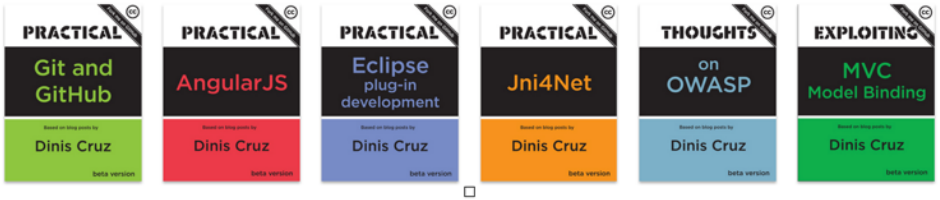
Surrogate Dependencies (in NodeJS)

@DinisCruz

London, 29th Sep 2016

Me

- Developer for 25 years
- AppSec for 13 years
- Day jobs:
  - Leader OWASP O2 Platform project
  - Application Security Training
    - JBI Training, others
  - Part of AppSec team of:
    - The Hut Group
    - BBC
- AppSec Consultant and Mentor
  - *"I build AppSec teams…."*
- https://twitter.com/DinisCruz
- http://blog.diniscruz.com
- http://leanpub.com/u/DinisCruz



A SURROGATE DEPENDENCY

Defintion

Surrogate

From Wikipedia, the free encyclopedia

A **surrogate** is a substitute or deputy for another person in a specific role and may refer to:

https://en.wikipedia.org/wiki/Surrogate

Surrogate model

From Wikipedia, the free encyclopedia

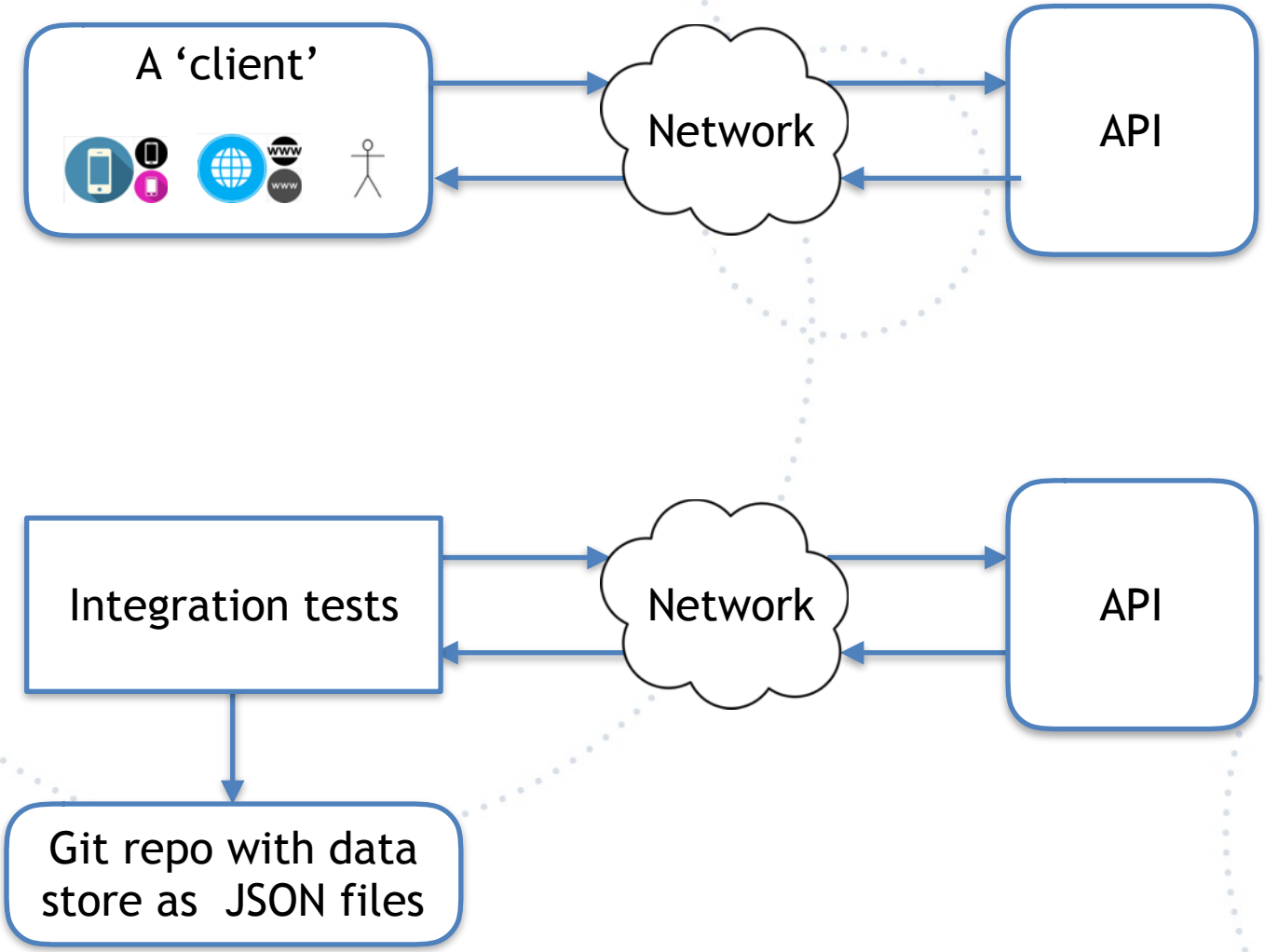
A **surrogate model** is an engineering method used when an outcome of interest cannot be easily directly measured, so a model of the outcome is used instead. Most engineering design problems require experiments and/or simulations to evaluate design objective and constraint functions as function of design variables. For example, in order to find the optimal airfoil shape for an aircraft wing, an engineer simulates the air flow around the wing for different shape variables (length, curvature, material, ..). For many real world problems, however, a single simulation can take many minutes, hours, or even days to complete. As a result, routine tasks such as design optimization, design space exploration, sensitivity analysis and *what-if* analysis become impossible since they require thousands or even millions of simulation evaluations.

https://en.wikipedia.org/wiki/Surrogate_model

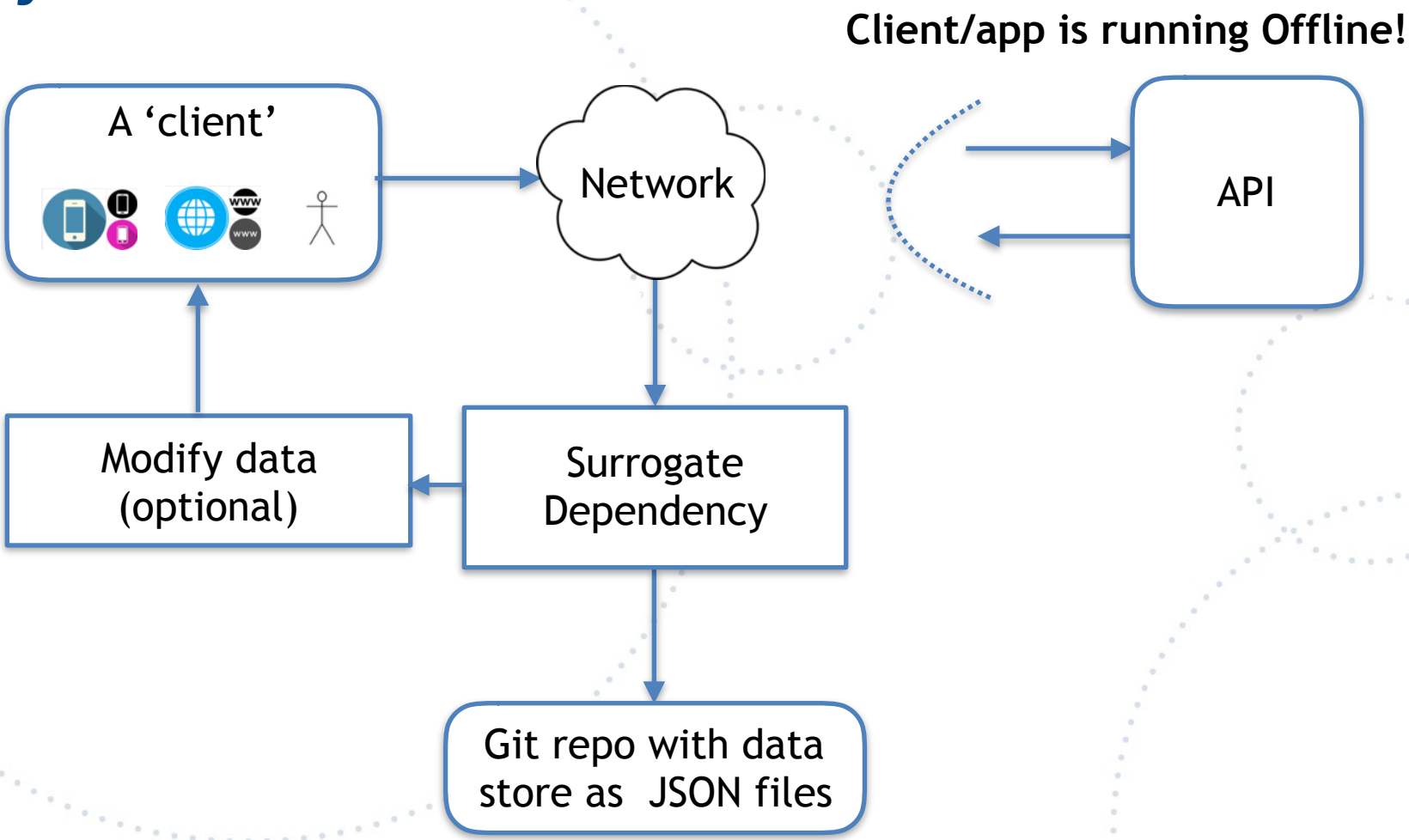
What is it?

- It tests the API and replays responses
  - Use integration tests to 'lock' the api used
  - Save responses in JSON format
  - Replay data to client
- Allow client to be offline

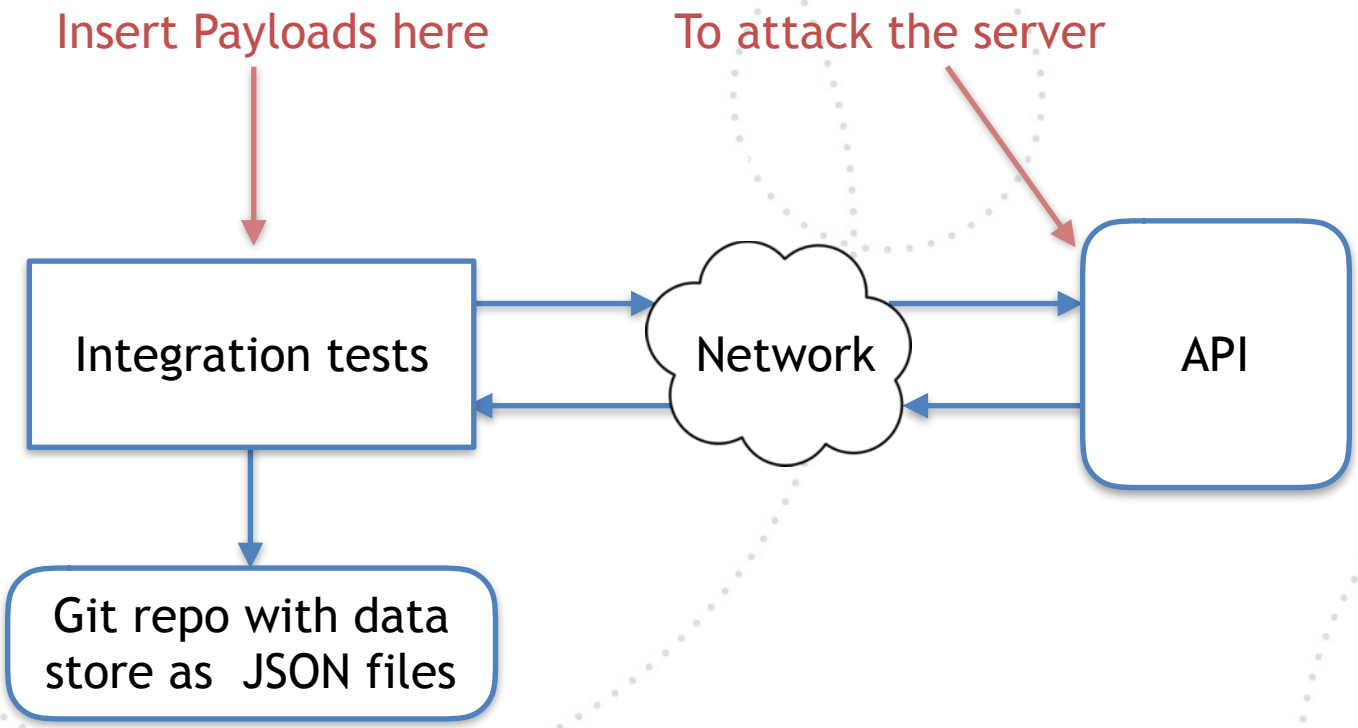
Locking the API using tests



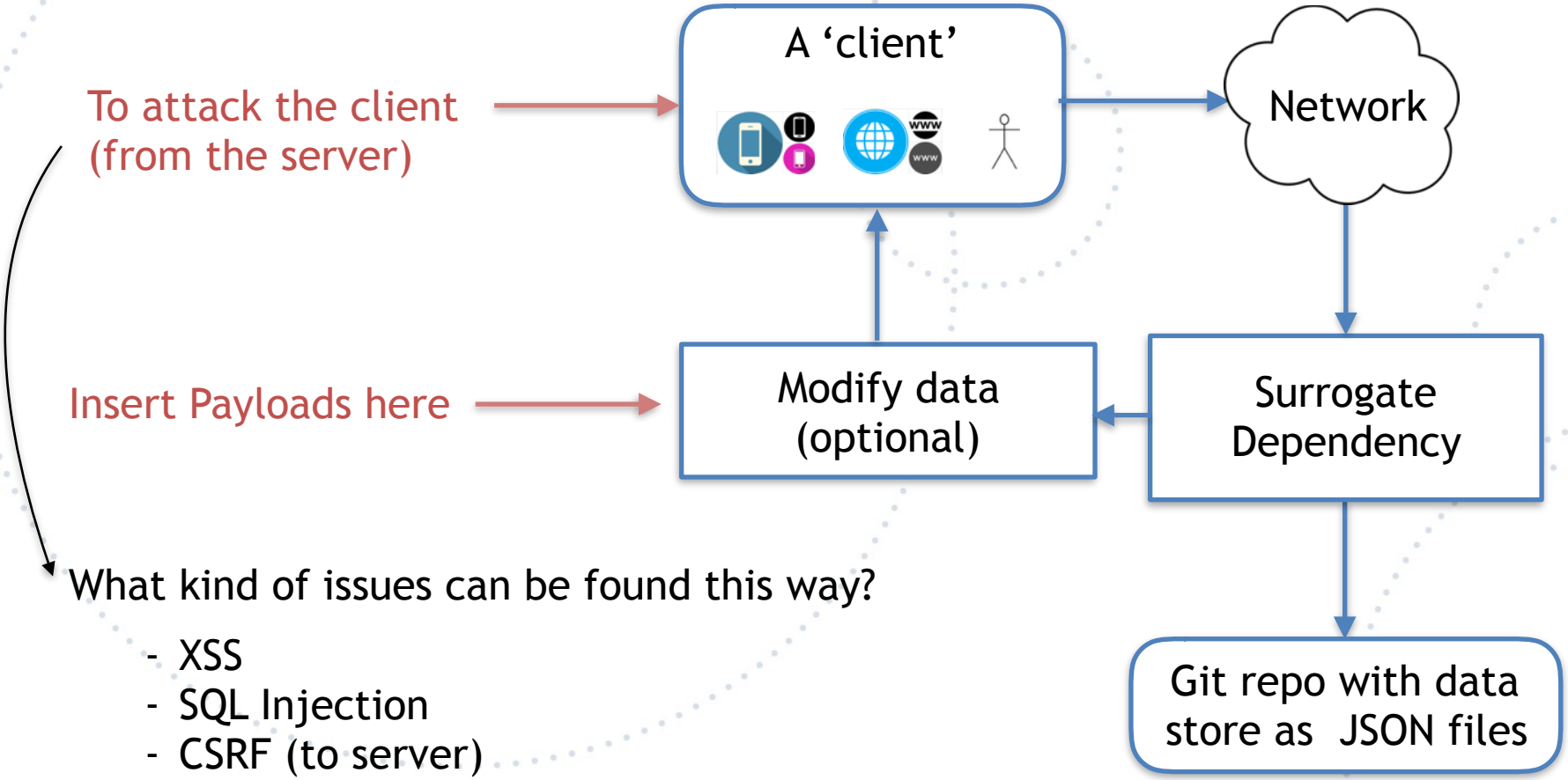
Replay stored JSON



Adding security tests (to server)



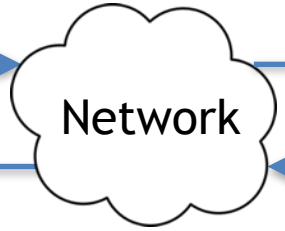
Adding Security Tests (from server)

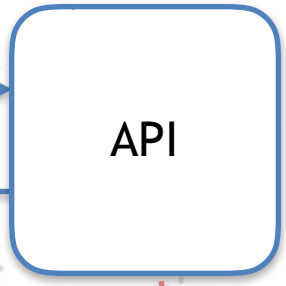


What kind of issues can be found this way?

- XSS
- SQL Injection
- CSRF (to server)
- DoS
- Steal Sessions tokens

Once you know where the client is vulnerable

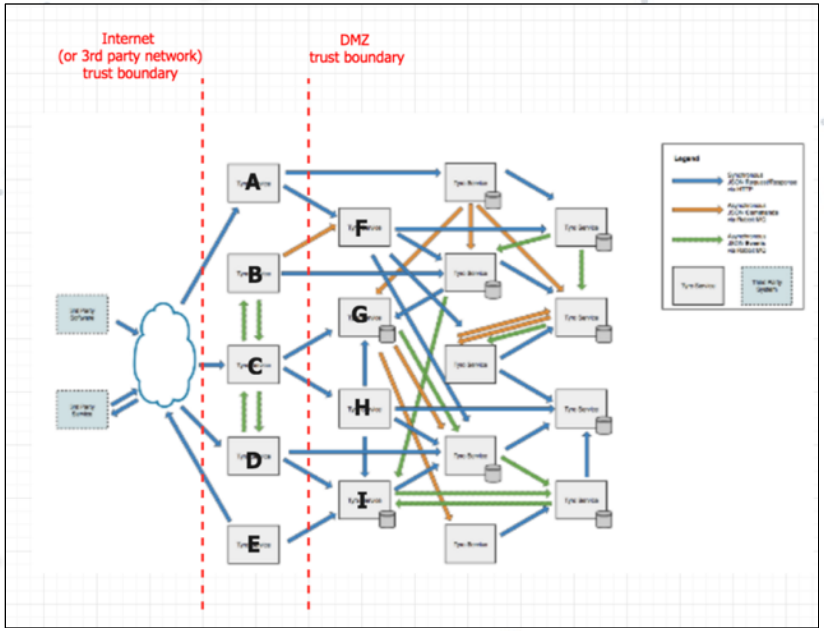




You 'ask' the API where did that data come from?

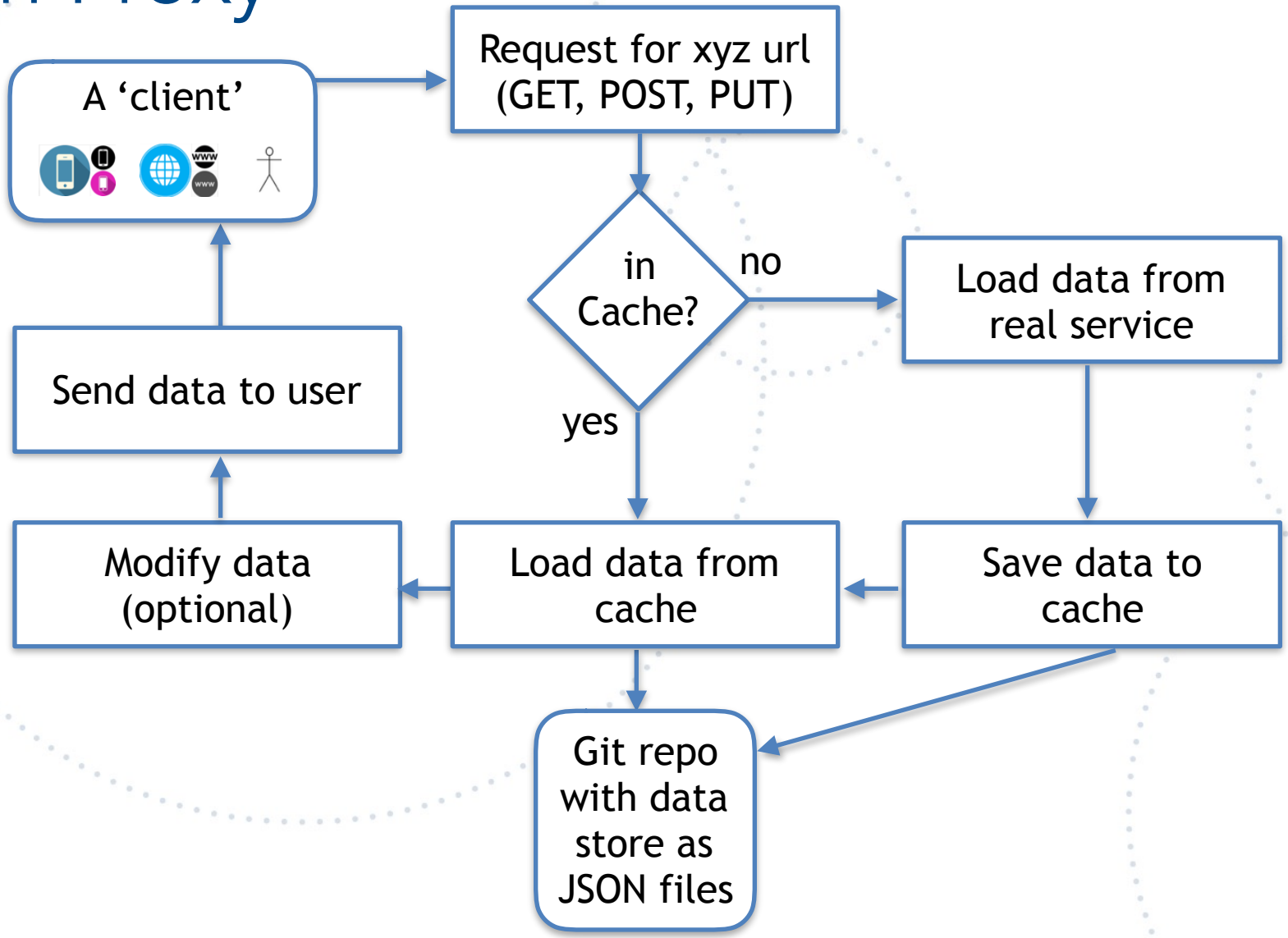
Once you know which data received from the server will exploit the client

… and follow the rabbit holes



Which might lead to and external source (i.e. attacker)

With Proxy


A 'client'
Request for xyz url (GET, POST, PUT)
in Cache?
no
yes
Load data from real service
Save data to cache
Load data from cache
Modify data (optional)
Send data to user
Git repo with data store as JSON files

Demo

Running a mobile app ‘offline’

BUILDING A TEST FRAMEWORK

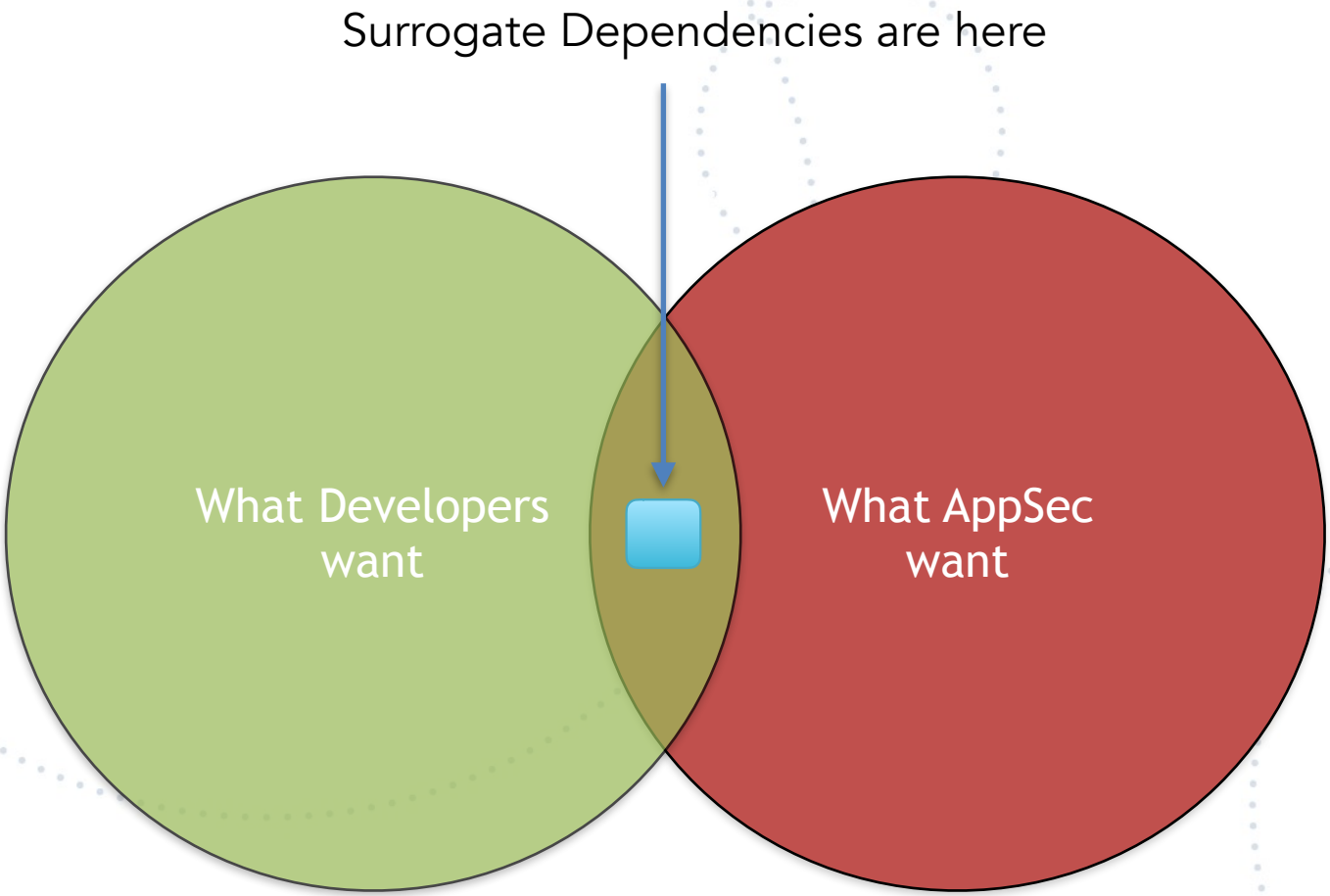
Problems that developers have

- Fragile dev and QA environments
- Inefficient TDD (specially for Integration tests)
- Lack of 'production-like' data
- Can't work offline
- Lots of manual testing
- Massive Versioning issues with dependencies (namely Web Services)
- Weak Schema contracts
  - remember that 'String' is not a type and Strings are not Strongly typed :)
- No/few dedicated micro services for their app

Key for AppSec is to make Devs more productive

- We need projects/activities that align AppSec needs with Dev needs
- The 'Surrogate dependencies' (which allows the app to run offline is one of those projects)

Aligning AppSec with Dev

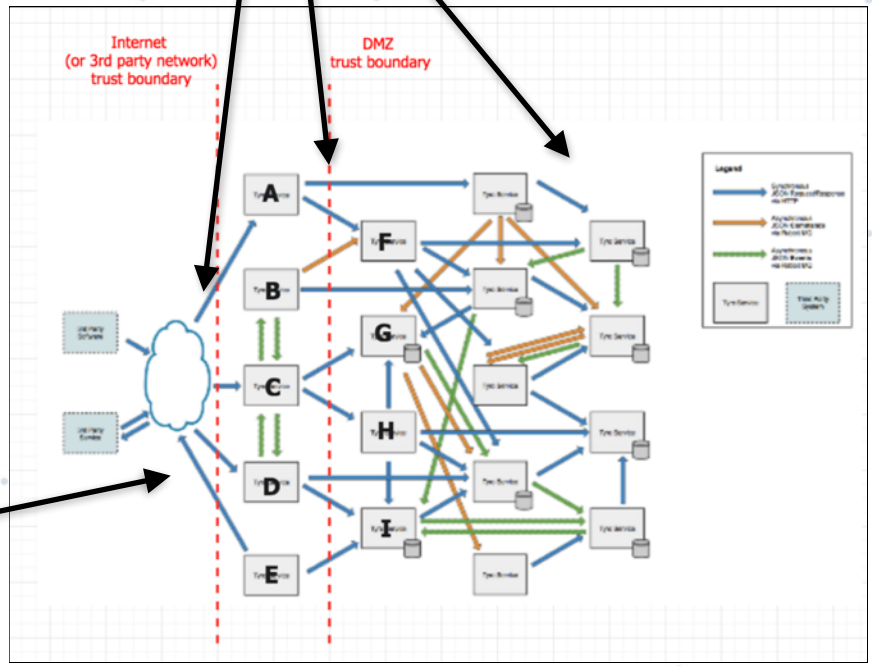


What do I mean by an Dependency

- Anything that is external to the application under development
  - Web Services
  - Message Queues
  - Inbound Http traffic (i.e. users)
  - Other protocols (SMTP, FTP)
- Basically all inputs (i.e. the real Attack surface)
- For now lets focus on Web Services (i.e. json, xml and html traffic)

Why a new Test Framework

- Be able to answer:
  - *What APIs are used at each layer?*
  - *What is their schema?*
    - *'string' is not a type*
    - *we need to ban strings*
  - *What happens if the server's response is is malicious*



http://www.grahamlea.com/2015/07/microservices-security-questions/

Answer Questions

- What happens if data is malicious:
  - from Client
  - from Server
  - to Server
- How can we have assurance of the Application properties
  - "…prove there are no exploitable XSS…"

Why NodeJS

- It's JSON Native
- Fast
- Effective TDD
- Powerful APIs
- JsDom

TECHNOLOGIES

JSDom

- Ability to simulate the browser DOM in Node
- Even supports complex frameworks (and event loops) like Angular
  - yes, you can run on NodeJS (i.e. server) Angular controllers, directives, services (with live Http Requests)

WallabyJS

- WallabyJS
  - real time unit test execution
  - real time code coverage

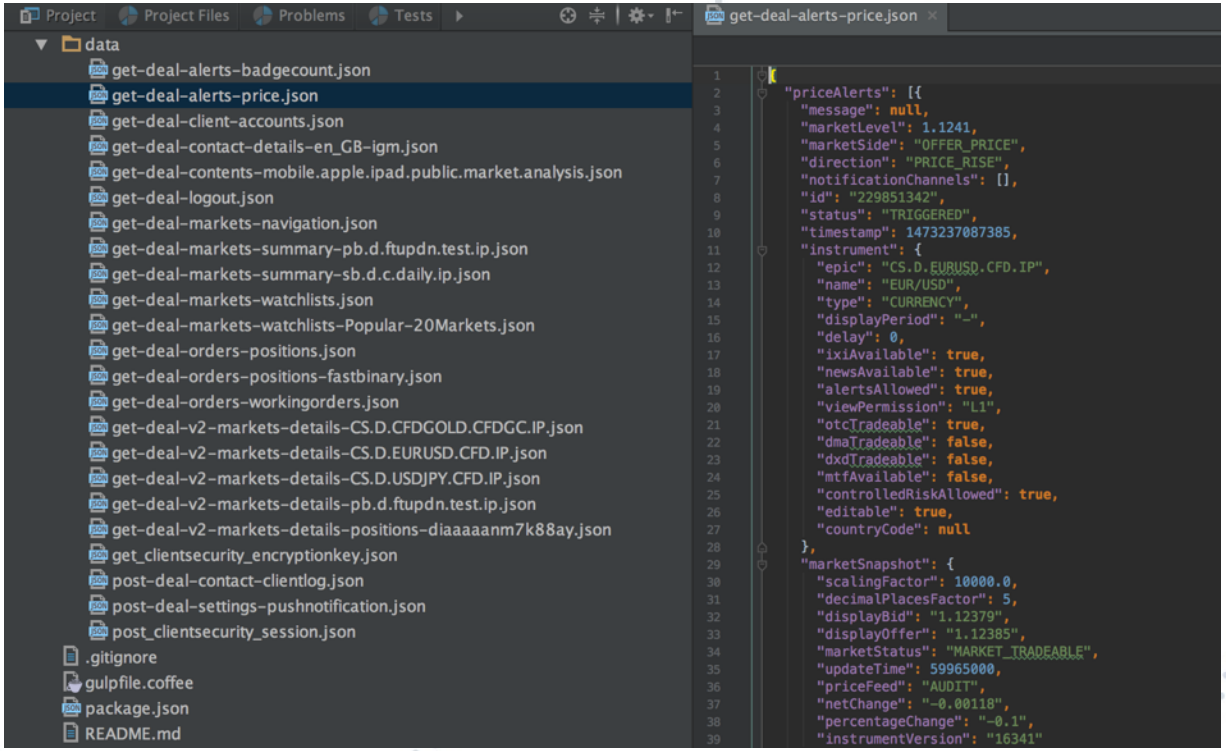
Unit/Integration tests

- That hit the live server and save the JSON

```
class Rest_Deal extends IG_Rest
  constructor: ->
    super()
    @.server =  'https://net-api.ig.com'
    @.prefix = '/deal'


  save_Data: (path, callback)->
    method = 'get'
    file = "./data/#{method}#{@.prefix.to_Safe_String()}#{path.to_Safe_String()}.json"
    @.get_With_Token @.prefix + path, (data)->
      data.json_Pretty().save_As file
      callback data, file

  client_Accounts: (callback)->
    @.get_With_CST @.prefix + '/client/accounts?preferences=true', callback

  contents_Market_Analysis: (callback)->
    @.save_Data '/contents/mobile.apple.ipad.public.market.analysis', callback

  order_Positions: (callback)->
    @.save_Data '/orders/positions', callback

  markets_summary: (callback)->
    @.save_Data '/markets/summary/PB.D.FTUPDN.TEST.IP', =>
      @.save_Data '/markets/summary/SB.D.C.DAILY.IP', callback
```

Git as database

- Content is stored as JSON on the file system



- Version control received data (using git diffs)

JSON

- Very good for data storage
- Powerful diffs (between test execution runs)
  - provide visualisation of dynamic data
  - identify inconsistent data
- Write tests against store JSON to confirm schema, data received
  - easy to identify bad server data deliveries (for example: multiple requests required, when one should be used)
  - Over supply of data (i.e. assets sent when they are not needed by client)
- Confirms 'happy paths' data
- Will be used for payload injections and DoS tests

Microservices

- Surrogate dependency is a model/template for dedicated micro-services
- Eventually Microservices should replace the original Surrogate dependency
- the Microservices will have their own Surrogate dependencies

JSDom

- Used to load html pages and render the Javascript
- Much better than selenium and PhatomJS since it is native to Node
- Test execution is super fast

XSS Proxy

- New module will add ability to act like a proxy
  - make requests to live server when request is not in cache
  - save response from live server in cache
- Idea is to auto-generate the tests for the requests recorded
- This will make it easier to create new 'surrogate dependencies' projects

Containers

- Best surrogates are the real code running inside a container
  - 2nd best solution is when the surrogates only exist on the 2nd level of dependencies
- Btw, if the app your are coding today is not designed to support containers (i.e micro services) in the near future
- Where you will be able to run dozens, hundreds or thousands versions in a separate container (aka Docker)
  - You are not aligned with the next major dev revolution (similar to git)
  - In a couple years, your app will be as 'legacy' as what you today call 'legacy'
  - key vision is that each 'user' should run in it's own container

Open source project

- XSS Proxy is already there
  - https://github.com/o2platform/node-ssl-strip
- Other code coming soon to OWASP
- Be involved :)
  - Your developers will love it and you will dramatically improve yours testing capabilities

Thanks, any questions

@diniscruz

dinis.cruz@owasp.org