The API Assessment Primer

Jason Haddix & Greg Patton
OWASP AppSecEU | May 21, 2015
Agenda

• Introduction
• Why API security matters
• Assessment considerations
• Common API vulnerabilities
• Takeaways
About me

Greg Patton
SAST Manager, HP Fortify on Demand

• Manage the static analysis testing team for HP FoD
• Nearly ten years of DAST experience with web & mobile apps
• Attended my 1st OWASP meeting on June 7, 2007 (Houston, TX)

hacker@hp.com
Why API Security Matters
APIs are everywhere

- Mobile apps
- Internet of Things (IoT)
- Service Oriented Architecture (soa)
- Enterprise thick-client apps
API insecurity

- New surface area = dangerous surface area
- Many API developers haven’t had security training
- Many assume that because back ends aren’t visited by end-users they are more secure (obscurity assumption)
API insecurity

Most APIs are vulnerable

- Analyzing any given API is likely to yield significant vulnerabilities
- The newer, more eager the shop – the higher the chance of issues
API Assessment Considerations
API testing approach

• Acquire information
• Map the API
• Capture runtime traffic
• Use automated scanners
• Manually test, test, test
What to collect pre-assessment

Ask customer for

- **Source code**
  - Static analysis & review
- **Documentation**
  - Regular user
  - Admin documentation
- **Valid request data**
  - Known-good param values
  - Order of function calls
Core toolset

Web proxy / HTTP editor
- ZAP proxy
- BURP suite pro

Web service testing tools
- SoapUI
- WSAttacker
- HP WebInspect
- Postman

Network capture tools
- Wireshark
- Echo Mirage

Browser extensions
- Chrome: Advanced Rest Client
- Firefox: Hackbar
API Mapping

**Fully map the API**, listing all methods and functionality at the start of an assessment

Examine:
- asmx
- /help & help docs
- WSDL (.NET)
- WADL (Java)
- Doxygen & help docs

Google
- inurl:wsdl site:example.com

Explore
- runtime operations
API Mapping | Testing

Try different HTTP Methods

- Don’t assume other verbs won’t work
- May discover hidden functionality
API Mapping | Testing

Try different content types and executions

- JSON vs. XML vs. Text
- REST vs. XML
Common API Vulnerabilities
Common API Vulnerabilities

- Broken Authentication & Session Management
- Information Leakage
- Not-So-Hidden Functionality
- Lack of Access Control
- Tampering & Trust Flaws
- Lack of Insecure Transport
- Injection Flaws
- Failure to Protect Keys
Authentication & Session | Concerns

- No authentication
- Insecure framework implementation
  - openID
  - oAuth
- Non-expiring session tokens
- Weak password complexity
- Lack of account lockout
- Lack of logout/session expiration mechanism
Authentication & Session | Testing

- Attempt to send requests with no authentication
- Review authentication scheme or framework
- Attempt to use simple passwords
- Attempt to use old session tokens
- Verify logout functionality truly expires sessions
- Weak password complexity
- Attempt to lock account
Authentication & Session | Protections

- Require authentication
- Require strong password
- Use up-to-date frameworks
  - Latest version of oAuth, etc.
- Ensure there is a way to logout / expire sessions
- Pay special attention to sensitive operations
- Use rate limiting to guard against Brute Force abuse
Information Leakage | Concerns

Often APIs respond with more data than required

- Apps returning all records instead of only needed or requested records
  - Particularly common in mobile applications
- Lack of data limiters
  - No limits on the number of requests a user can send
  - Brute force all records
Information Leakage | Concerns

2014 RSA Mobile App - Exposed Personal Data

• App designed for connecting with conference activities, viewing schedules, venue maps, etc.
• App used a web API to download information about every registered user of the application

• http://blog.ioactive.com/2014/02/beware-your-rsa-mobile-app-download.html
Information Leakage | Testing

- Review API responses
  - Do they return more data than what was requested?

- Try wildcard values
  - *, %, ?, space, etc.

- Review error messages
  - Do they reveal technical information?
  - Do they reveal enumeration flaws?
Information Leakage | Testing

Payload | "firstname". | "lastname". | "email". | _hash_
--- | --- | --- | --- | ---
46411 | Fortify | Tester | hacker@hp.com | 3d6f90060369fc8dad2792691e9268f1X
46412 | Tese bh test | 22@test.fr | 1853330a52a2cd504ed2f053366fd2d0aY
46414 | Ipadtest | frcm.fr | 2ca6bd73c6a6f3d0a3213bdc5cb5a2d7aX
46415 | Hendriks | tpej@gmail.com | c41b8f5a26892adb72ccf27243f0d8kg
46416 | Brooks | squar dotdesign.co.za | 338fe4403a64610f6481581a4ef42bc6N
46417 | Capstick | stick91@gmail.co.za | 5ae93505cb7ed96d1418325dd4732fru
46418 | Tyler | tye@gmail.com | 20dbc719a0f001543126fd953981f3e0YA
46419 | Yildirim | parker.fox@gmail.com | ab76a61b940f1f53fead9b15915d04f6o
46420 | Parker | parker.ally.com | 06afac6d30834ac15d2ae2dfs595767fMz
46421 | Engelbrecht | evinn@gmail.com | d43d7c2774889b1805c3d489ada0cecccDk
46422 | Almutawa | mutawa@hotmail.co.uk | 1a8551ee612a8648b993085106b765eB5
46423 | TEREYACOGLU | mail.com | 1e7a600c0b4ab86e3ca0875184f71af849f3
46424 | YILDIRIM | 34@hotmail.com | 6758e48f3f3615925fbc2b447bcQ
46425 | | | | 

HTTP/1.1 302 Found
Location: /Mobileapp/Customer Info.php/ MUTE/1.1

POST /Mobileapp/Customer Info.php/ MUTE/1.1
Host: www
Proxy-Connection: keep-alive
Accept: */*
Accept-Encoding: gzip, deflate
Cookie: EBBSESSID=4tp5pu2ojuv6pluj4qmid5d4g3; frontend-092084kfej7jau7jkebsh5hr10j; EBBSESSID=4tp5pu2ojuv6pluj4qmid5d4g3; incapses_700_174310-1llw70jmD4/vl5t/2lAmWYQAAAAANyx2llolNvxw1TD55yg=-; visid_incap_174310-ubD9TunJ694z4sFmDfYuNAYQAAAA
Content-Type: application/x-www-form-urlencoded; charset=utf-8

OWASP AppSecEU 15
Amsterdam, The Netherlends
23
Information Leakage | Protections

- Only return requested & needed data
- Review responses for sensitive information
- Review error messages
Hidden functionality | Concerns

API hidden functionality flaws are largely introduced due to faulty developer assumptions, i.e. not thinking like an attacker

• assume obscurity
• assume users will use functions only as intended
Hidden functionality | Testing

- Test different HTTP verbs
  - GET, POST, PUT, DELETE, etc.
- Check for API verbs
  - edit user, add user, delete user
- Review WSDLs, etc. for functionality not called at runtime
- Fuzz to find hidden operations
  - https://www.owasp.org/index.php/OWASP_SecLists_Project
Hidden functionality | Protections

- Ensure only required methods are exposed
- Ensure authentication scheme protects sensitive functions
Lack of access controls | Concerns

APIs don’t always verify the requestor is authorized for the target object

• Indirect Object References
Tampering & trust | Concerns

- Tampering with commands
  - Bypass client-side controls
  - Tampering with queries
- Incoming & outgoing data
- Malicious upload/download
Lack of access controls
Tampering & trust

| Testing |

Intercept & modify requests

- Modify parameters to attempt to access other data
  - Account numbers, User IDs, Order numbers, etc.

Intercept & modify responses

- Change the content available in mobile apps
- Bypass controls
Lack of access controls
Tampering & trust

Testing

Enumeration of User Orders
Lack of access controls
Tampering & trust

Testing

Enumeration of User Orders
Lack of access controls
Tampering & trust

| Protections |
- Validate Parameters
- Test for proper protection of sensitive information
- Review who has access to sensitive information
- Ensure only authorized users have access to sensitive information
APIs often lack sufficient protection of confidentiality and integrity of data in transit.

- Devices connected to untrustworthy networks
- Sensitive data transmitted in clear-text
  - No encryption
  - Encryption not enforced
- Poorly implemented SSL/TLS
Transport security flaws | Testing

- Review network traffic
- Check for cipher flaws & versions
  - SSLdigger, SSLScan, & other SSL testing tools
Transport security | Testing
Ensure data is protected in transit

- Ensure sensitive data is never transmitted in clear-text
- Turn on and enforce transport encryption
  - HTTPS everywhere
Injection | Concerns

- SQL injection
- Cross-site scripting
- Xpath injection
- XML DoS
- XXE – XML external entity
• Fuzz all parameters
• Utilize web scanners
• Manually tamper with requests
• Fuzz parameters and review results
• https://www.owasp.org/index.php/Projects/OWASP_SecLists_Project
Injection | Protections

- Validate all parameters server-side before generating output
- Do not assume clients will adhere to the API specifications
Key Management | Concerns

- Mobile app binaries
  - hardcoded
  - in manifest & .plist files
- Thick-client apps
- Online source code repositories
  - GitHub, BitBucket, etc.
$2375 Mistake

• Developer accidentally uploaded Amazon S3 keys to GitHub
  – Took them down & deleted all traces within 5 minutes
• Automated bot searching for API keys found them
• Amazon API allows users to spin up EC2 instances
• $2375 bill overnight

http://www.devfactor.net/2014/12/30/2375-amazon-mistake/

• Similar Amazon WS story

https://securosis.com/blog/my-500-cloud-security-screwup
Key Management | Testing

- Search for API keys
- Review online source code repositories for API Keys
- Run Strings on binaries & GREP for keys
- Review mobile binaries
  - Manifest files
  - .plist files
  - SQLite Databases
“Keys should be kept under a fake (virtual) rock outside your front door.” – R. Grosse
Takeaways
Takeaways

Adopt the attacker mindset

- Think like an attacker while evaluating your own APIs
- Identify places that developers likely made assumptions
- Attempt to take advantage of those assumptions
- As a developer, think in terms of *abuse* vs. just regular *use*
Takeaways

Go with an absolute least-privilege approach

- Do not expose any operations that are not needed
- Do not expose any data that is not required
Takeaways

Leverage available resources

- [https://www.owasp.org](https://www.owasp.org)

- **OWASP IoT Top 10**

- **OWASP Mobile Security Project**
Reach out

Greg Patton
hacker@hp.com

http://hp.com/go/fortifyondemand