCI6MTQ0ODk1MDMzOSwiYXVkIjoiYWNtZS5leGFtcGxlIiwic3ViIjoiMTIzNCIsInVzZXJuYW1lIjoiam9
 obi5kb2UifQ.9DttD6SC7VLoZnWhFAqbdmRm-LTgHzRjEpMUOamZT3I
```

```
HTTP/1.0 200 OK
X-Authorization-New-Token:
 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJBY21lIEJhbmsiLCJpYXQiOjE0MTc0MTQzMzksI
 mV4cCI6MTQ0ODk1MDMzOSwiYXVkIjoiYWNtZS5leGFtcGxlIiwic3ViIjoiMTIzNCIsInVzZXJuYW1lIjoiam9
 obi5kb2UifQ.MZZQoJRlQqPN8uoHZpz8uZsAEBMpxxR0xmi_yxx7sWY
Content-Type: application/json
Content-Length: 1354
{"data": [{"id": 1234}]}
```

This can be done in different ways:

- Once a specific period of time has elapsed, via token expiration. Note that token expiration is not necessarily the same as session expiration.
- After a certain number of requests have been made.
- Be sure to fully expire this token from time to time (absolute and idle timeout)

JSON Web Tokens or "JOT's"



https://www.notsosecure.com/crafting-way-json-web-tokens/

JWT is an open standard to exchange information

JWT tokens represent easy-to-exchange data objects

Content is signed to ensure integrity

Content is base64-encoded, to ensure safe handling across the web

JWT supports various kinds of algorithms

E.g. signature with one shared key on the server-side, for use within one application

E.g. signature with a public/private key pair, for use across applications

The standardized way to exchange session data

Part of a JSON-based Identity Protocol Suite

- Together with specs for encryption, signatures and key exchange

Used by OpenID Connect, on top of OAuth 2.0

JWT represents data, not the transport mechanism

The ***cookies vs tokens*** debate can be a bit confusing

Cookies are a transport mechanism, just like the **`Authorization`** header
Tokens are a representation of (session) data, like a (session) identifier

JWT tokens can be transmitted in a cookie, or in the **`Authorization`** header

Defining how to transmit a JWT token is up to the web application
This choice determines the need for JavaScript support and CSRF defenses

Modern applications typically use JWT in the **`Authorization`** header

Frontend JavaScript apps can easily put the token into the **`Authorization`** header
JWT tokens are easy to pass around between services in the backend as well

Reference: Dr. Philippe De Ryck

7 Best Practices for JSON Web Tokens

Neil Madden Jan 25, 2017

security jwt json

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#1 - Learn about the underlying security properties

JWTs are not necessarily easier than other mechanisms

They use a standardized format (JSON)

JWTs look simple enough at the surface, but they’re actually fairly complex

They can be deployed in various different modes

There’s a plethora of cryptographic options

Getting the desired security properties depends on making sane choices

No need to be a crypto expert, but you should know about HMAC, encryption, …

If libraries make them for you, do a sanity-check before using it

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#2 – Don’t go overboard

A piece of advice that applies everywhere: Keep It Simple

Make sure you really understand what you need

Select the simplest option to meet your needs

Concrete guidelines for using JWT tokens

Don’t store unnecessary data

Don’t encrypt if you don’t need confidentiality

An HMAC suffices for simple services

Public key-based signatures are useful for large, distributed setups

If you need JWT tokens on a simple service, an HMAC probably suffices

A shared key known by all servers that need to validate a JWT

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#3 - Plan for how you will manage your keys

JWTs depend on crypto keys for signatures (and encryption)

Key management is not an easy problem

A couple of questions that you want to think of up front

How will you go about using a new key?

What happens if a server gets compromised?

How many services share key material, and need to be updated?

Encryption and signature keys should be rotated frequently

Frequency depends on the usage, but this still needs to be taken into account

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#4 - consider using "headless" JWTs

JWTs are untrusted data and need to be verified before using them

But all of the data used to verify them is right inside the token (except for the keys)

In 2015, two vulnerabilities in most libraries allowed JWT forgery

#1: many libraries accepted JWTs with the “none” signing algorithm

#2: libraries could be tricked to use an RSA public key as the key for an HMAC

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

A JWT is a base64-encoded data object

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkaXN0cmluZXQuY3Mua3VsZXV2ZW4uYmUiLCJleHAiOjI0MjUwNzgwMDAwMDAsIm5hbWUiOiJwaGlsaXBwZSIsImFkbWluIjp0cnVlfQ.dIi1OguZ7K3ADFnPOsmX2nEpF2Asq89g7GTuyQuN3so

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

Header

```
{
  "iss": "distrinet.cs
          .kuleuven.be",
  "exp": 1425078000000,
  "name": "philippe",
  "admin": true
}
```

Payload

```
HMACSHA256(
    base64UrlEncode(header)
    + "." +
    base64UrlEncode(payload),
    "secret"
)
```

Signature

#5 - Careful when combining encryption / compression

Compression is very useful to reduce the size of a JWT

Important when you store a significant amount of data in there

If the data is sensitive, encryption is required to ensure confidentiality

There is a class of attacks against compressed encrypted data

You need to be aware that this is a potential problem

And talk to experts to fully understand what's going on

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#6 - Consider JWT lifetimes and revocation

Long lifetimes for JWTs with session information can be problematic

What if the JWT is stolen?

How will you handle revocation?

A lot of people are bashing JWTs for lack of revocation

But this is true for any kind of client-side session object, regardless of the format
Revocation with server-side sessions is easy, but hard for client-side sessions

Embedding unique IDs in a JWT and keeping a blacklist is often recommended

The blacklist needs to be checked during token revocation

But to blacklist you need to know all your JWT identifiers …

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

Side note on revocation

Why not associate a counter value with each user

Embed the counter into the JWT, and keep a copy in the database

More lightweight than keeping track of issued identifiers

Revoking JWTs for a user account is as simple as incrementing the counter

Validating a JWT requires a check against the stored counter value

A match means that the JWT is not revoked

A stored counter value that is higher than the JWT value means revocation

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

#7 - Read the Security Considerations!

The different aspects of JWTs are covered by various RFCs

RFC 7515: JSON Web Signatures

RFC 7516: JSON Web Encryption

RFC 7517: JSON Web Key

RFC 7518: JSON Web Algorithms

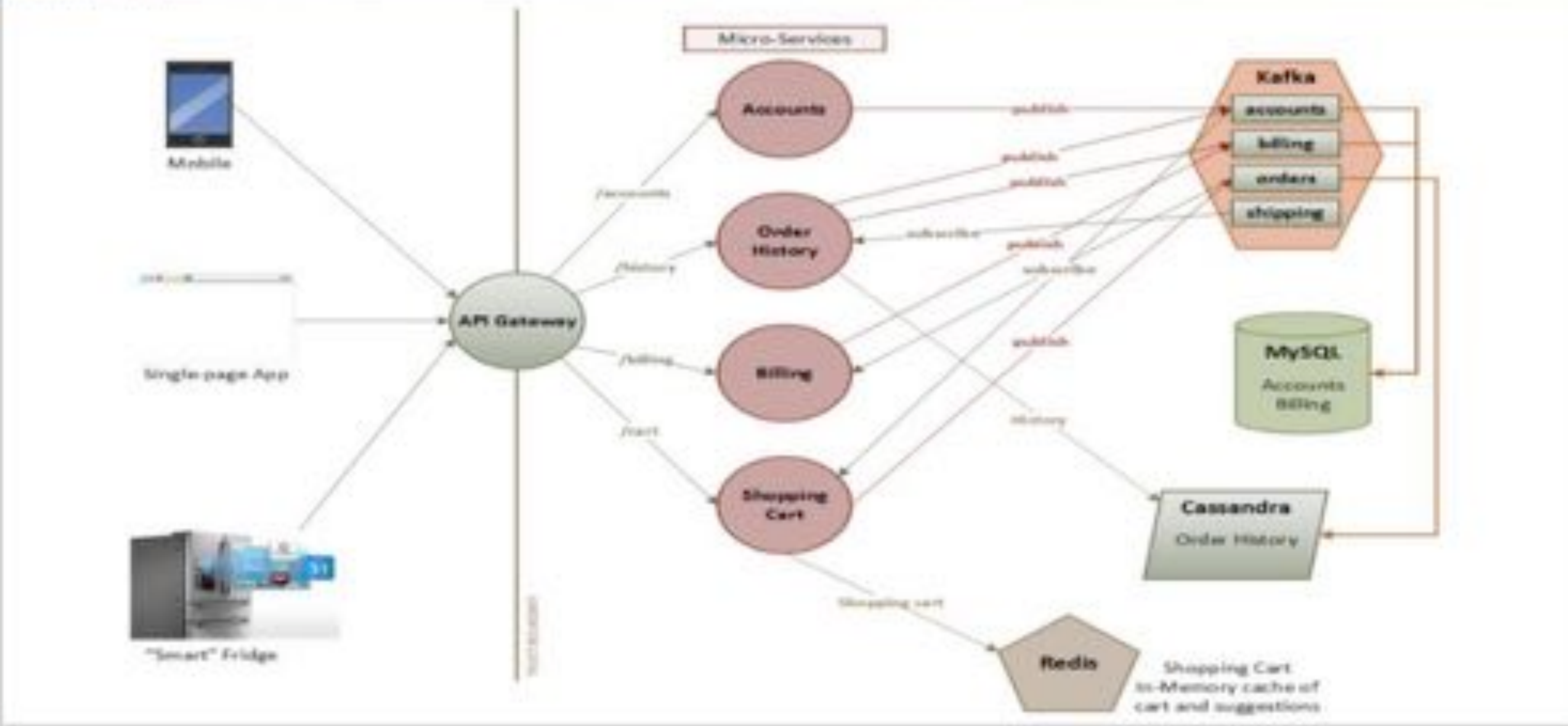
Understand the differences between headers, cookies, tokens, …

Make educated decisions about what to use where

Spread the word about what we have covered here!

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

A Simple Architecture



Reference: Jack Mannino

Token Binding

Token Binding

- First-party token binding:
  cryptographically bind tokens to a client
- Federated binding: cryptographically bind security tokens to a TLS connection

- https://tools.ietf.org/html/draft-ietf-tokbind-https
- https://tools.ietf.org/html/draft-ietf-tokbind-protocol
- https://tools.ietf.org/html/draft-ietf-tokbind-negotiation
- https://tools.ietf.org/html/draft-ietf-oauth-token-binding
- http://openid.net/specs/openid-connect-token-bound-authentication-1_0.html
- https://tools.ietf.org/html/draft-ietf-tokbind-ttrp

It’s been a pleasure.

jim@manicode.com