Ole Lensmar, @olensmar

API SECURITY TESTING

What do these companies have in common?

Their APIs have been hacked!

If you wanted to hack an API…

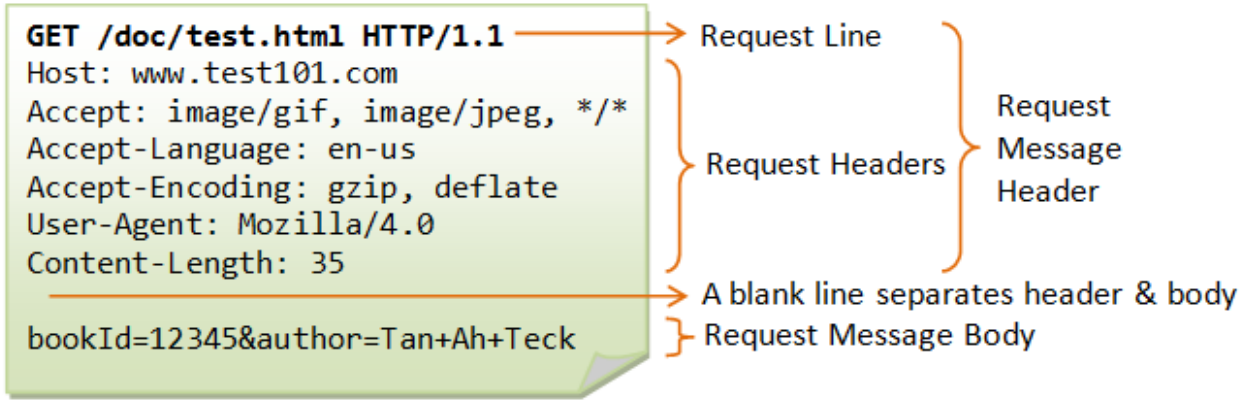
HOW WOULD YOU DO IT?

Hacking an API –the basics

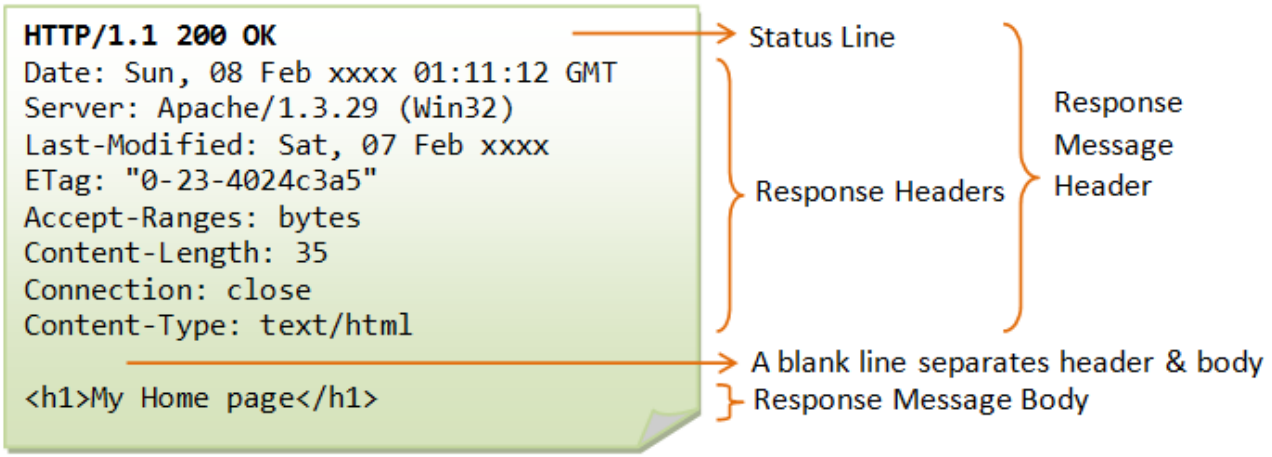
- REST and SOAP APIs predominantly use HTTP as their protocol
- Arguments are sent as part of the URL, as HTTP Headers or in the request body
- Message payload is predominantly JSON for REST and XML for SOAP

Understanding HTTP Transactions

Request



Response



Security Standards for Web APIs

- SSL commonly used for transport-level encryption
- Message level encryption and signatures:
  - SOAP/XML: WS-Security and related standards
  - REST: JSON Web Algorithms
- Authentication
  - SOAP: WS-Security/SAML
  - REST: Oauth 1 + 2, OpenID Connect, SAML, custom

So, we're hackers…

WHERE DO WE START?

API Attack Surface Detection

- We want to know as much as possible about an APIs endpoints, messages, parameters, behavior
- The more we know – the better we can target our attack!
- Unfortunately though – an API has no “UI” that can show is the attack surface

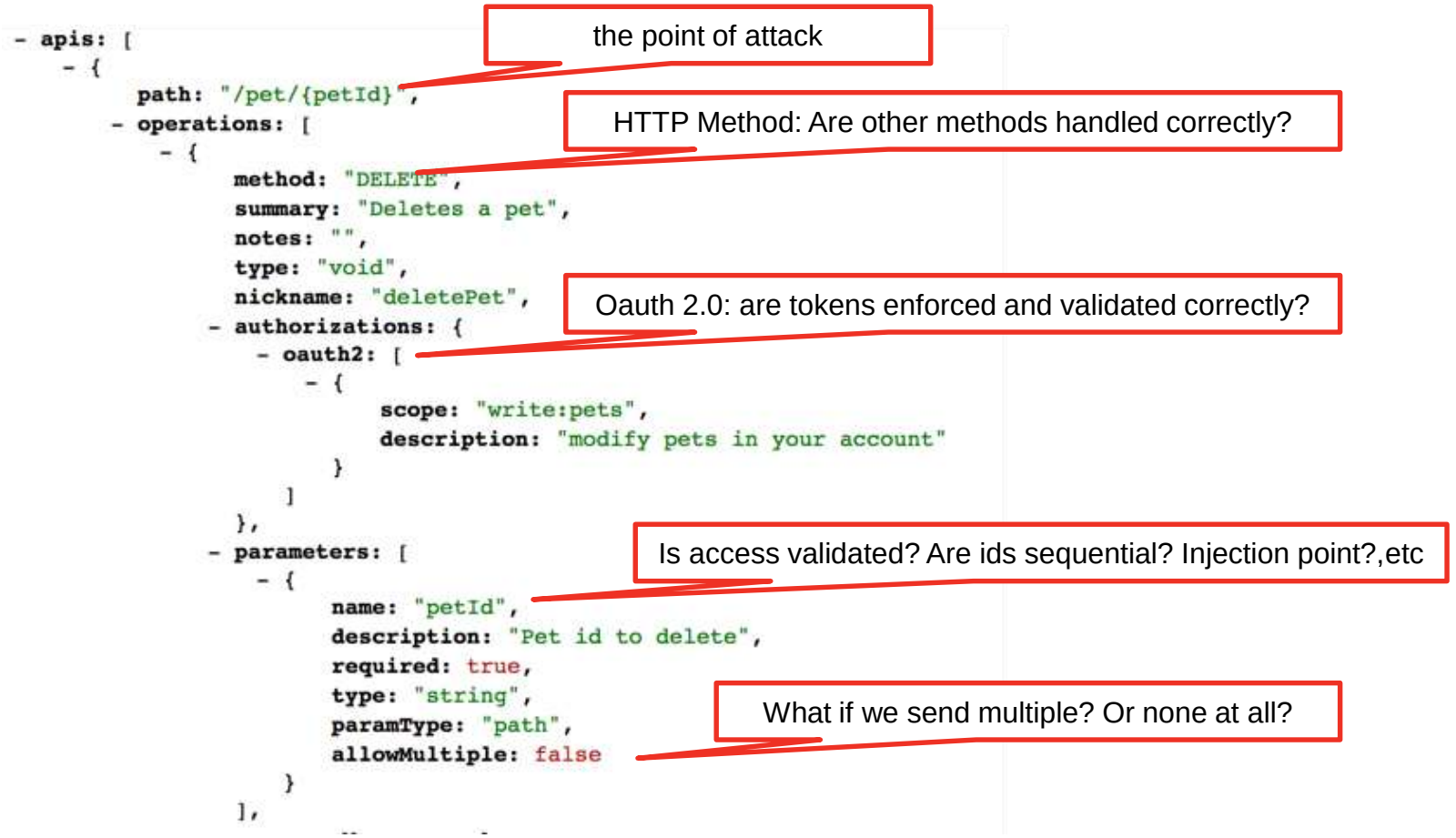
Attack Surface Detection: API Metadata

The more we know, the easier it is…

- api-docs.json
- WSDL/XML Schema
- Swagger, RAML, API-Blueprint, ioDocs, etc
- Hypermedia (JSON-LD, Siren, etc)
- Documentation / Developer Portals

Choosing between usability vs hackability

Attack Surface Detection: API Metadata



Attack Surface Detection: Other Methods

- **Discovery**
  - Record traffic via proxy or network sniffer to record and “learn” an API
- **Brute force**
  - Try commonly used endpoints (/api, /api/v1, etc)
  - Use error messages to uncover possible paths

With an Attack Surface, we can…

- Fuzzing
- Injection attacks
- Invalid / Out-of bounds content
- Malicious content
- Cross Site Scripting
- Cross-site Request Forgery

Why Hack an API?

- Provoke error messages or responses that give us system details
  - Database names
  - File paths
  - Component versions
  - Etc…
- Find security holes that give us access to system resources
- Put the API in an unavailable or unstable state (DOS)

API Attack Methods

HOW DO WE TEST FOR THEM?

API Fuzzing

What is it?

- Send random content as input parameters
- Automation can help us send millions of permutations
- Recursive Fuzzing – try all possible values
- Replacive Fuzzing – try common attack vectors

How do we test for it?

- Create automated fuzz tests that validate response messages to:
  - Not conceal system information
  - Return correct error messages / status codes
- Run them for a *long* time
- Run them in parallel / as load tests

Injection Attacks

What is it?

- Using SQL, XML, Xpath, JSON, JavaScript etc, attempt to inject code that is executed where it shouldn't be
- Primary injection: code is executed on the server
- Secondary injection: code is executed by 3rd party
- Example:

```
"SELECT * FROM pets WHERE petID='" + petId +"'";
http://petstore.com/api/v1/pet/123
-> SELECT * FROM pets WHERE petID = '123'
http://petstore.com/api/v1/pet/' or '1'='1
SELECT * FROM pets WHERE petID = '' or '1' = '1'
```

How do we test for it?

- Understand how the API works:

  SQL? NoSQL? Other APIs?
- Use well known injection vectors – validate for "unexpected" responses
- For example: validate that login call does *not* log you in
- Automate security tests

Invalid / Out-of-bounds attacks

What is it?

- Send input that we know is invalid
  - Out of range numbers
  - Invalid dates
  - Invalid enumeration values
  - Invalid data-types / formatting
- Can be auto-generated if the API has “good” metadata

How do we test for it?

- Send input that you know is invalid
  - Out of range numbers
  - Invalid dates
  - Random enumerations
  - Etc.
- Validate for:
  - Not displaying system information
  - Returning correct error messages / status codes

Malicious Content

What is it?

- Where files/images are uploaded/”attached”; attempt to upload executable files/scripts/etc
- Exploit server side parsing of content
- Example of XML Bomb:

```
<?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>
```

How do we test for it?

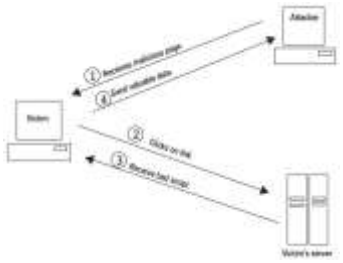
- Attempt to upload files that do no harm but indicate that they have incorrectly handled
- Both corrupt versions of accepted formats, and invalid formats.
- Validate that you get the right error messages!
- Test for parse vulnerabilities – use known Vectors

Be careful with this one…

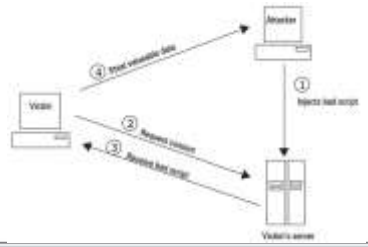
Cross-Site Scripting

What is it?

- **Reflective XSS:** Malicious script is included in link and “reflected” back to user



- **Persistent XSS:** Malicious script is injected into backend system and retrieved by user

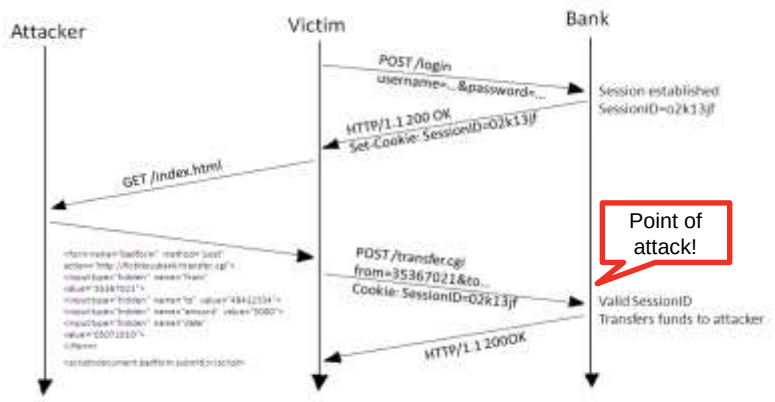


How do we test for it?

- In either case – create functional API tests that upload common attack vectors
- For **Reflective XSS tests**: validate that they are escaped (or removed) in the response
- For **Persistent XSS tests**: create end-to-end test that simulates the “other client” and validates correspondingly

Cross-Site Request Forgery (CSRF)

What is it?



How do we test for it?

- The common workaround is to include an unpredictable token with each request
- Create functional tests that validate:
  - The API call fails without that token
  - Tokens can not be re-used
- Run a fuzzing test on the token itself to validate that it can't be spoofed or bypassed

Insufficient SSL configurations

What is it?

- Eavesdropping on API traffic
- APIs should *always* use SSL – but sometimes they don't,
  or it isn't enforced (HTTP works also)
- Is the SSL certificate self-signed? (browsers will warn you – but code in a native mobile apps might silently allow access)

How do we test for it?

- Create simple tests that fail if HTTPS is not enforced
- Create simple tests that fail if certificates are self signed
- Run these in production as monitors – small system configuration changes/tweaks could have side effects

Insecure Direct Object References

What is it?

- For parameters that are IDs and seem to be sequential, try submitting IDs to get access.
- In Query Parameters, Headers and Message Bodies
- Call methods/operations that you shouldn't have access to

How do we test for it?

- Inspect actual API requests / metadata

You should question usage of direct object references!

- Create functional tests that validate authorization enforcement
- Combine with fuzzing or boundary tests on IDs

Other things to think about…

Bad Session/Authentication Handling

- Are session tokens re-used or sequential?
- Do session tokens timeout correctly?
- Are tokens exposed in unencrypted traffic?
- Are tokens added to URL when sending links?
- Are login endpoints restricted?

Bad security configuration

- Based on error messages and system information exposed by previous attacks
- Target all layers
  - Network
  - Server
  - Application
  - Client
- Examples:
  - exposed management consoles
  - directory listings
  - stack-traces
  - default passwords

So where does this leave us?

GENERAL CONCEPTS TO REMEMBER

API Security Testing requires you to

- Understand API Technologies
- Understand the API and its implementation
- Understand how Security Vulnerabilities work

Putting it into practice

- Automate Basic Security Tests using free tools
- Run automated Security Tests simultaneously as Load and Functional tests
- Stay up to date on Vulnerabilities

Resources

OWASP:
http://owasp.org

WS-Attacks:
http://ws-attacks.org/

Zed Attack Proxy (ZAP):
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Ready! API Secure:
http://smartbear.com/product/ready-api/secure/overview/

@olensmar
ole.lensmar@smartbear.com