Security in web application

- OWASP Top 10
- Dynamic analysis (Flaws inkection, Tools)

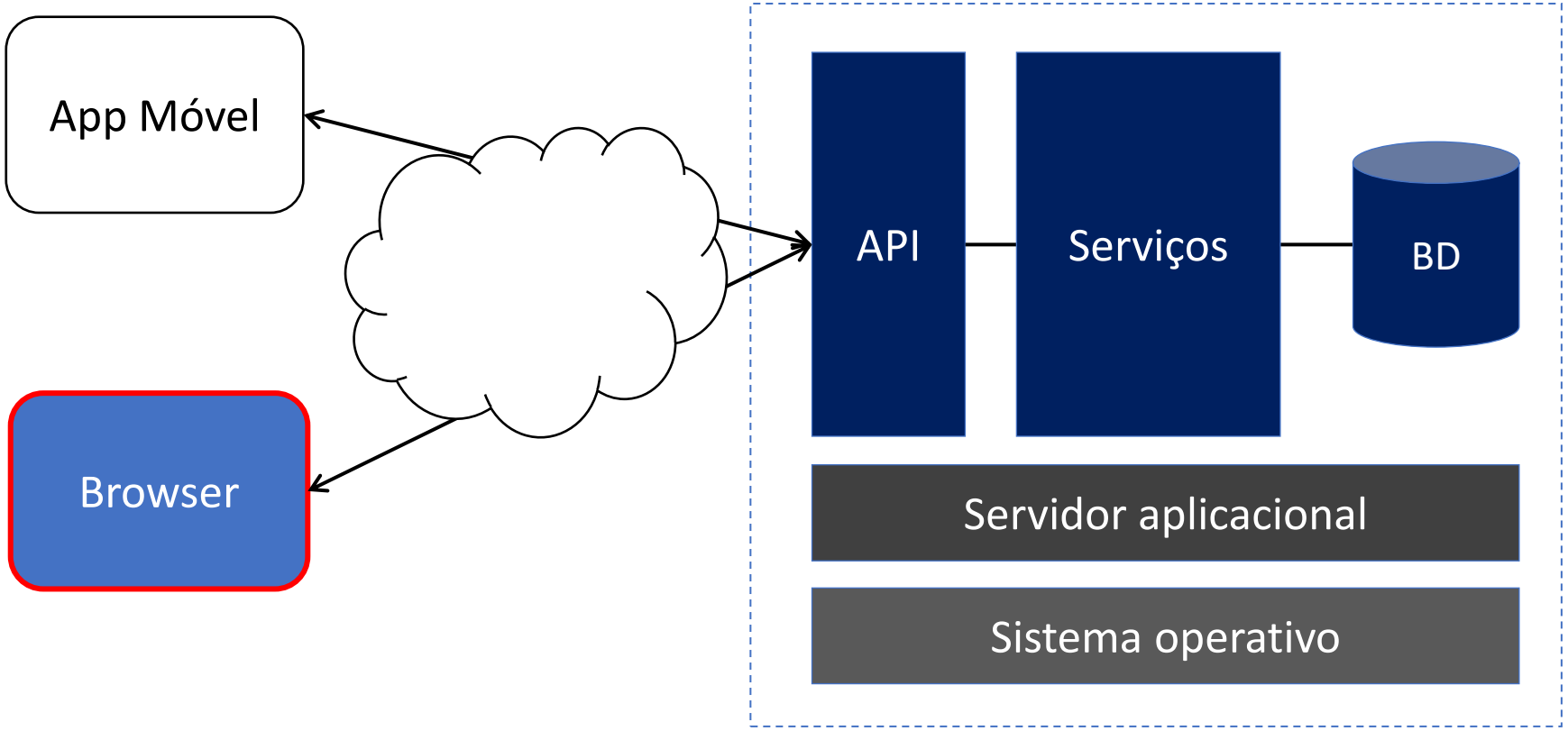
ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

Summary

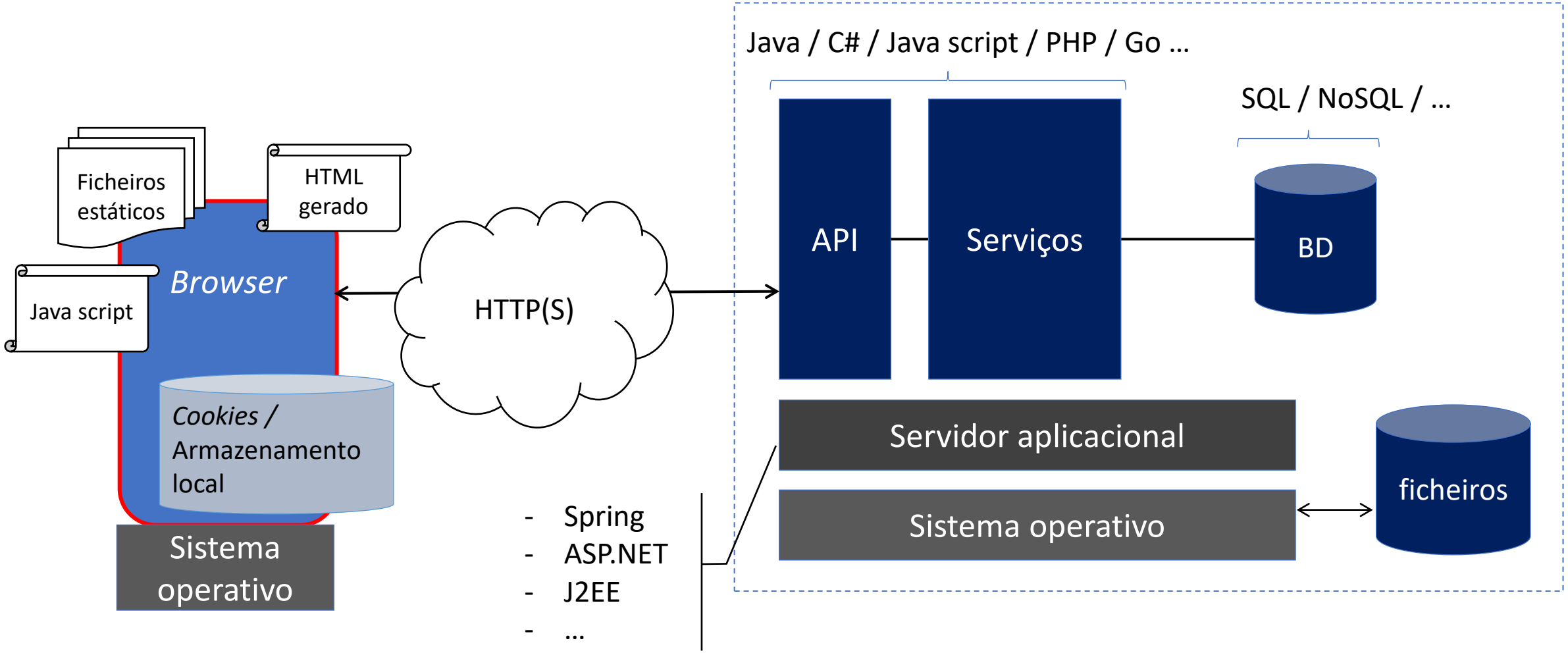
- Threats in web applications
- OWASP Top 10
- Attack injection
- fuzzing
- Practical case with the Zed Attack Proxy (ZAP) tool

Introduction

- Web applications are made up of several parts, running in different contexts
- The application code can be distributed, from the browser to the database

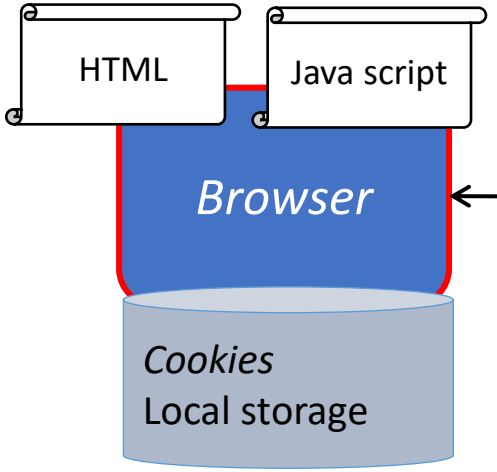


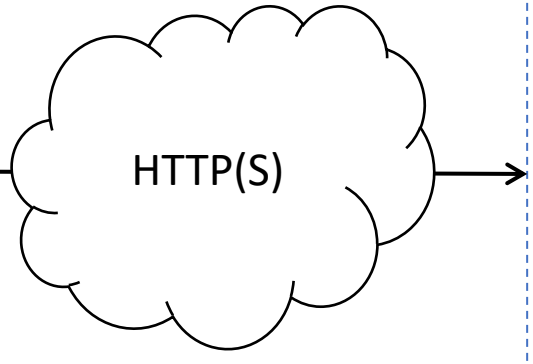
Tecnologias



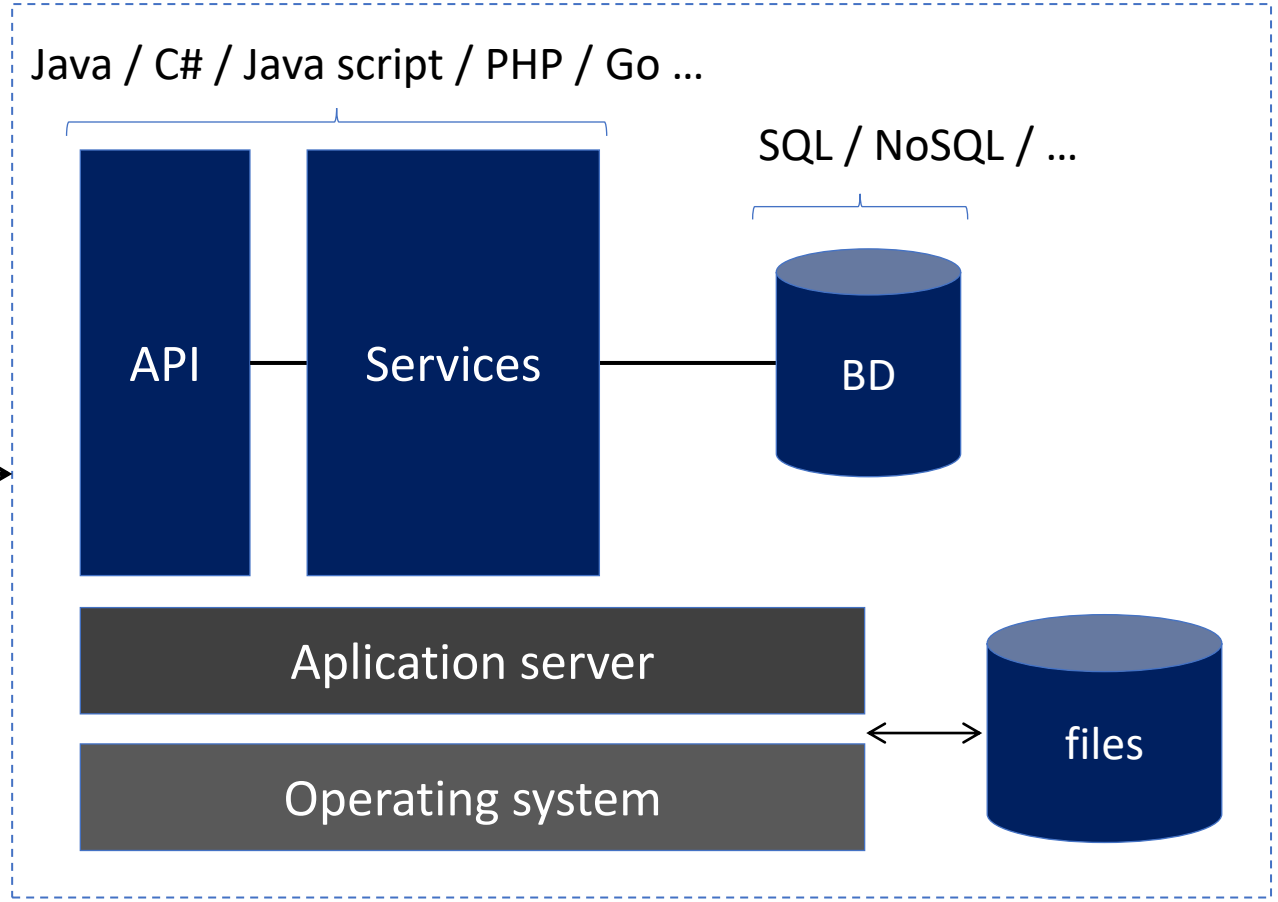
Ameaças

- Inject code in browser or BD
- forge authentication
- Bypass access control
- …





- man-in-the-middle
- Unsafe settings
- DDoS
- …



- OS vulnerabilities
- Application server vulnerabilities
- …

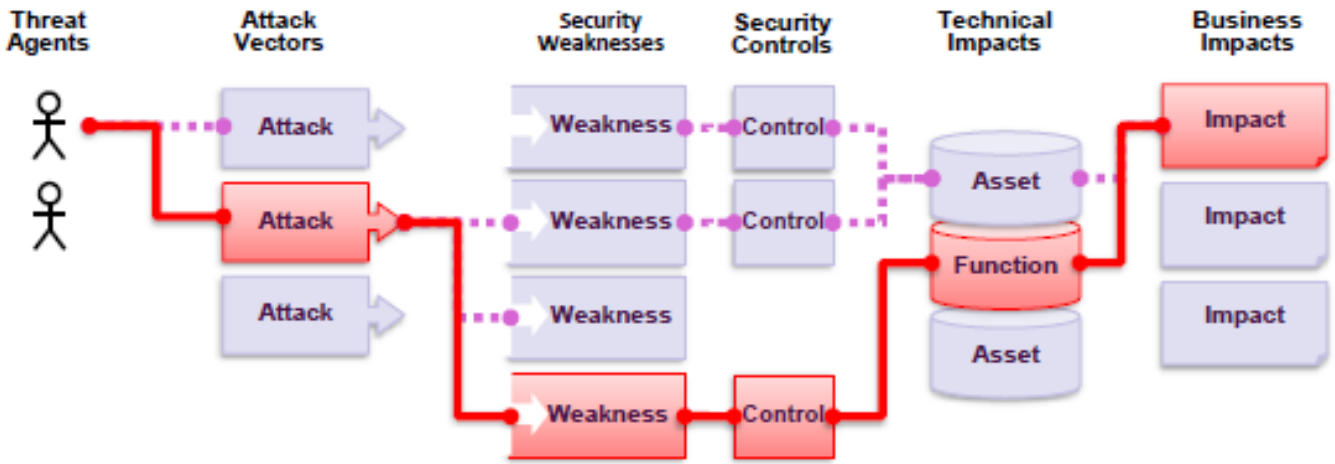
OWASP

- Open Web Application Security Project (OWASP)
  - Nonprofit Foundation to Improve Software Security

- Community keeps several projects open
  - Methodologies for identifying risks
  - Open source libraries for integration in different languages/frameworks
  - Projects to train security analysis skills in mobile websites and applications
- OWASP Top 10, https://owasp.org/www-project-top-ten/
  - Current version 2017
  - 2020 version in progress

OWASP – security risks

- The attacker can take several paths to harm the system
  - The different paths can be easier or harder. The impact on the business may also vary.
- OWASP Risk Rating
  - probability * impact
  - Exploitability
  - Prevalence
  - Detectability

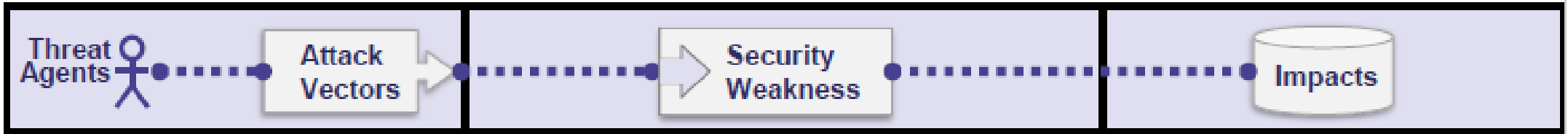


Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

Exemplo
para “Injection”

A1: *Injection*

- Exemplos: SQL injection, Serviços de directoria (ex: LDAP)
- Prevenção
  - Usar API segura que evita misturar comandos e dados
  - *Whitelist* (lista de aceites)
  - Object Relational Mapping (ORM)



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

A2: *Broken Authentication*

- Examples: Using common password lists, non-expiring sessions
- Prevention
  - Test new passwords against the most used ones
  - Using multi-factor authentication
  - Generate server-side random session identifiers

A3: *Sensitive Data Exposure*

- Examples: use of weak TLS settings, bad password storage
- Prevention
  - Automatically check server settings
  - Encrypt stored data / use strong hash functions
  - Rank importance of data according to legislation

A4: XML Enternal Entities (XXE)

- Examples: Application directly accepts and processes XML documents
- Prevention
  - Using less complex formats like JSON
  - Disable processing of external entities in XML
  - whitelist

A5: *Broken Access Control*

- Examples: Passing security controls by modifying the URL, Elevation of privileges by handling cookies / JSON web tokens
- Prevention
  - With the exception of public resources, Deny by default
  - Implementation and testing of access control mechanisms so that they can be reused

A6: *Security Misconfiguration*

- Examples: Active default accounts, latest updates not installed
- Prevention
  - Removing unused applications
  - Use automatic process to identify appropriate settings

A7: *Cross-site scripting (XSS)*

- Examples: Reflected, Stored, DOM-based
- Prevention:
  - Using libraries that encode data correctly
  - Encoding output prevents stored XSS

A8: *Insecure Deserialization*

- Examples: Direct serialization of application structures (eg for cookies)
- Prevention:
  - Use mechanisms to verify integrity
  - Do not accept serialized objects from unsafe sources

A9: *Using Components with Known Vulnerabilities*

- Examples: Unupdated systems, including OS, web app, database, libraries, …
- Prevention
  - Remove dependencies with components
  - Obtain components only from reputable sources

A10: *Insufficient Logging & Monitoring*

- Examples: Audit events are not logged, the application is not able to detect or alert on ongoing attacks
- Prevention
  - Logins and login failures are logged with enough context to identify suspicious or malicious accounts
  - Records are kept long enough for forensic analysis
  - Establish an incident response plan and recovery plan

API Security Top 10 - 2019

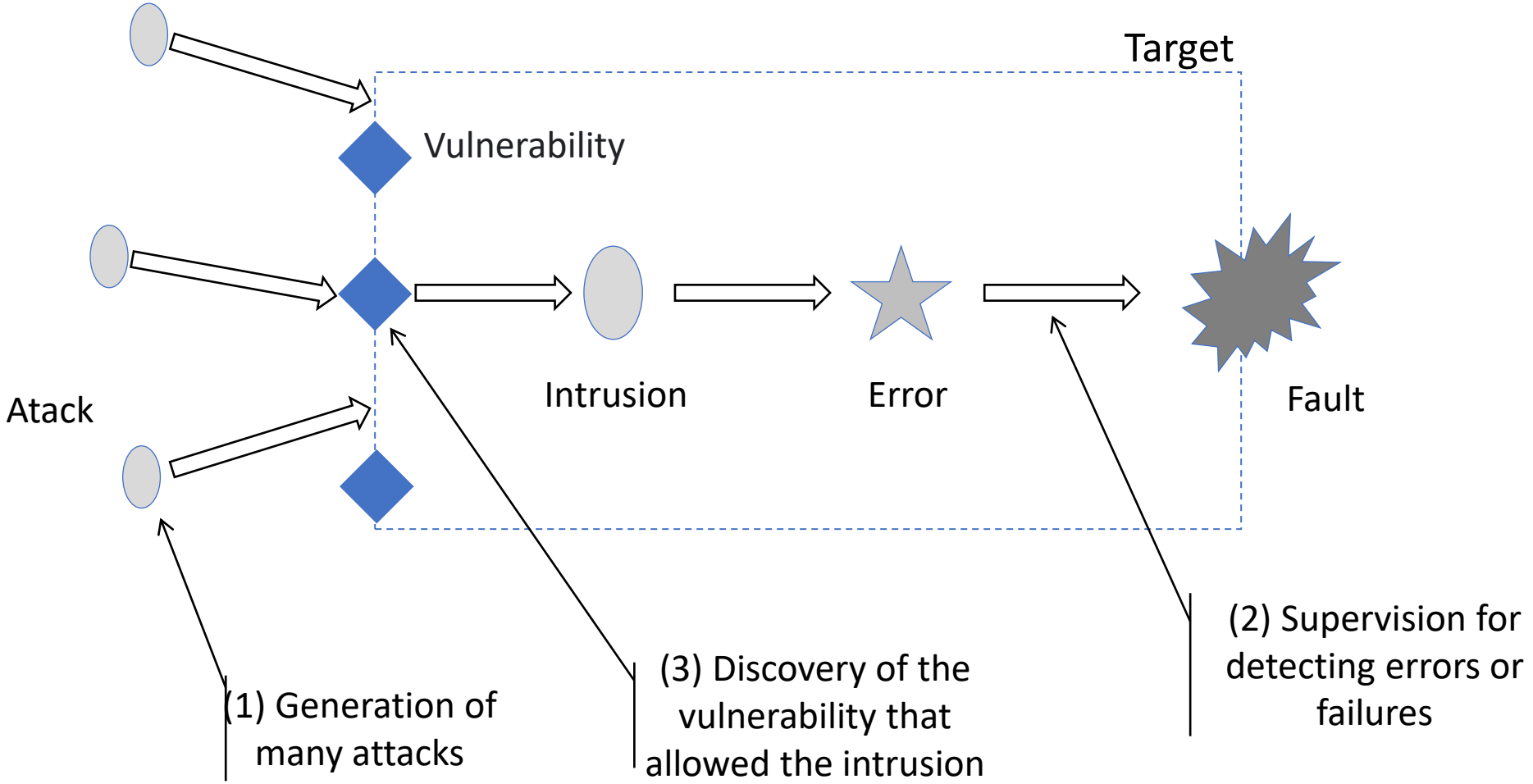
- https://owasp.org/www-project-api-security/
- Projeto em curso da OWASP que já produzui uma lista Top10
- Esta lista concentra-se em estratégias e soluções para compreender e mitigar as vulnerabilidades exclusivas e riscos de segurança de interfaces de programação de aplicativos (APIs).
- Exemplos
  - **API3:2019 Excessive Data Exposure**

    Em muitos casos as propriedades de um objeto são todas expostas, sem considerar sua sensibilidade individual, deixando para os clientes realizarem a filtragem de dados antes de exibi-los ao utilizador

Intrusion testing and searching for vulnerabilities in web applications

Attack injection, vulnerability scanning

Attack injection



Adaptado de https://ieeexplore.ieee.org/document/1633534

"Using Attack Injection to Discover New Vulnerabilities"

Fuzzing

«The original work was inspired by being logged on to a modem during a storm with lots of line noise. And the line noise was generating junk characters that seemingly was causing programs to crash. The noise suggested the term "fuzz".»

Bart Miller

- Techniques for finding faults by automatically injecting poorly formatted data
- The generation of data for fuzzing is an essential part of the process.

Fuzzers

- Fuzzers can be Recursive or Substitutes
- Recursives: Generation of different combinations of a given alphabet

```
http://www.example.com/00000000
...
http://www.example.com/11000fff
...
http://www.example.com/ffffffff
```

- Substitutes: Replaces the entry with a set of predefined entries

```
http://www.example.com/>"><script>alert("XSS")</script>&
http://www.example.com/'';!--"<XSS>=&{()}
```

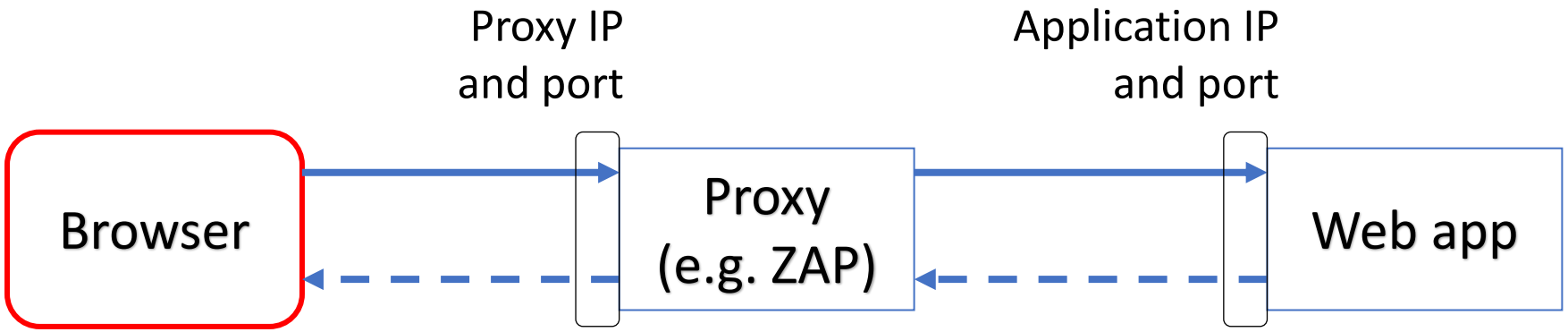
- Example: FuzzDB
- https://github.com/fuzzdb-project/fuzzdb

Varredores de vulnerabilidades

- Fuzzers and attack injectors look for unknown vulnerabilities
  - Vulnerability scanners look for known vulnerabilities
  - Run through vulnerability database
  - inject attacks
- They monitor the effect on the application trying to detect if it contains the vulnerability
- General requirements of a web vulnerability scanner
  - Identify specific sets of vulnerabilities present in public databases
  - Generate report for each vulnerability
  - Have an acceptable false positive rate

Proxies

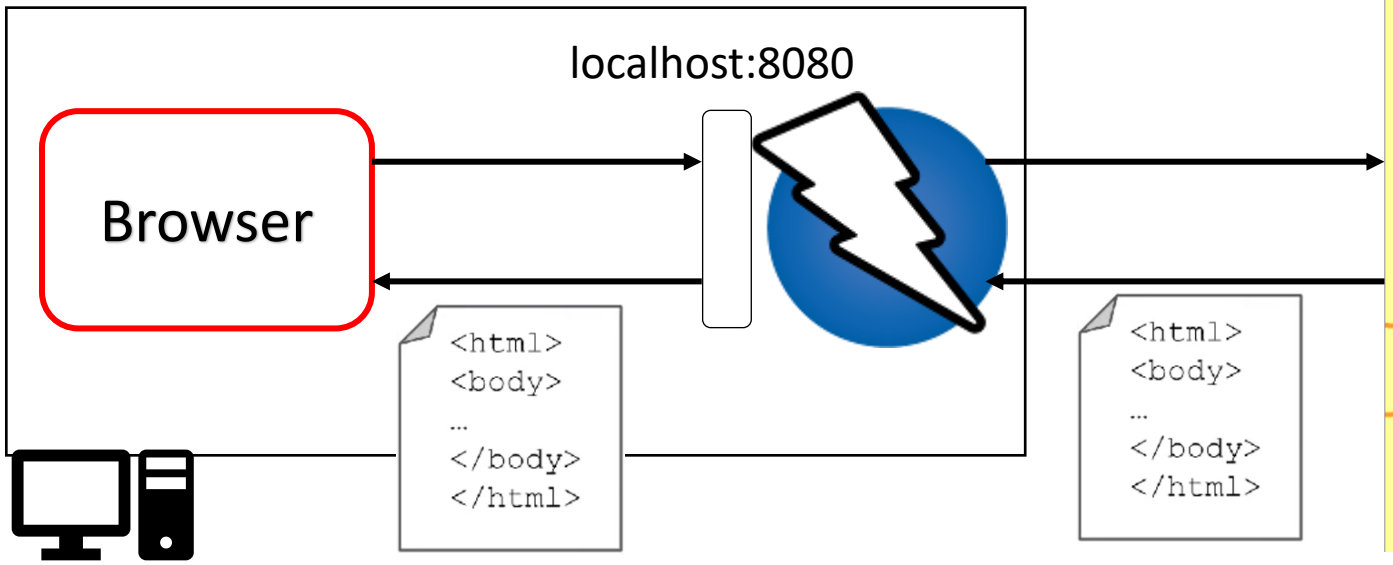
- Intersection of requests and responses
- HTTPS connections must be transparently intersected with authorized man-in-the-middle

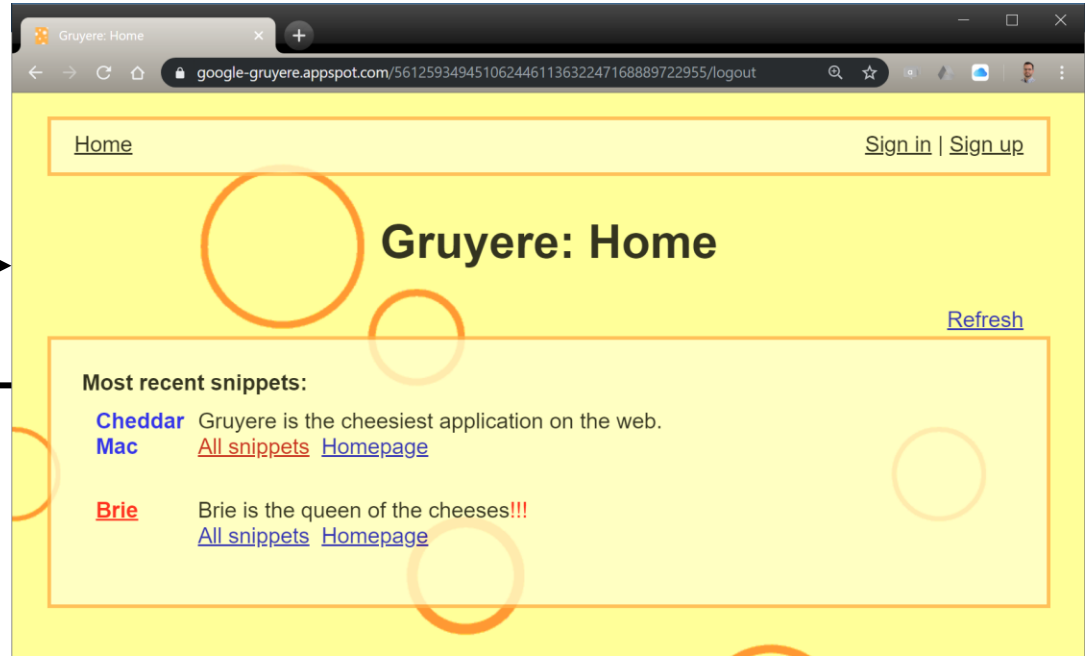


Tool ZAP

Zed Attack Proxy (ZAP)

- https://owasp.org/www-project-zap/

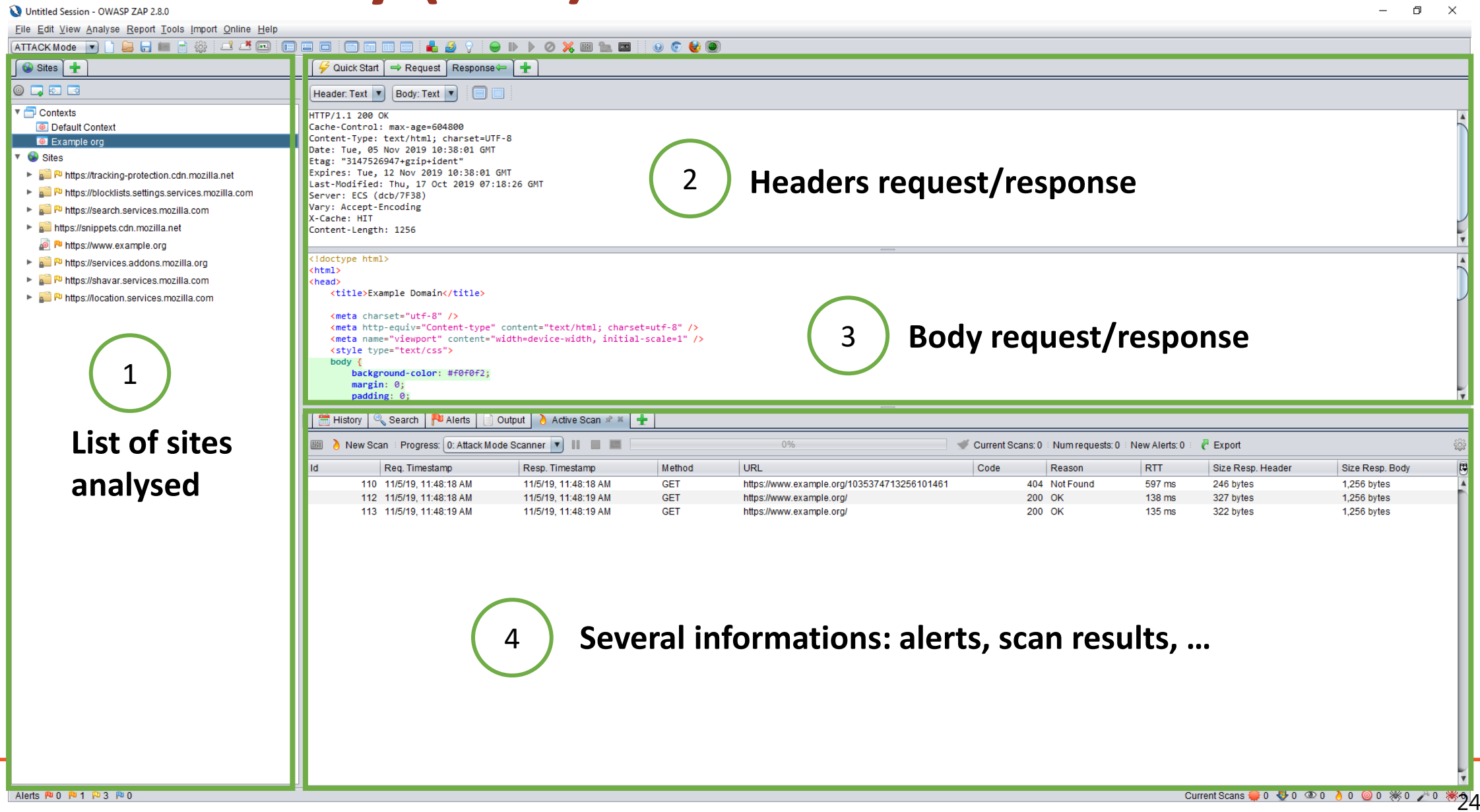




ZAP – operation modes

- Passive – passively analyzes all requests passing through it or generated by crawling components
  - In terms of penetration testing, this mode does not modify site data.
  - It is safe for general sites where you are not allowed to attack
  - Detects, for example, lack of critical headers or incorrect configuration of cookies
- Active – actively tries to find vulnerabilities using known attacks on selected targets
  - Attacks the website using known techniques to find vulnerabilities
  - This mode modifies data and may insert malicious scripts on the website.
  - **You can only run this mode for sites that we have testing permission for.**

Zed Attack Proxy (ZAP)



1

List of sites analysed

2

Headers request/response

3

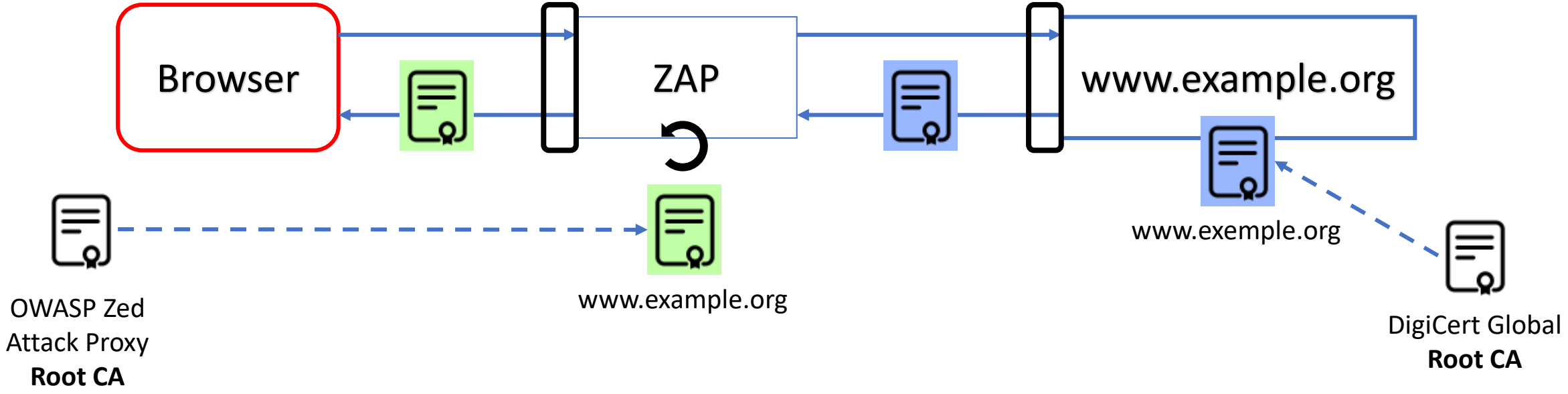
Body request/response

4

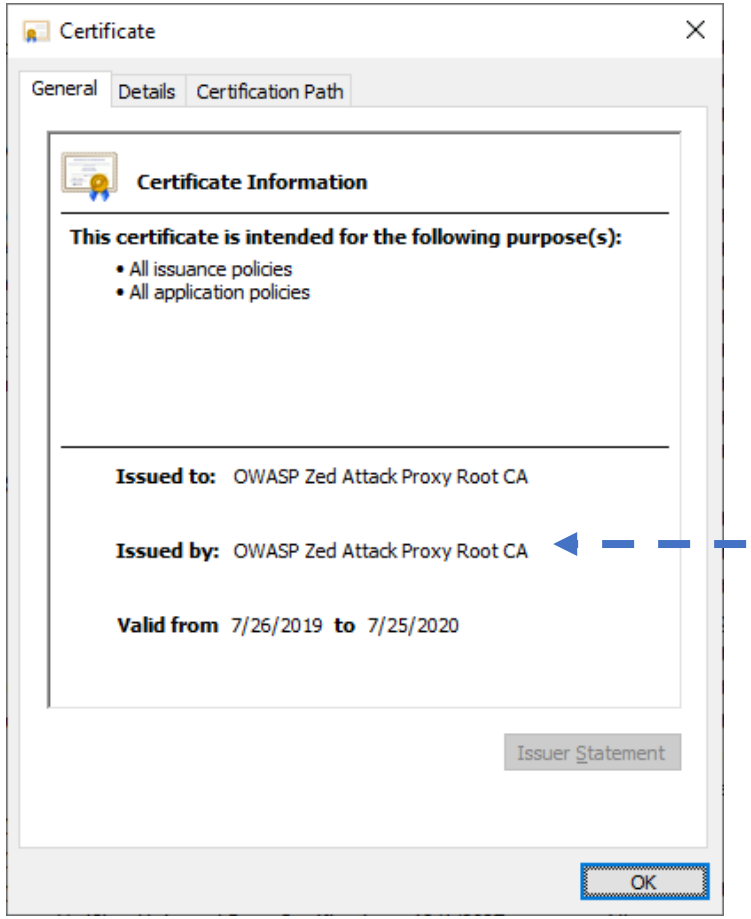
Several informations: alerts, scan results, …

ZAP – Setup

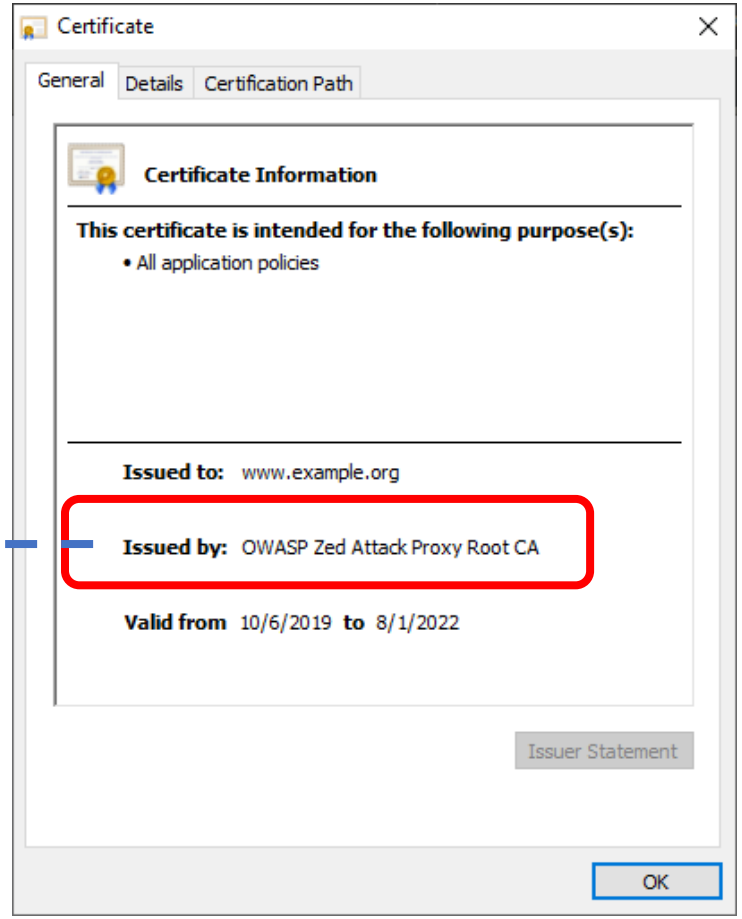
- Configure proxy in browser and root of trust
- On sites with HTTPS, ZAP generates a new certificate with the name of the original
  - The new certificate is signed with “OWASP Zed Attack Proxy Root CA”



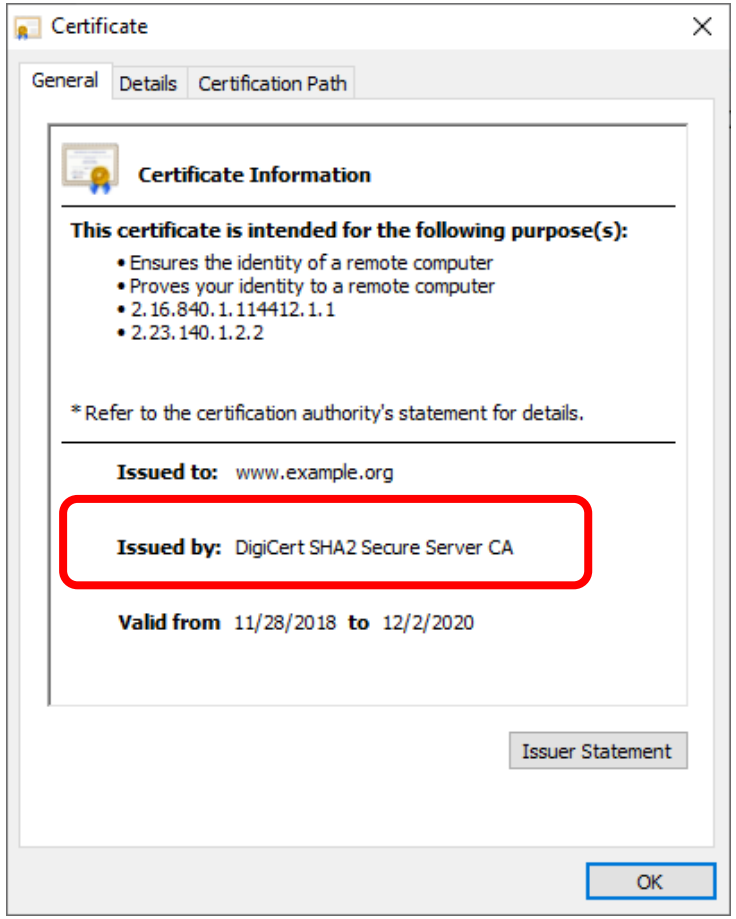
Original and generated certificate



Trust root



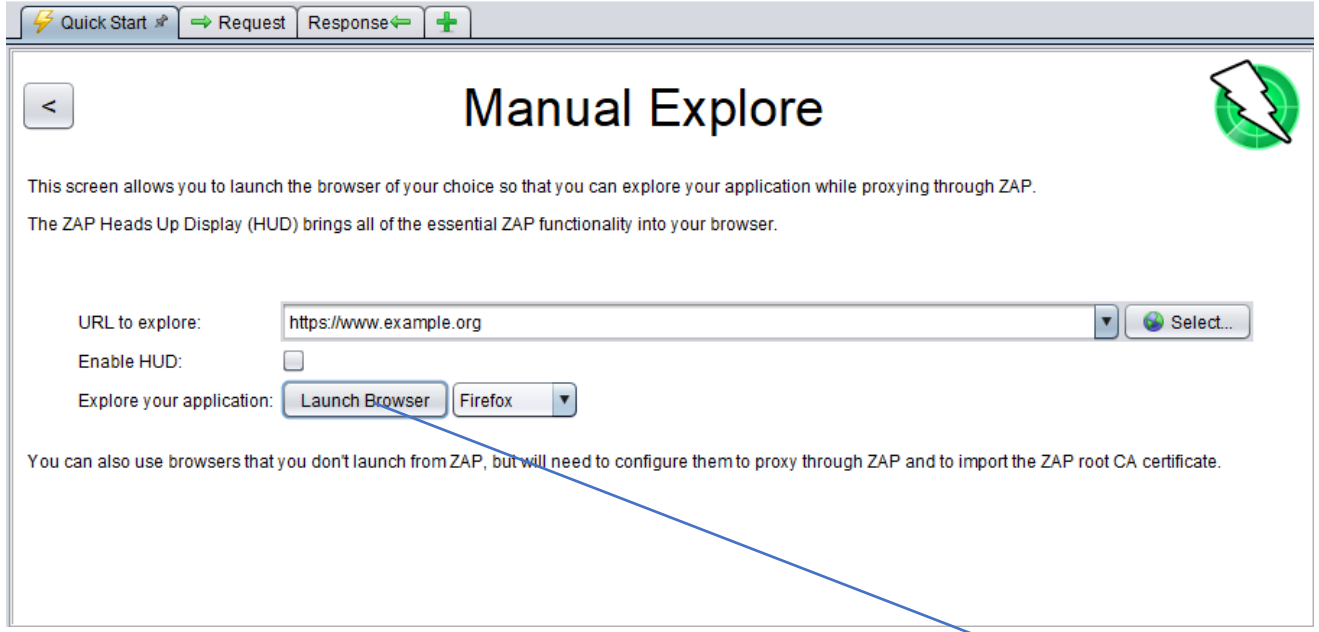
Generated by ZAP

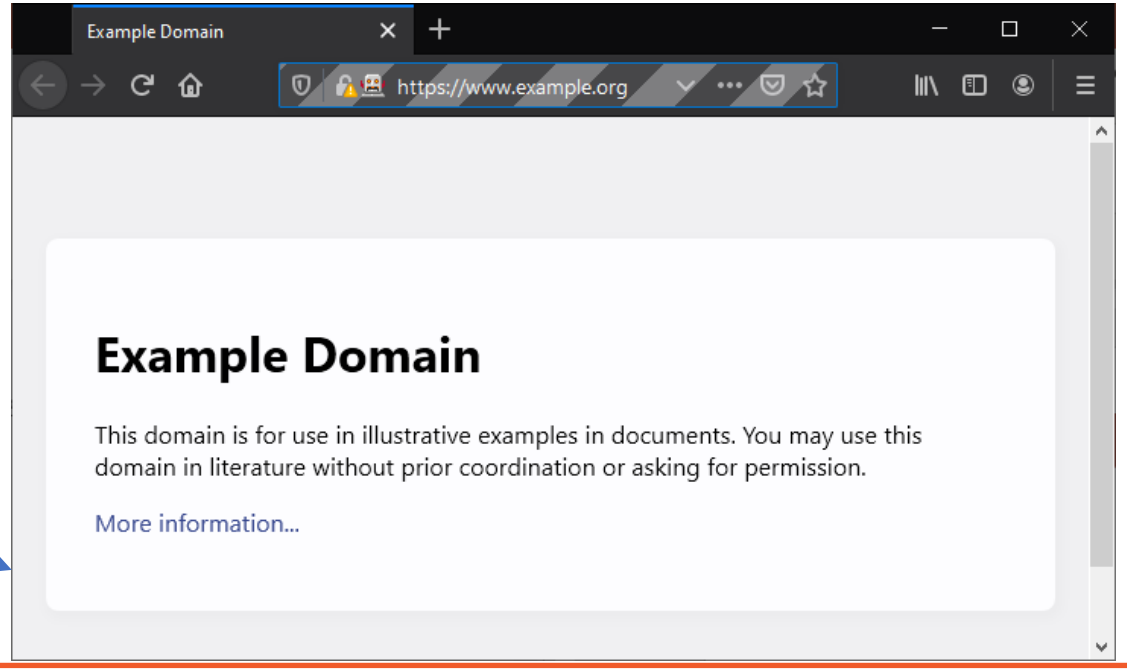


original

Setup automático

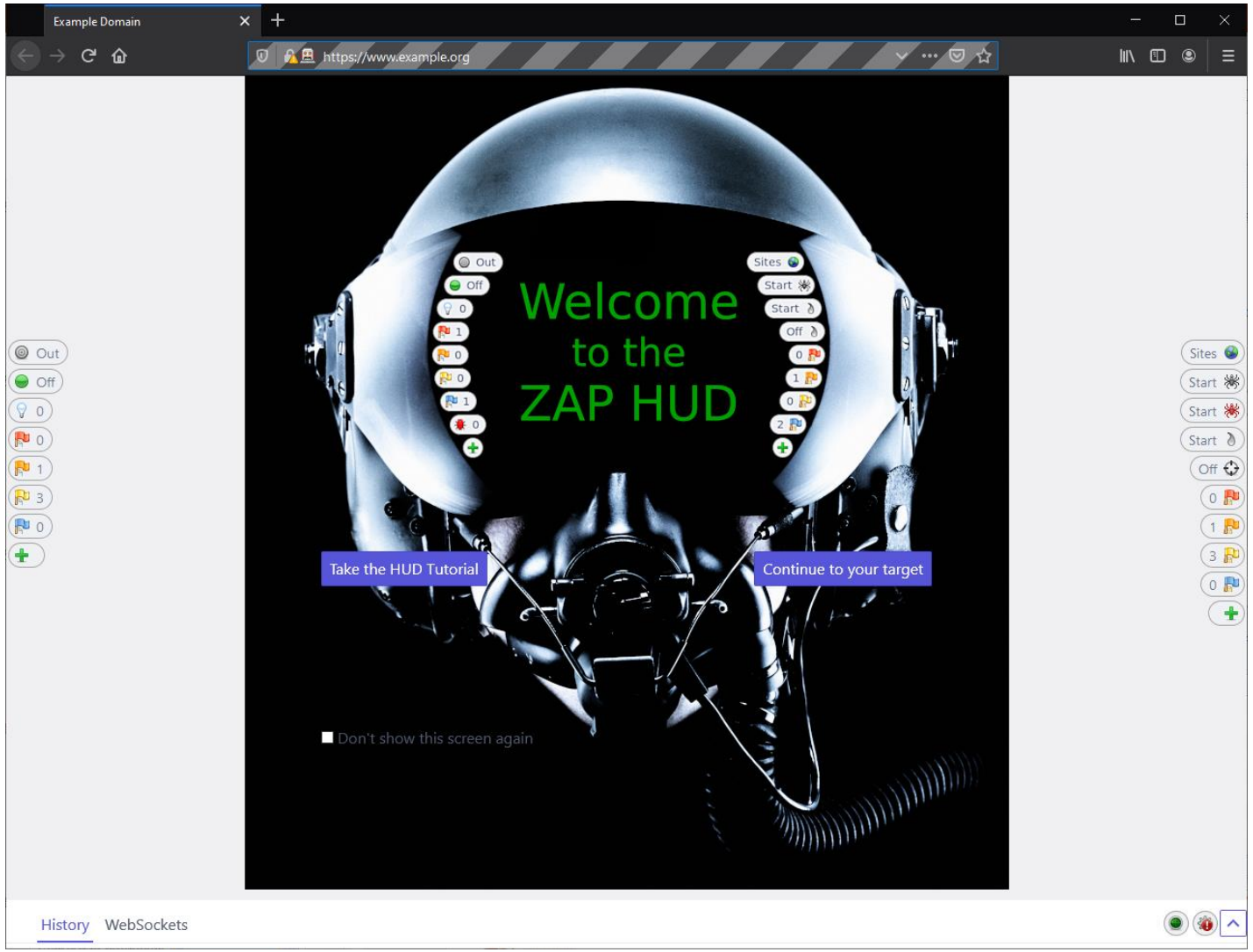
- “Quick Start” possibilita o arranque do *browser* com configurações predefinidas



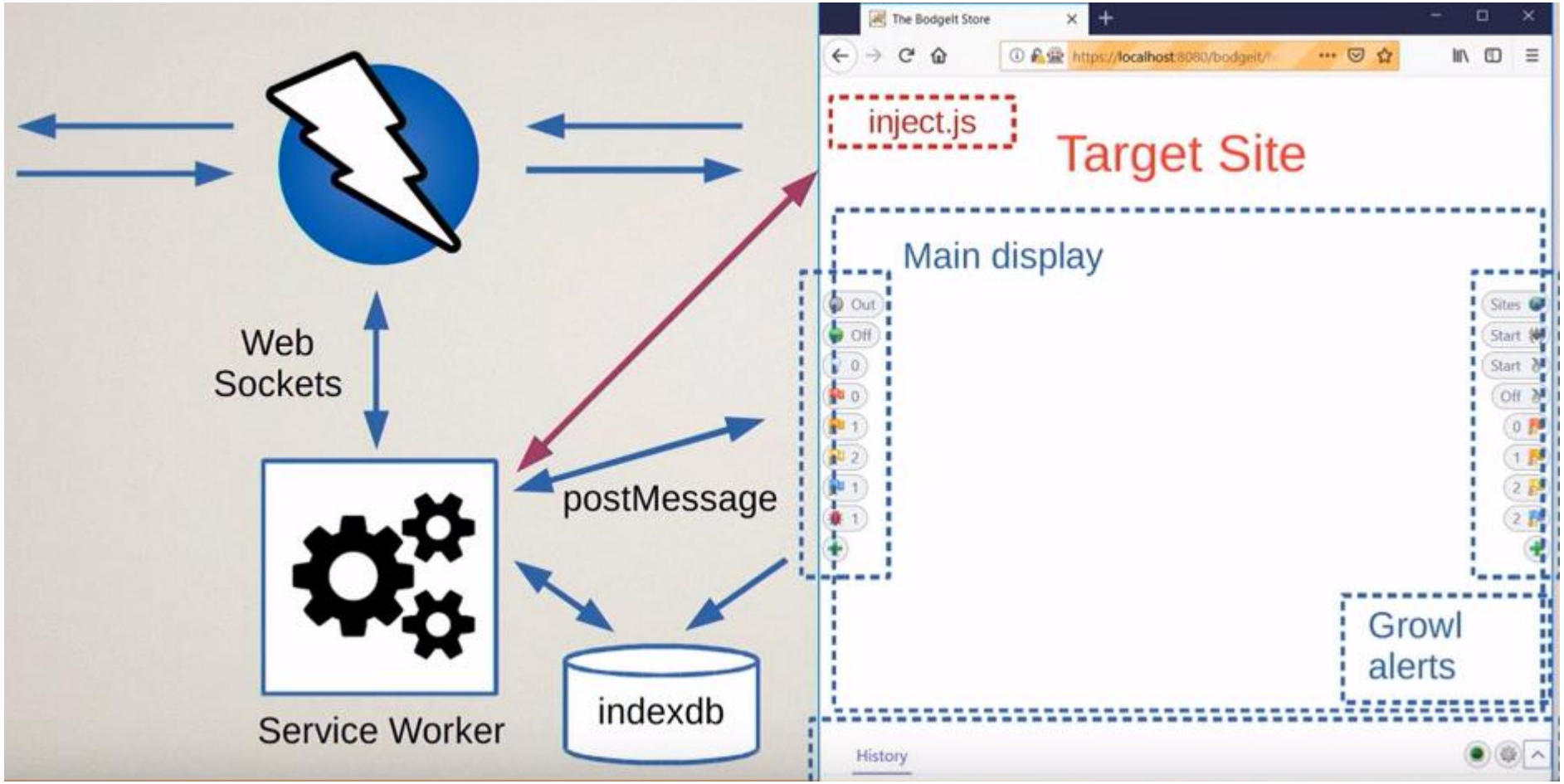


ZAP com *Heads Up Display* (HUD)

- Minimalist interface
- Base set of options, which can be configured
  - page alerts
  - website alerts
  - Crawler
  - Vulnerability Scan
  - Interaction history
  - …



HUD implementation



https://www.youtube.com/watch?v=1hbKGDgx_p0

Google Gruyere

- Web Application Exploits and Defenses
- https://google-gruyere.appspot.com/
- Application written in phyton, purposely with several security holes

Some exemples of attack

- Modify orders in transit
  - Accounts with more than 16 letters in the username
- XSS (stored)
  - https://google-gruyere.appspot.com/...../newsnippet.gtl
    - Experimentar inserir o seguinte texto nos *snippets*
    - read this!
- Get admin password (id administrator) with fuzzing
  - Custom fuzzing vector with Top500 passwords
  - https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/500-worst-passwords.txt
  - https://raw.githubusercontent.com/danielmiessler/SecLists/master/Usernames/top-usernames-shortlist.txt

Fuzzing targeting login with “administrator” account

Fuzzer

Fuzz Locations | Options | Message Processors

Header: Text | Body: Text

```
GET
https://google-gruyere.appspot.com/5612593494510624461136322471688897
22955/login?uid=administrator&pw= HTTP/1.1
Host: google-gruyere.appspot.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/
20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8
Accept-Language: pt-PT,pt;q=0.8,en;q=0.5,en-US;q=0.3
Connection: keep-alive
Referer:
https://google-gruyere.appspot.com/5612593494510624461136322471688897
22955/login?uid=administrator&pw=
Cookie: GRUYERE=
```

Fuzz Locations:

L...	Val...	# of ...	# of ...
He...		0	0

Add... | Remove | Payloads... | Processors...

☑ Remove Without Confirmation

Cancel | Reset | Start Fuzzer