HL7® FHIR® Security

Education Event

Fuzzing with FHIR

HL7 International | ONC Office of the National Coordinator for Health Information Technology

About Me

- Over 29 years in the design, architecture, and implementation of Healthcare IT solutions; with intense emphasis, in recent years, on information security & privacy for various Healthcare solutions used in the marketplace
- Engineer at heart - and passionate about information security & privacy
- Serving as co-lead on the FHIR at Scale Taskforce (FAST – *now an HL7 Accelerator) Security Team

Disclaimers

- Tools used in this presentation are intended to demonstrate concepts and in no way constitute an endorsement of any kind
- The techniques shared today are for *use only in environments (e.g., development/test) where developers have permissions and authorization*

Agenda

- Introduction
- Overview of Fuzzing
- Fuzzing Tools
- Demonstrations
- Conclusion, tips, and references

Introduction - About Fuzzing[1]

- Also known as fuzz testing, negative testing, or robustness testing
- Can be applied in many ways (e.g., files, binaries, protocols, payloads, etc.) – *for this presentation, the focus is on REST FHIR APIs*

This present will encourage you to utilize fuzzing throughout the entire SDLC to ensure your FHIR API implementations remain resilient and secure.

Why Implement?

- Why are the World's Biggest Companies Implementing Fuzz Testing[1]?

In 2019, **Google** used internal fuzzing to discover more than 20,000 vulnerabilities in Chrome

Microsoft uses fuzzing as one of the stages in its software development lifecycle

Fuzzing is an integral component of the **US Department of Defense (DoD)** software development lifecycle

Robust & Secure APIs

Robustness: focuses on the system's stability and reliability under stress or unexpected conditions.

Who else recommends?

- Standards & ISO Norms
  - https://www.code-intelligence.com/what-is-fuzz-testing#industries
- Such as …
  - **ISO/SAE 21434** Road Vehicles — Cybersecurity Engineering
  - **MDCG 2019-16** Guidance on Cybersecurity for medical device
  - **IEC 81001-5-1** Health software and health IT systems safety, effectiveness and security
  - **ISO/IEC 12207** Systems and Software Engineering – Software Life Cycle Processes

Realities of Fuzzing

- *"Penetration testing should include fuzzing APIs as a final step in your penetration testing efforts of an API"*[2] - Alissa Knight

Developer Call-to-Action

Your APIs will be fuzzed.

Rigorously perform this critical testing endeavor early in your SDLC.

Benefits of Fuzz Testing[3]

- Increased security
- Improved stability
- Enhanced compatibility (e.g., XML vs JSON)
- Early Bug Detection
- Cost-Effective Testing

Developer Benefits & Goals

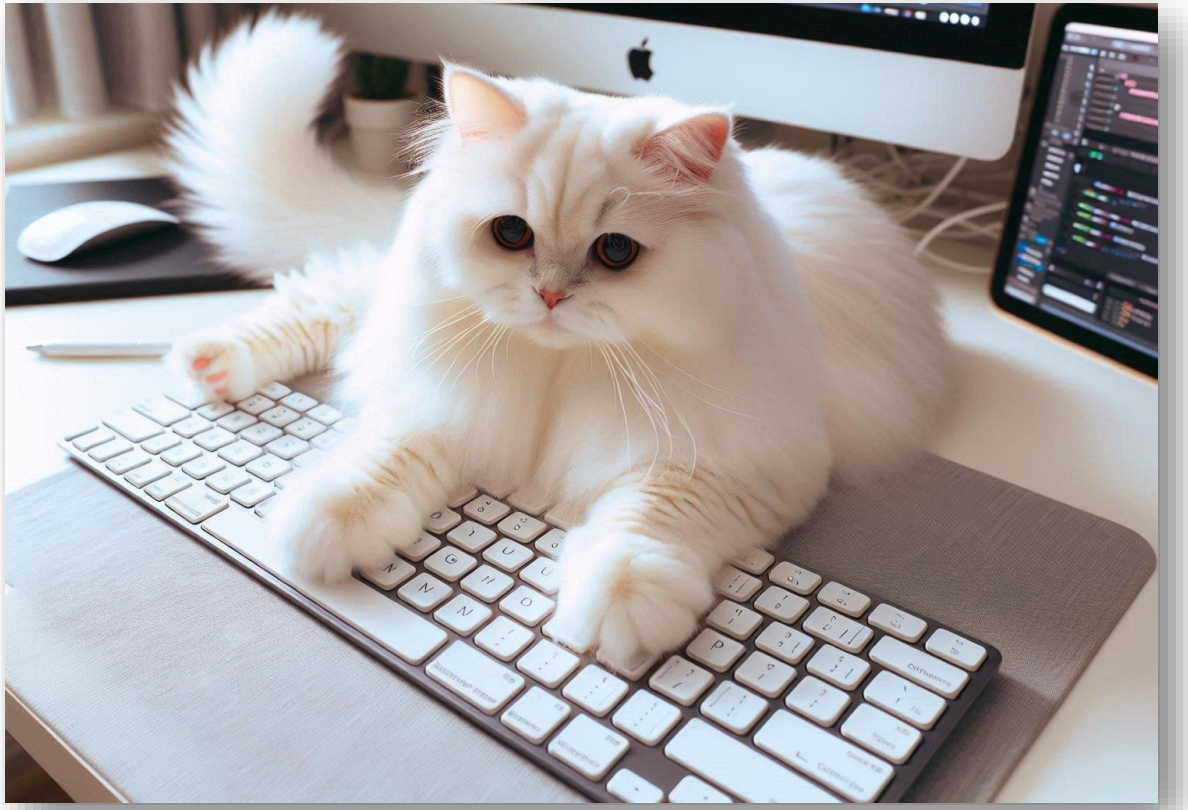
- Identify edge cases, corner cases, or boundary conditions that may cause unexpected or undesirable behaviors in the application

Discover and fix vulnerabilities that may not be detected by other testing methods, such as unit testing, integration testing, or code review

Frontend (UI) vs Backend

- Frontend Fuzz Testing – *<u>*Generally*</u> →User Experience & Usability
- *Backend Fuzz Testing - Security & Stability*
- Never trust data from a frontend or client; always validate server-side (i.e., backend)

**Invalid or malicious data not validated server-side or encoded properly when rendered by the browser could introduce security concerns.*



Fuzzing Types[4,5]

- Generation-based (from scratch) or mutation-based (modify existing inputs)
- Dumb (unstructured) or smart (structured), depending on whether it is aware of the input structure.
- Can be white-, grey-, or black-box, depending on whether it is aware of program structure.

Vulnerabilities

- Input validation flaws
- Authentication and authorization weaknesses
- Data parsing and serialization issues
- Business logic flaws
- Denial-of-Service (DoS) vulnerabilities
- Sensitive information exposure
- XML and JSON-based oriented bugs
- REST API or GraphQL vulnerabilities
- Unintended functionality

Ready – set – GO!

1. Environment & tools
2. Identify the target(s)
3. Define the fuzzing strategy (e.g., techniques, test types, etc.)
4. Prepare test cases
5. Instrumentation & Monitoring
6. Execute test cases
7. Evaluation & feedback loop (continuous improvement)

Fuzzing – Example Focus Areas

FHIR_BASE/**abc/Observation**?code=**http://loinc.org|79892-6**

URL Path

Query Parameter(s)

Test case: a very long string of random characters or numbers that may exceed the expected length or format of the input parameter

FHIR_BASE/**abc/Patient** Patient?given=valueGiven&**family=valueFamilyYyYyYyYy..repeated-1000x**

Fuzzing – Example Focus Areas (continued)

- **Payload** – A JSON object that has invalid syntax, missing fields, extra fields, or nested objects

```
{
  "resourceType"" : "Patient",
  "id" : "example",
  "identifier" : [{
    "use" : "usual",
    "type" : {
      "coding" : [{
        "system" : "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code" : "MR"
      }]
    },
```

Fuzzing – Example Focus Areas (continued)

- **Data elements** – malformed elements, questionable value given the context, erroneous data, etc.

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "12345" } ],
  "name": [ {
      "family": "Chalmers",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "2974-12-25"
}
```

Fuzzing – Example Focus Areas (continued)

- **Data elements** – malformed elements, questionable value given the context, erroneous data, etc.

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "12345" } ],
  "name": [ {
      "family": “\u0000Chalmers",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "1974-12-25"
}
```

Fuzzing – Example Focus Areas (continued)

- **Data elements** – malformed elements, questionable value given the context, erroneous data, etc.

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "12345" } ],
  "name": [ {
      "family": “\u000Chalmers",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "1974-12-25"
}
```

To be demonstrated

- Test Case: "family": “\u0000Chalmers"

Persistence successful; problems server side when the rendered format is XML

GET FHIR_BASE/Patient/ID_HERE/_history/1?_format=html/json → **HTTP 200**

GET FHIR_BASE/Patient/ID_HERE/_history/1?_format=html/xml → **HTTP 500**

- Test Case: "family": “\u000Chalmers"

JSON error fails to persist → HTTP 400

Data elements[7]

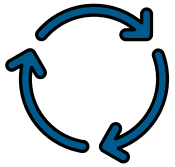
- Fuzzing tools and techniques can be *tailored to focus specifically on* ***data elements*** *of the given type*:
  - Simple / primitive types
  - General-purpose complex types
  - MetaDatatypes
  - Special purpose datatypes
- See https://hl7.org/fhir/R5/datatypes.html

FUZZING TOOLS

Tools - Data & CLI

Example tools that generate fuzz data (e.g., mutation) and execute tests via the CLI

- Radamsa: https://gitlab.com/akihe/radamsa
- Fuzz Faster U Fool (ffuf): https://github.com/ffuf/ffuf (example)
- *What tools do you use today?*



A mutation-based[5] fuzzer leverages an existing corpus of seed inputs during fuzzing. It generates inputs by modifying (or rather mutating) the provided seeds.

Tools w/ UI

- *OWASP ZAP*: an all-around testing tool for web applications that also appears helpful for APIs – *including fuzzing*
- *Burp Suite*: a general-purpose testing tool for web applications with proxy and scanning utilities – *test API fuzzing*; *supports fuzzing*
- *Postman*: a tool for testing and managing API requests; supports fuzzing

Reference: https://www.impart.security/api-security-best-practices/api-security-tools

Helpful Reference - Perilous Strings

- Big List of Naughty Values[6]
  - Evolving list of strings that have a high probability of causing issues
- Intended for use in helping both automated and manual QA testing

```
0xffffffff
0xffffffffffffffff
0xabad1dea
1234567890123456789012345678901234567890123456789
1,000.00
1 000.00
1'000.00
1,000,000.00
1 000 000.00
1'000'000.00
```

DEMOS

Demo Environment

- Use any OS capable of hosting a FHIR server (demo uses Ubuntu 22.04.3 LTS)
- FHIR Server w/ Sample Data - https://github.com/mitre/hapi-r4-eye-synthea
- Tools installed run fuzzing tests:
  - curl – quick demo to just try any strange data that comes to your mind as a developer☺
  - Radamsa - to generate fuzzed data (other top fuzzers, also see OWASP Fuzz Vectors)
  - Burpsuite Community Edition & Postman (w/ Postman Runner for automation)
- Used perilous values (BLNS) to inspire new test cases

The selection of the FHIR server for this demo was arbitrary. Any errors observed are intended to demonstrate the concepts and do not imply weaknesses for that FHIR server.

Fuzzing Demos (2 min each)

In real-world automation, any of the focal points can be fuzzed at great lengths to test security & resiliency at depth

- Radamsa
- URL parameters
- Query parameters
- Payload / Data Elements

Radamsa Screen Shot

- Generate fuzz values on demand & use with Postman or curl
- Incorporate into QA automation for continuous fuzzing

```
bsn:~$ echo "444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444" | radamsa
65535
11967695535
-18446744073709551617
444444444444444444444444444444444
9223372039002259456
◆-◆◆42949◆67297]1
```

Every invocation will result in different values

- *Tools intensify test scenarios*: ensure validation is robust and error notifications returned to the client are non-revealing (e.g., without stack trace)

DEMO CURL

URL Parameters

- Demo Base URL: http://localhost:8080/hapi-fhir-jpaserver/fhir

```
curl FHIR_BASE/Patient/FUZZ_VALUE
```

- Notice the variation of errors (e.g., Java runtime, Web Server)
- Some outcomes are simple patient not found vs exceptions

URL Parameters – Fuzz Values

```
44%20or%2017%20or%2021
444444444444444444444444444
%D444444444444444444444444444
%%444444444444444444444444444
_444444444444444444444444444
444444444444444444444444444_
```

DEMO BURPSUITE

Burp Suite – URL Parameters

- Demo Base URL: http://localhost:8080/hapi-fhir-jpaserver/fhir
- Some of the values below were produced using Radamsa
- Notice how just changing a single byte (negative vs underscore) for some of these request change the nature of the response drastically

FHIR_BASE/Patient/FUZZ_VALUE

URL Parameters – Fuzz Values

```
-1459718740743934381843137242761681328355309536743368938226627833964769614393
1459718740743934381843137242761681328355309536743368938226627833964769614393
_1459718740743934381843137242761681328355309536743368938226627833964769614393
□–□□42949  67297□1
```

DEMO POSTMAN

POSTMAN – Query Parameters & Payload

- Steps a developer with postman can follow to fuzz query parameters
  1. Prepare the Fuzzing List (save to local json file)
  2. Set Up the Postman Environment
  3. Configure Query Parameters
  4. Create a Data File (has fuzz parameters to be replaced during test)
  5. Use the Collection Runner
  6. Run the Fuzz Test
  7. **Analyze the Results**

Demo - Postman

- Test Case: "family": “\u0000Winters"

GET FHIR_BASE/Patient/ID_HERE/_history/1?_format=html/json → **HTTP 200**

GET FHIR_BASE/Patient/ID_HERE/_history/1?_format=html/xml → **HTTP 500**

- Test Case: "family": “\u000Winters"

Fails to persist → HTTP 400

Test Case: "family": “\u0000Winters"

POST http://localhost:8080/hapi-fhir-jpaserver/fhir/Patient (Persist then View)

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "654987" } ],
  "name": [ {
      "family": "\u0000Winters",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "1974-12-25"
}
```

http://localhost:8080/hapi-fhir-jpaserver/fhir/Patient/ 25557/_history/1?_format=html/json --> **HTTP 200**
http://localhost:8080/hapi-fhir-jpaserver/fhir/Patient/ 25557/_history/1?_format=html/xml --> **HTTP 500**

Test Case: "family": “\u0000Winters"

JSON

HTTP 200 OK

Response Headers

```
X-Powered-By: HAPI FHIR 5.0.2 REST Server (FHIR Server; FHIR 4.0.1/R4)
X-Request-ID: vTBmOxFKTS46cG4R
```

Response Body


```
1   {
2     "resourceType": "Patient",
3     "id": "25557",
4     "meta": {
5       "versionId": "1",
6       "lastUpdated": "2024-09-05T15:29:32.758+00:00",
7       "source": "#LUwN71QkRyygfliI"
8     },
```


XML

HTTP 500 Internal Server Error

Response Headers

```
X-Powered-By: HAPI FHIR 5.0.2 REST Server (FHIR Server; FHIR 4.0.1/R4)
X-Request-ID: SnlAGs0jr6iWX7zR
```

Response Body


```
1   <OperationOutcome xmlns="http://hl7.org/fhir">
2     <text>
3       <status value="generated"/>
4       <div xmlns="http://www.w3.org/1999/xhtml">
5         <h1>Operation Outcome</h1>
6         <table border="0">
7           <tr>
8             <td style="font-weight: bold;">ERROR</td>
9             <td>[]</td>
10            <td>
11              <pre>Failed to initialize STaX event factory</pre>
12            </td>
13          </tr>
```

Test Case: "family": “\u000Winters"

NOTE: This test case has three consecutive zeros instead of 4 (as in the prior example)

POST http://localhost:8080/hapi-fhir-jpaserver/fhir/Patient (Persist then View)

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "654987" } ],
  "name": [ {
      "family": "\u000Winters",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "1974-12-25"
}
```

\u0000Winters,1974-12-25

HTTP 400

```
"issue": [
    {
        "severity": "error",
        "code": "processing",
        "diagnostics": "Failed to parse request body as JSON resource. Error was: Failed to parse JSON encoded FHIR content
            character escape sequence\n at [Source: UNKNOWN; line: 5, column: 24]"
    }
]
```

Automation – Postman (Runner)

- Create collect with a request with {{place holders}}

```
{
  "resourceType": "Patient",
  "identifier": [ { "system": "urn:oid:1
  "name": [ {
      "family": "{{family}}",
      "given": [ "Peter", "James" ]
  } ],
  "gender": "male",
  "birthDate": "{{birthDate}}"
}
```

Postman Runner – CTRL + SHIFT +R

- Create a CSV with values the Postman Runner will incorporate into the request during automation runs

```
family,birthDate
\u0000Winters,1974-12-25
\u000Winters,1974-12-25
Winters,1974-12-251
Cats-9-Lives,2924-01-01
```

FUZZING RESOURCE DATA ELEMENTS

Fuzzing Resource Elements – Warm up

- Patient Resource
  - Name: Use different formats, special characters, and extremely long names.
  - Birth Date: Test with invalid dates, future dates, and edge cases like leap years.
- Observation Resource
  - Value: Use different data types, invalid values, and boundary values.
  - Code: Test with invalid codes, special characters, and extremely long codes.
- Medication Resource
  - Dosage: Use different formats, invalid values, and boundary values.
- Encounter Resource
  - Period: Test with overlapping periods, invalid dates, and future dates.

Conclusions & Tips

Conclusions

- Fuzzing is inevitable
- Fuzzing early in the SDLC is a MUST!
- Ensures robust & secure FHIR implementations
- Continuous is key (CI/CD)
- Zero trust – reinforces hardened data validation (*deserves special attention – see next slide*)

Tips

- Identify edge cases & corner cases
- Master the tools (UI & CLI)
- Reference BLNS (for developer fuzzing ideas)
- Go extreme (think corner cases)!
- Also test production logging & monitoring to identify possible fuzzing attacks or malformed data

Zero Trust! Trust but Verify

True Story

- I'm visiting a best friend's house, which boasts a beautiful cherry tree, as the sun sets.
- I ask about the taste? …
  - friend kindly suggests trying a few. States the tree was treated for pests this year, unlike last year.
- Why not? After eating several and brief chat, it was time to head home
- A few hours later – one word (*sick*)!

Always verify data; don't trust it implicitly!

THANK YOU!

QUESTIONS?

References

1. https://brightsec.com/blog/fuzzing/
2. FHIR API Whitepaper – Alissa Knight
3. https://fastercapital.com/content/Fuzz-Testing--How-to-Test-Your-Product-by-Providing-Invalid-or-Unexpected-Input.html#Benefits-of-Fuzz-Testing
4. https://testfully.io/blog/fuzz-testing/
5. https://coalfire.com/the-coalfire-blog/fuzzing-common-tools-and-techniques
6. https://github.com/minimaxir/big-list-of-naughty-strings
7. https://hl7.org/fhir/R5/datatypes.html