Generation of vulnerabilities and threat reports for web applications

Antonio Robles Gelabert

MISTIC

Security of web applications

Tutor

Jordi Duch Gavaldà

7 de gener de 2019

Vull dedicar aquest treball a na Marta, la meva parella des de fa quasi 8 anys. Ella més que ningú sap l'esforç que li he dedicat a aquest projecte. Esper que tots els plans que hem deixat de fer per treure endavant aquesta feina valguin la pena. Gràcies per tot.

Resum

El projecte a desenvolupar consistia en una aplicació web que permetés analitzar les vulnerabilitats d'altres webs i mantenir un històric d'anàlisis per usuari.

Amb aquest objectiu s'ha desenvolupat una aplicació web que es caracteritza sobretot per la seva modularitat. Per una part, hi ha una API que permet realitzar distints tipus d'anàlisis i registrar-los. D'altra banda, hi ha una aplicació web que permet als usuaris interactuar còmodament amb l'API.

L'API s'ha creat de la forma més genèrica possible. Pel desenvolupament d'aquest projecte s'han definit 6 tipus d'anàlisis distints, però s'ha dissenyat de tal forma que és molt fàcil afegir nous tipus. A més, l'API està totalment separada de la part web. Això vol dir que en qualsevol moment es pot desenvolupar un altre client per l'API sense necessitat de modificar res d'aquesta.

El client web s'ha desenvolupat amb les tecnologies més potents del moment. L'objectiu principal en aquest apartat ha estat cercar la màxima simplicitat i comoditat possible pels usuaris. D'aquesta forma, la web ofereix a l'usuari un panell molt simple on pot veure els distints anàlisis realitzats per cada una de les webs, així com també permet realitzar nous anàlisis.

Summary

The project was about a web application that had to allow analyze the vulnerabilities of other websites and keep a history of these analysis for each user.

Keeping it in mind I have designed a web application that stands out for its modularity. On one side, there are an API that allows you to do different types of analysis and register the results. On the other hand, there are a web application that uses the API so that the users can do these analyses through a web interface.

The API has been created as generic as possible. For this project, 6 types of analysis have been defined, but the API design allows to add new types without problems. Moreover, the API is completely isolated from the web application. For this reason, we can develop other clients for the API without any additional change.

The web client has been developed with the best technologies at this moment. The main goal has been to achieve the maximum simplicity and the best experience for the users. For this reason, the application has a simple panel where we can find the analyses of each website and also realize new analyses.

Índex

1. Introducció

En aquest capítol es dona una visió general del treball realitzat. En la secció 1.1 es posa en context el desenvolupament d'aquest projecte. En la secció 1.2 es dona a conèixer la motivació que hi ha darrera d'aquest treball. En la secció 1.3 s'exposen els objectius que es volen assolir amb la realització d'aquest projecte. En la secció 1.4 s'analitzen les limitacions que afecten o poden afectar al projecte. En la secció 1.5 es dona a conèixer la planificació realitzada per aquest projecte. Per finalitzar, en la secció 1.6, s'explica com està estructurada la memòria del treball.

1.1 Contextualització

En l'actualitat, el tema de la seguretat informàtica ha obtingut un nivell de rellevància elevat. El continu desenvolupament tecnològic i l'evolució de la societat fa que cada vegada sigui més important aplicar altes mesures de seguretat en l'àmbit de la informàtica.

Cada vegada es veu més clar que el futur s'articula al voltant de termes com *Big Data*, *Internet of Things* o Intel·ligència Artificial. Aquests avanços tecnològics suposen una oportunitat increïble, però a l'hora suposen un risc que s'ha de controlar. Hem d'evitar per tots els mitjans que es produeixin efectes col·laterals que afectin a la seguretat dels nostres sistemes.

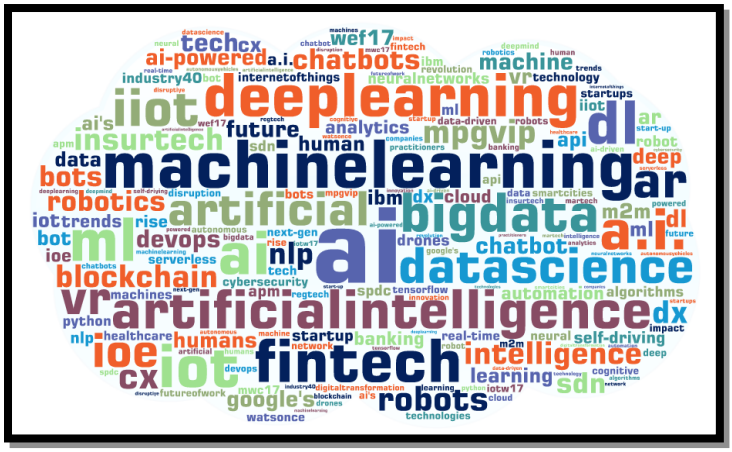


Figura 1.1: Temes d'actualitat en el món de la tecnologia

D'altra banda, cada cop és més comú que persones amb poc o cap coneixement sobre el món de la informàtica s'aventurin en empreses basades en aquest món. Aquest fet es deu principalment a la facilitat que existeix en el món actual per crear sistemes informàtics i començar a obtenir beneficis en poc temps. Ara bé, això suposa un gran perill, ja que provoca la proliferació de sistemes que no destaquen per la seva excel·lència tècnica.

En conseqüència, és imprescindible conscienciar a la societat sobre la importància de la seguretat i proporcionar eines tant a les organitzacions que es poden veure afectades com a altres organitzacions encarregades de la seguretat informàtica de tercers. A més, és important fer aquestes eines accessibles, de forma que no sigui imprescindible ser un expert per poder utilitzar-les i entendre-les.

1.2 Motivació

En aquest context sorgeix l'oportunitat de crear una eina que permeti analitzar la salut de les nostres aplicacions webs així com possibles complicacions que es puguin produir.

A nivell general és interessant desenvolupar una eina senzilla que permeti als usuaris finals mantenir d'alguna forma el control de les seves aplicacions. A més, proporciona un únic punt de control per distints focus d'anàlisis.

A nivell personal m'ha cridat l'atenció des del primer moment ja que engloba tots els ingredients que em resulten interessants. Per una part, hi ha un gran treball d'investigació per trobar les eines adequades quant als anàlisis per integrar en el sistema. Aquesta part d'anàlisi i comparació de distintes opcions m'ha resultat molt interessant. A més, m'ha permès conèixer moltes eines relacionades amb la seguretat que fins el moment desconeixia. D'altra banda, m'ha motivat molt la part de la integració en una solució web, ja que tinc experiència en aquest àmbit i crec que podia aportar molt en aquest sentit.

1.3 Objectius

Podem distingir dos tipus d'objectius. Per una part, hi ha els objectius propis del TFM. D'altra banda, hi ha els objectius que m'he marcat a nivell personal per obtenir un producte interessant i diferent baix el meu criteri.

L'objectiu principal del TFM era desenvolupar una aplicació web que permeti analitzar les vulnerabilitats i altres riscs de seguretat d'altres webs. A més, també es volia poder mantenir un històric dels anàlisis per cada un dels usuaris. Finalment, aquest sistema havia de ser fàcil d'utilitzar i d'entendre.

Quant als meus objectius personals, destacaria la modularitat. En cap moment la meva intenció ha estat desenvolupar l'eina i acabar-la amb tot el seu potencial. El que he cercat des del primer moment ha estat desenvolupar una eina modular de forma que es pugui millorar i ampliar constantment.

Figura 1.2: Exemple de disseny modular en dispositius mòbils

Per mi un requisit fonamental és que l'eina no es quedi en un client web, també s'ha de poder accedir a ella mitjançant altres vies com bots, aplicacions mòbils o altres integracions. D'altra banda, l'objectiu no era definir un conjunt d'escàners o analitzadors i desenvolupar l'eina al voltant d'això. L'objectiu era que l'eina sigui un sistema genèric que permeti afegir o modificar els analitzadors de forma senzilla.

1.4 Limitacions

Per encarar aquest projecte s'han hagut de tenir en compte certes limitacions.

En primer lloc, s'ha hagut de tenir en compte la limitació temporal. Aquest projecte tenia una duració limitada d'uns mesos. Per tant, totes les fases del projecte s'han hagut d'ajustar a un període màxim marcat. Aquesta limitació pot haver produït que no s'hagi arribat a la millor solució en tots els aspectes, però és imprescindible per garantir un mínim producte viable.

En segon lloc, s'ha hagut de tenir en compte la limitació econòmica. Aquesta limitació afecta principalment a les eines d'anàlisi i escaneig que es poden utilitzar, ja que hi ha eines molt potents però que requereixen un pagament o contractar un servei. Totes aquestes opcions han estat descartades donat l'àmbit del projecte.

Finalment, s'ha hagut de tenir en compte la limitació quant a l'àmbit del producte a realitzar. En un altre context podríem ser molt més ambiciosos i crear un primer producte molt complet i potent. Ara bé, en aquest cas, s'han afegit els mòduls necessaris per veure la capacitat de l'aplicació sense pretendre cobrir totes les possibilitats. És a dir, s'ha desenvolupat un mínim producte viable.

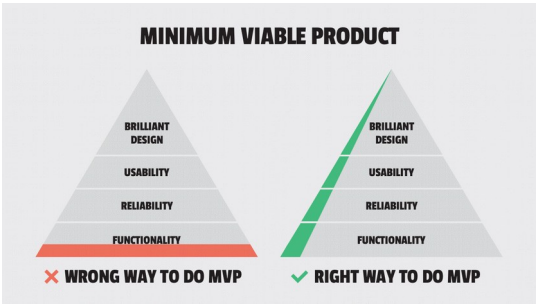


Figura 1.3: Il·lustració d'un mínim producte viable

1.5 Planificació

Aquesta ha estat la planificació establerta pel desenvolupament del projecte al llarg de les setmanes:

Setmanes	Tasca
1 - 4	Investigació de les distintes eines per analitzar vulnerabilitats
5	Definició de l'arquitectura del projecte
6	Disseny de l'API
7 - 9	Implementació de l'API i integració dels analitzadors inicials
10	Disseny de la interfície web
11 - 13	Implementació del client web
14 - 15	Documentació

1.6 Estructura del document

La memòria del projecte consta de sis capítols.

2. Es mostra com s'ha realitzat la investigació dels distints tipus d'analitzadors i escàners que hi ha al mercat. Es comparen les solucions trobades i s'exposen els resultats obtinguts.

3. Es veu detalladament com s'ha dissenyat l'arquitectura del sistema i per quins components està format.

4. S'analitza l'API desenvolupada, els mòduls que conté i les tecnologies que s'han utilitzat per la seva creació.

5. S'analitza el client web, els mòduls que el formen i les tecnologies utilitzades durant el seu desenvolupament.

6. Es presenten els resultats del projecte, una reflexió personal i possibles ampliacions de cara al futur.

2. Investigació

En aquest capítol s'analitza com va ser el procés d'investigació d'analitzadors de vulnerabilitats i altres eines similars. En la secció 2.1 es defineix quin és l'objectiu de la investigació, és a dir, que volem aconseguir. En la secció 2.2 es detalla com s'ha produït la investigació i quins són els aspectes destacables de cada eina que hem trobat. Per finalitzar, en la secció 2.3 compararem les eines trobades i veurem quines són les més adequades per aquest projecte.

2.1 Objectiu

Abans de començar a pensar en l'aplicació, de fer el disseny de la interfície i fins i tot de l'arquitectura, hem de veure de primera mà quin és el potencial existent en aquest context i fins a on podem arribar nosaltres. Per aquest motiu, és molt important fer un anàlisi previ de quines solucions existeixen en el mercat i quines eines podem usar nosaltres per treure el màxim profit de la nostra aplicació.

L'objectiu principal és comparar distintes eines que trobem en el mercat, analitzadors de vulnerabilitats, escàners, i altres eines que ens puguin ser útils. Entre totes les eines que vegem ens quedarem amb la millor de cada tipus, tinent en compte aspectes com la facilitat d'integració, la capacitat de l'eina o el seu rendiment.

Un dels aspectes claus en aquesta investigació serà el tema econòmic. En aquest món les poques solucions de qualitat no solen ser gratuïtes, però en el nostre àmbit només ens interessa les que si ho són, per tant les altres es descartaran automàticament. Tot i així, pot ser interessant veure que ens podrien arribar a proporcionar aquestes eines.

2.2 Anàlisi

Durant la investigació s'han arribat a analitzar fins a 12 opcions distintes. En algunes entrarem més en detall i altres les passarem més per damunt, però les repassarem totes. Moltes eines han estat descartades directament pel fet de no ser gratuïtes, però hi ha dues eines que si que veurem en aquest anàlisi. A continuació podem veure cada una de les eines analitzades:

nmap

Aquesta eina de codi obert serveix principalment per realitzar un escaneig de ports. A més, permet detectar sistemes connectats a una xarxa, determinar el sistema operatiu d'una màquina o fins i tot detectar els serveis que operen a cada port i les seves versions.

D’altra banda, és molt robusta. Té moltes formes d’aconseguir el seu objectiu, no hi ha una única forma. A més, durant l’escaneig, és capaç d’adaptar-se a les condicions de la xarxa perquè la seva tasca tengui èxit.

També és molt ressenyable la seva capacitat per passar desapercebuda. No realitza tasques a la xarxa que generin un grau de sospita elevat. És més, va ser creada per evadir els sistemes de detecció d’intrusos. Aquesta característica és molt important ja que del contrari els sistemes podrien amagar els seus punts febles.

Finalment, cal destacar que és molt fàcil d’utilitzar. Es tracta d’un programa que s’executa per línia de comandes amb un gran conjunt de paràmetres molt fàcils d’entendre. A més, incorpora un manual amb explicacions i exemples.

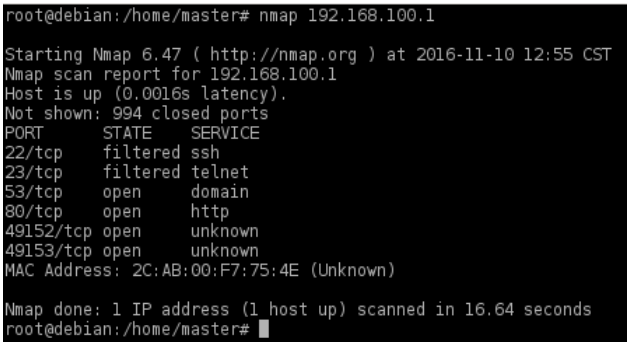


Figura 2.1: Exemple d’execució de l’eina nmap

Aquesta eina és molt famosa dins la comunitat. És gratuïta, molt potent i fàcil d’utilitzar. Per aquest motiu és una candidata clara a ser utilitzada dins l’aplicació.

nikto

Aquesta eina permet detectar configuracions per defecte, fitxers per defecte o insegurs, servidors i software desactualitzat, etc. La seva instal·lació és senzilla i accessible quasi per qualsevol família de sistemes operatius.

D’altra banda, el seu ús no pareix del tot senzill. Es tracta d’un programa executable per línia de comandes com en el cas anterior, però el seus paràmetres no són del tot clars, almenys per gent sense un coneixement profund.

Pareix que també té certa popularitat, però no pareix tan fàcil d’utilitzar ni tant potent com l’anterior.

Nessus

Aquesta eina realitza directament escaneigs de vulnerabilitats a diversos sistemes operatius. Es basa principalment en **nmap** per trobar els ports oberts i després intenta aplicar diferents explotis per comprovar si existeixen vulnerabilitats.

He tingut l'oportunitat de provar-lo i els resultats són molt bons. És una eina molt potent. A més, es pot configurar per línia de comandes i obtenir els resultats en varis formats de text distints. Per tant, és fàcil d'integrar en qualsevol sistema.

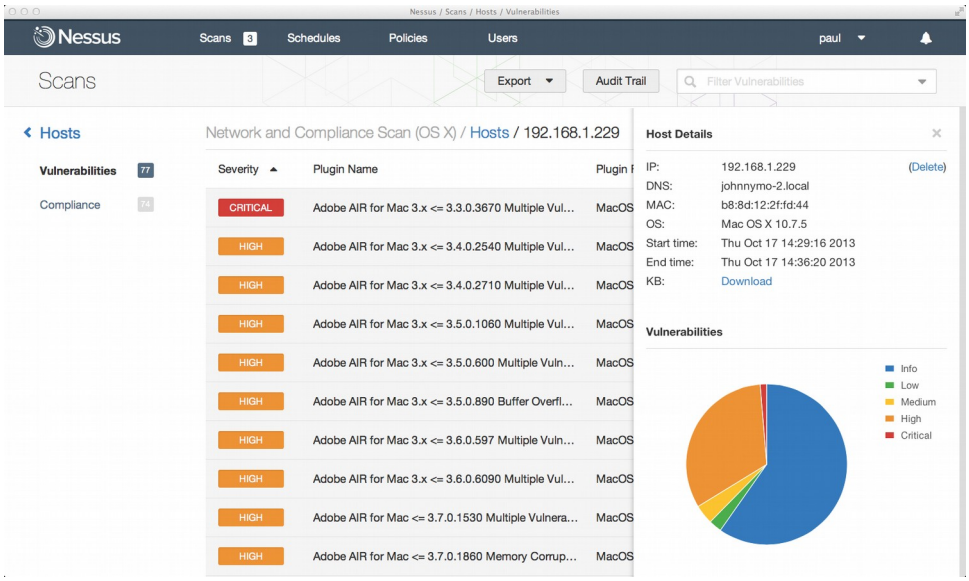


Figura 2.2: Exemple d'anàlisi de vulnerabilitats amb l'eina Nessus

D'altra banda, les seves operacions poden ser fàcilment detectables. Per tant, els sistemes analitzats podrien detectar la seva presència i alterar el seu comportament.

Tot i així, el seu principal problema és no ser gratuïta (almenys per les versions comercials), però he considerat interessant comentar aquesta eina ja que és d'una qualitat elevada.

OpenVAS

Bàsicament, aquesta eina és la versió gratuïta de Nessus, o això és el que diuen. Principalment ofereix les mateixes característiques amb més limitacions i un entorn molt més simplista.

Ara bé, també he tingut l'ocasió de provar-lo i no hi ha color. El rendiment d'un i l'altre no té res a veure. OpenVAS es fa més complicat d'entendre i utilitzar. A més, he experimentat problemes de rendiment en la meva màquina un cop ho posava en marxa.

Per tant, les seves característiques són molt desitjables, però els seus resultats (almenys en el meu cas) no són bons.

whois

Aquest protocol permet consultar a una base de dades qui és el propietari d'un domini. Tradicionalment s'utilitza mitjançant línia de comandes. És molt fàcil d'utilitzar i molt fàcil d'entendre.

Aparentment no pareix que aquesta eina ens pugui ajudar molt, però ens dona informació distinta, i això sempre és important. A més, en aquesta base de dades poden aparèixer telèfons, adreces, correus electrònics i més informació d'interès per un atacant. Per exemple, un atacant podria fer ús d'aquestes dades per dur a terme un atac de Phishing. Per tant, està bé tenir aquesta informació per poder prendre les mesures de seguretat adients.

arachni

arachni és una eina gratuïta que permet avaluar la seguretat de les aplicacions web. No és com les que hem vist fins ara, aquesta eina és distinta. Aquesta eina es basa principalment en allò que pot treure interactuant amb la web. Per exemple, realitza intents de XSS o injecció SQL.

D'altra banda, la seva documentació és molt extensa. Permet moltes accions i moltes configuracions, massa pel meu gust.

Aquesta eina pareix interessant, aporta coses distintes. Tot i així, sembla que per treure-li suc s'ha d'examinar exhaustivament.

nmap vulners

En aquest cas no es tracta d'una eina, es tracta d'un script que complementa una altra eina. Anteriorment ja hem comentat l'eina **nmap** i tot el seu potencial. Doncs bé, com hem comentat abans, **nmap** permet escanejar els ports oberts, els serveis que fan ús d'aquests ports i a vegades inclús la versió d'aquests serveis.

L'script **nmap vulners** aprofita tot el potencial de nmap per obtenir la versió dels serveis i els cerca a una base de dades de vulnerabilitats per trobar coincidències. D'aquesta forma, pot detectar vulnerabilitats en el sistema analitzat. És més, les vulnerabilitats detectades estan confirmades, ja que només es basa en versions concretes. Si **nmap** no és capaç de detectar la versió aquest script ja no farà la cerca a la base de dades.

Això té un potencial molt gran ja que simplement amb **nmap** i un paràmetre extra serem capaços de detectar vulnerabilitats. L'únic inconvenient és que depenem de que **nmap** sigui capaç de detectar la versió.

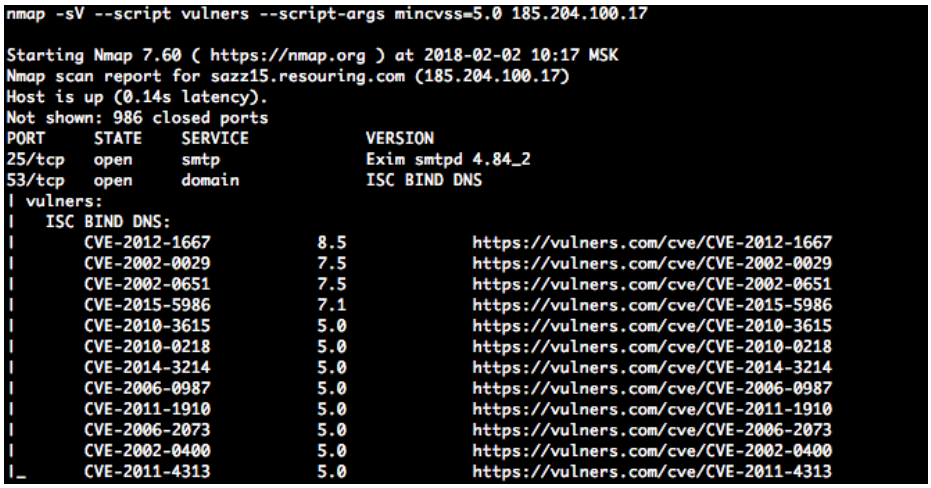


Figura 2.3: Exemple d'execució de nmap amb l'script vulners

nmap vulscan

Aquest cas és molt similar a l'anterior. Al igual que amb **nmap vulners**, **nmap vulscan** és un script que complementa l'eina **nmap** per detectar vulnerabilitats. Una de les diferències és que aquest script permet triar entre distintes bases de dades de vulnerabilitats.

Tot i així, la principal diferència és que els resultats d'aquest script no són del tot fiables, ja que no es basa amb la versió per detectar la vulnerabilitat. Això vol dir que donat un servei pot detectar distintes vulnerabilitats per distintes versions. Per tant, els resultats d'aquest anàlisi no vol dir que siguin certs.

Ara bé, aquest cas també és molt interessant, ja que ens protegeix d'alguna forma dels casos en que **nmap** no és capaç de detectar la versió. En aquests casos tendrem un conjunt de vulnerabilitats possibles que nosaltres podrem analitzar si són certes o no.

BuiltWith

Aquesta eina és molt potent i diferent. Permet detectar quines tecnologies estan essent utilitzades per la pàgina web que nosaltres indicam. Això és molt interessant, ja que darrera cada llibreria o framework pot haver una o varies vulnerabilitats que igualment fan a la nostra web vulnerable. Per tant, tenir un llistat de tecnologies amb les seves versions és una informació molt útil per evitar problemes en el futur.

He tingut l'oportunitat de provar aquesta eina i els resultats són molt bons. Aporta mola informació i de forma molt precisa. Desafortunadament, aquesta eina no és gratuïta.

Wappalyzer

Aquesta eina permet fer exactament el mateix que l'eina anterior. La principal diferència és que aquesta si que es pot usar gratuïtament. Tot i que usar la seva API requereix fer un pagament, es pot instal·lar el seu software en una màquina nostra i fer ús del seu potencial.

També he tengut l'oportunitat de provar-ho i els resultats també són molt bons. És cert que es queda un pas enrere de **BuiltWith**, però igualment són resultats molt interessants. A més, és fàcil d'integrar i d'entendre.

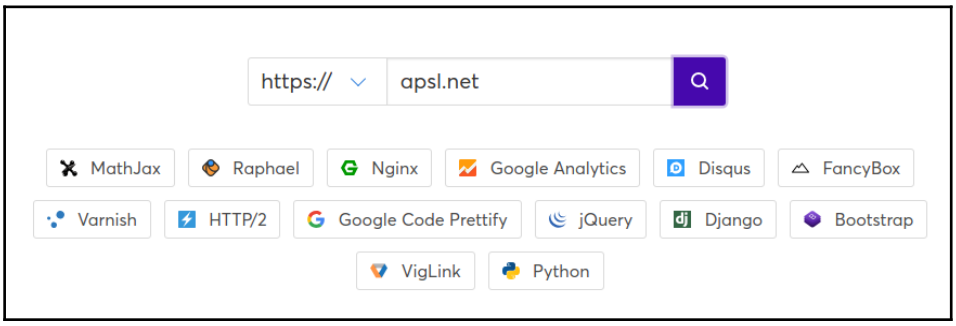


Figura 2.4: Anàlisi de les tecnologies de la web apsl.net amb Wappalyzer

WAScan

Aquesta eina és un analitzador de la seguretat de les aplicacions web. Permet realitzar distints tipus d'anàlisis: atacs XSS, injeccions SQL, cerca de fitxers vulnerables, atacs de força bruta, etc.

Sembla una eina potent i amb moltes possibilitats. Tot i així, la gran quantitat d'opcions fa que sigui un poc complicat arribar a treure-li tot el potencial. A més, pareix que necessita molta interacció manual en alguns del seus processos.

sublist3r

Aquesta eina és molt interessant. Permet detectar tots (o pràcticament tots) els subdominis d'una pàgina web. Per aconseguir-ho fa ús de distintes fonts com: Google, Yahoo, Bing, ReverseDNS, etc.

Aquesta eina ja l'havia utilitzat abans i és molt potent i molt fiable. Obtenir els subdominis d'una web pot parèixer poc rellevant però és molt interessant. Moltes vegades es detecten APIs, panells d'administració o eines d'ús intern que normalment estan menys protegits que la part més visible.

D'altra banda, l'eina és molt fàcil d'utilitzar i consta d'una versió per python. Per tant, és fàcilment integrable.

2.3 Resultat

Després d'analitzar fins a 12 eines distintes hem hagut de fer una selecció d'aquelles que hem cregut que s'adaptaven millor a les condicions del projecte. Entre els factors més importants s'ha mirat que cada eina aporti alguna cosa distinta. A més, s'han seleccionat aquelles que són fàcilment integrables i que no requereixen un esforç excessiu per treure tot el potencial. I evidentment, s'han descartat aquelles que no són gratuïtes. El resultat és el següent:

1. **nmap**: Aquesta eina ha estat seleccionada ja que ens permet fer un anàlisi del servidor. Podem veure quina és la IP, el sistema operatiu i fins i tot els ports que està utilitzant i amb quins serveis. És molt fàcil d'entendre i d'integrar en qualsevol sistema.
2. **nmap vulners**: Aquest script juntament amb nmap ens permet detectar vulnerabilitats en el sistema. És pràcticament la més important de totes, ja que la seva informació és molt valuosa.
3. **nmap vulscan**: Aquest script és molt útil per complementar l'anterior. Ens permet valorar possibles vulnerabilitats en aquells casos que nmap no pugui detectar la versió d'un servei.
4. **whois**: Aquesta eina també ha estat seleccionada ja que ens aporta un tipus d'informació distinta i no requereix cap esforç per usar-la.
5. **Wappalyzer**: Aquesta eina també ha estat seleccionada ja que ens permet conèixer les tecnologies que usa una web. Tot i que la seva integració no és trivial, cap altra eina ens aporta aquesta informació. A més, un cop integrada, és molt fàcil d'utilitzar.
6. **sublist3r**: Aquesta eina és la darrera eina seleccionada per la primera versió del projecte. Com hem comentat abans, ens aporta un tipus d'informació molt distinta a les altres eines. A més, és molt potent i fàcil d'utilitzar.

Per tant, aquestes són les 6 eines seleccionades per integrar de forma inicial en la nostra aplicació. Com es pot observar, són 6 eines molt distintes que ens aporten distinta informació però molt valuosa. D'altra banda, serà molt interessant veure com l'aplicació permet integrar dins ella 6 eines tan distintes sense que això generi mal de caps.

3. Arquitectura

En aquest capítol es detalla com s'ha dissenyat l'arquitectura del sistema. En la secció 3.1 s'explica que és una API i perquè ens pot ser molt útil en el nostre sistema. En la secció 3.2 es veu en que consisteix el Client Side Rendering i com pot encaixar dins la nostra aplicació. Finalment, en la secció 3.3 s'explica com s'ha dissenyat l'arquitectura del sistema i que ens aporta amb aquesta estructura.

3.1 Que és una API

Abans de veure com s'ha dissenyat l'arquitectura del sistema hem de tenir clars dos conceptes: API i Client Side Rendering.

Una API és una capa d'abstracció sobre un sistema que ens permet interactuar amb aquest mitjançant un conjunt d'accions que ens proporciona. El seu objectiu és proporcionar una forma senzilla de comunicació entre diferents components de software.

Les APIs han adquirit una gran popularitat en l'actualitat. Això es deu a que en el món de la informàtica cada vegada tot està més connectat i les APIs són una de les formes més senzilles de realitzar aquestes integracions. Molts dels components de software que utilitzam avui en dia utilitzen APIs. Per exemple, les aplicacions mòbils que ofereixen la mateixa funcionalitat que una web solen utilitzar APIs. O també, els motors de reserves que tant s'utilitzen en l'actualitat, solen utilitzar APIs per comunicar-se amb cada un dels proveïdors.

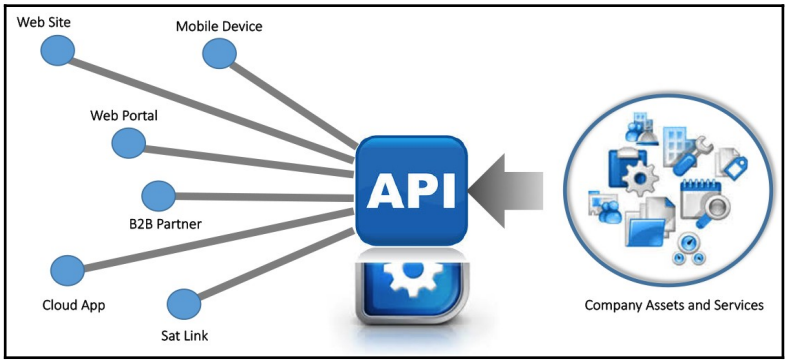


Figura 3.1: Esquema d'una arquitectura basada en una API

D'altra banda, hi ha distintes formes de dissenyar una API. Fins fa poc, l'arquitectura més popular era SOAP. De fet, segueix essent la més emprada per integracions de vols i hotels. Tot i així, nosaltres utilitzarem l'arquitectura REST. Aquesta arquitectura és molt més simple que SOAP, ja que no requerix protocols afegits, actua directament sobre HTTP. A més, aquesta arquitectura destaca per no mantenir cap tipus d'estat, el que la fa molt flexible. En l'actualitat han adquirit una gran popularitat.

El motiu principal pel qual són tan populars és perquè són molt fàcils de desenvolupar i d'usar. Bàsicament, es defineixen un conjunt de rutes (o endpoints) que representen recursos del sistema. Damunt aquestes rutes es defineixen les possibles accions, com per exemple: afegir, modificar o eliminar. La forma d'indicar una acció és molt senzilla, mitjançant mètodes HTTP com GET o POST. Per tant, si es realitza un POST sobre la ruta d'usuaris sabrem que es vol crear un usuari.

En resum, les APIs ens permeten definir una forma molt simple d'interactuar amb el nostre sistema. Aquesta característica serà clau en el disseny del nostre sistema.

3.2 Client Side Rendering

Un cop hem vist en que consisteix una API, ara toca veure que és el Client Side Rendering.

Fins fa pocs anys, sempre que s'accedia a una pàgina web es renderitzava de la part del servidor. Això vol dir que al accedir a la URL el servidor carregava el contingut i contestava al client, en aquest cas el navegador.

Doncs bé, el Client Side Rendering consisteix en carregar el contingut de la web de la part del client, com el seu propi nom indica. És a dir, al accedir a la URL, el servidor contesta amb l'estructura bàsica (el mínim contingut). Després, es carrega el contingut de la web des del client mitjançant JavaScript. Normalment es solen realitzar peticions AJAX contra el servidor per carregar els distints fragments de la pàgina.

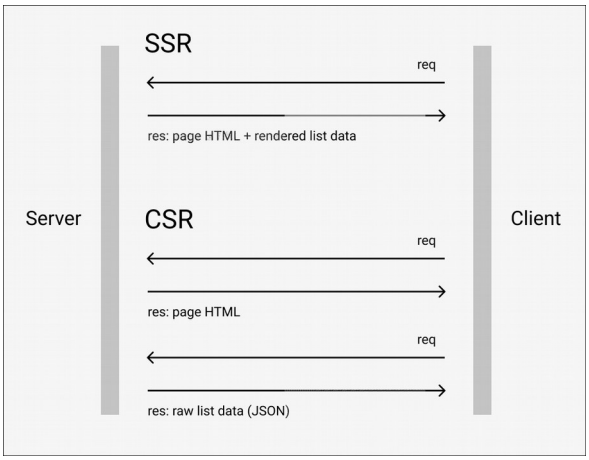


Figura 3.2: Comparativa de Server Side Rendering i Client Side Rendering

D'altra banda, estam acostumats a que per modificar el contingut d'una web es recarregui la pàgina, fent una nova petició al servidor i esperant la resposta. Amb el Client Side Rendering això no funciona així. Cada un dels continguts que es requereixen es van carregant de forma asíncrona sense recarregar la pàgina. De fet, a aquests tipus d'aplicacions se les sol anomenar Single Page Application, ja que no canvien mai de pàgina.

Aquests tipus d'aplicacions tenen una sèrie d'avantatges desitjables per la nostra aplicació. En primer lloc, es carreguen molt ràpid, ja que el contingut es va actualitzant a mesura que és necessari. En segon lloc, és molt còmode modificar el contingut de la pàgina, ja que tan sols s'ha d'actualitzar el fragment modificat i no la pàgina sencera. I finalment, s'entén molt bé amb les accions asíncrones, ja que formen part de la pròpia naturalesa de l'aplicació.

A continuació veurem perquè ens serà de gran ajuda aquest tipus d'aplicació web.

3.3 Disseny

Ara que ja hem vist que és una API i en que consisteix el Client Side Rendering podem veure com s'ha dissenyat l'arquitectura del sistema. Com hem comentat anteriorment, el gran objectiu d'aquest projecte és que sigui modular. Per aquest motiu, s'han creat dues aplicacions.

La primera aplicació és el core del sistema. Aquesta aplicació conté els següents components:

- Base de dades amb tota la informació dels usuaris i dels anàlisis realitzats.
- Panell d'administració on es poden dur a terme tasques d'administració així com revisar l'estat dels anàlisis realitzats.
- Els analitzadors de vulnerabilitats i altres eines similars.
- L'API que permet interactuar amb el sistema per realitzar i consultar anàlisis.
- Sistema de coes per poder dur a terme anàlisis de forma asíncrona, ja que hi ha alguns anàlisis que poden tardar varis minuts.

A continuació podem veure un esquema de com funciona l'aplicació core:

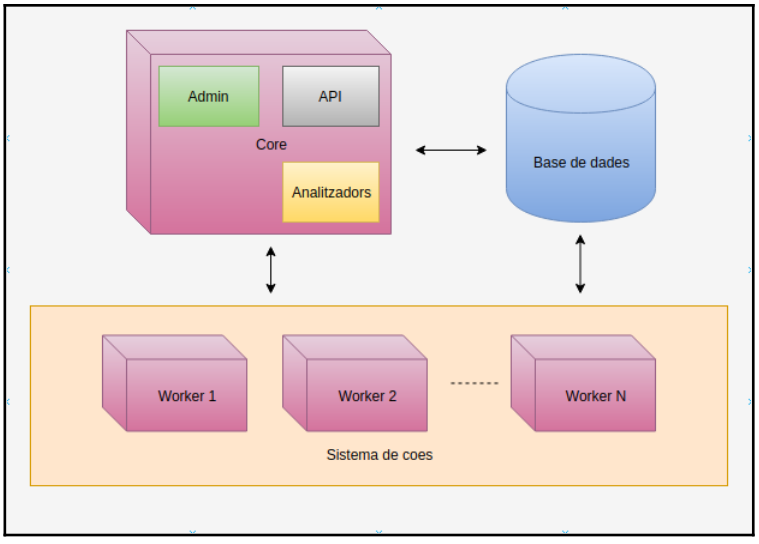


Figura 3.3: Esquema del funcionament de l'aplicació core

La segona aplicació és el client web. Aquesta aplicació bàsicament ofereix als usuaris una interfície web amb la que interactuar per realitzar els anàlisis. Per poder oferir la informació a l'usuari i dur a terme les accions que aquest reclama es connecta amb la API de l'aplicació anterior. De fet, la interfície està formada per un conjunt de components on cada component està vinculat a un recurs de l'API. D'aquesta forma, cada acció de l'usuari esdevé en una petició asíncrona a l'API que acaba provocant l'actualització del component.

A continuació podem veure un esquema de com funciona el client web:

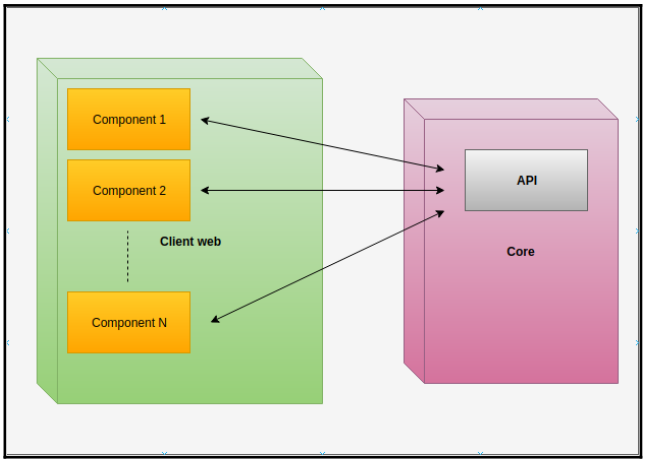


Figura 3.4: Esquema del funcionament del client web

D'altra banda, és interessant que de cara a l'exterior aquestes dues aplicacions es percebin com una única web. Per aconseguir-ho, s'utilitza un servidor web que fa de reverse proxy. És a dir, aquest servidor web decideix en funció de la ruta si la petició la deriva a l'aplicació core o l'aplicació web. El funcionament és molt simple, el servidor web escolta al port 80, mentre que les altres aplicacions escolten als ports 8001 i 8002 del localhost de la màquina (així no són accessibles des de l'exterior). Quan arriba una petició per /api o /admin el servidor web deriva les peticions a l'aplicació core. En cas contrari, deriva les peticions a l'aplicació web.

A continuació podem veure un esquema de com funciona aquesta distribució:

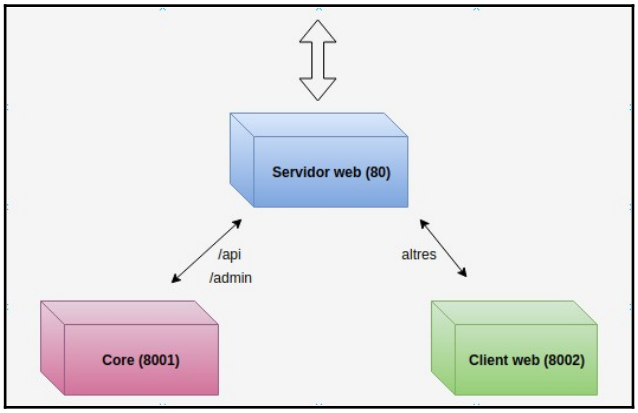


Figura 3.5: Esquema de la distribució de peticions entre el core i el client web

Amb aquesta arquitectura aconseguim l'objectiu de la modularitat, ja que podríem crear tants clients nous com volguéssim simplement connectant-los a l'API de l'aplicació core. D'altra banda, podríem fer canvis en el core sense afectar al client web sempre i quan l'API es comportés igual. És a dir, el comportament intern de l'aplicació core pot canviar, tan sols hem de mantenir el format de la API de forma que els clients es puguin comunicar amb ella com ho feien fins el moment.

4. Core

En aquest capítol s'analitza en detall l'aplicació core del sistema. En la secció 4.1 repassam les tecnologies utilitzades per l'aplicació. En la secció 4.2 veim quina estructura té el model i quines coses ens permet fer el panell d'administració. En la secció 4.3 veim com s'integren els analitzadors dins l'aplicació i com es poden afegir nous analitzadors. Finalment, en la secció 4.4 s'explica quina estructura té l'API i per quines rutes està formada.

4.1 Tecnologies

A continuació farem un repàs de les tecnologies més importants que s'han usat pel desenvolupament de l'aplicació core.

Docker

Docker és un sistema de contenidors que permet desenvolupar software en un entorn aïllat i després desplegar-lo fàcilment sense possibilitats de conflictes. Permet definir imatges que si funcionen quan se despleguen en una màquina han de funcionar en totes.

Django

L'estructura principal de l'aplicació és Django. Django és un framework web basat amb el llenguatge de programació python que permet desenvolupar webs de forma molt ràpida i elegant. Moltes de les tasques repetitives en quasi totes les webs estan resoltes dins Django. Aquest software et permet centrar-te en els detalls de la teva web sense haver de resoldre problemes de baix nivell.

Django Rest Framework

Django Rest Framework és un plug-in per Django que permet desenvolupar APIs REST. Aquest plug-in és molt conegut dins la comunitat Django gràcies al seu potencial. Permet crear una API molt completa amb molt poques línies de codi. A més, funciona molt bé amb el model de Django, i facilita molt la conversió del model al format de l'API.

PostgreSQL

Pel sistema gestor de base de dades (SGBD) s'ha escollit PostgreSQL. PostgreSQL és un dels SGBD més coneguts del mercat i dels més potents. En aquest cas l'hem escollit perquè cercàvem un sistema relacional i perquè és el que millor s'entén amb Django.

Celery

Celery és una coa de tasques asíncrones. Principalment permet executar funcions del sistema de forma asíncrona, és a dir, sense bloquejar el fil d'execució de la petició entrant. Aquesta eina és molt important ja que alguns anàlisis poden requerir més temps del desitjable en una petició HTTP.

4.2 Model i administració

En el framework Django, per definir les taules de la base de dades s'utilitzen les classes del model. Aquestes classes tenen una doble funcionalitat, defineixen les taules de la base de dades i a l'hora són la representació en objectes del llenguatge d'aquestes taules.

En aquest cas el model no té molt de misteri. La part més interessant és la dels distints anàlisis, com es defineixen i com s'ha dissenyat perquè sigui molt fàcil afegir-ne nous.

A continuació podem veure els models principals de l'aplicació:

- **Website**: Conté la informació de les webs analitzades (fins ara la URL).
- **AnalysisRequest**: Conté informació de les peticions d'anàlisi. És a dir, quan un usuari indica que vol realitzar un anàlisi, es crea una entrada en aquesta taula. Entrarem en més detall en la següent secció.
- **WebsiteAnalysis**: Conté informació dels distints anàlisis que un usuari ha realitzat sobre una pàgina web. Principalment serveix per indicar que un usuari ha començat a observar una pàgina web.
- **BaseAnalysis**: Aquest és el cas més interessant de tots. Es tracta d'una classe abstracta, i per tant, no representa una taula a base de dades. Ara bé, tots els tipus d'anàlisis que s'afegeixen al sistema han de ser fills d'aquesta classe. D'aquesta forma garantim una sèrie de característiques que han de complir tots els anàlisis perquè el sistema funcioni. Per tant, el model també s'ha dissenyat de forma que sigui el més modular possible.

Si ens fixem bé, no apareixen els usuaris. Això és perquè Django ja implementa per defecte una taula pels usuaris de l'aplicació.

Ara que ja hem vist els models principals, podem repassar ràpidament els models creats per cada un dels anàlisis.

Servidor

- **ServerAnalysis**: Cada entrada d'aquesta taula representa un nou anàlisi d'aquest tipus.
- **Port**: Conté la informació d'un port detectat en un anàlisi. Cada port està vinculat a un **ServerAnalysis**.

Vulnerabilitats

- **VulnerabilitiesAnalysis**: Cada entrada d'aquesta taula representa un nou anàlisi de vulnerabilitats confirmades o vulnerabilitats possibles. Dins la taula hi ha un camp que permet distingir les dues variants.
- **Vulnerability**: Conté la informació d'una vulnerabilitat detectada en un anàlisi.
- **VulnerabilityReference**: Cada vulnerabilitat té varies referències. Aquesta taula conté aquestes referències.
- **CVSS**: En la majoria de les vulnerabilitats tenim la seva avaluació CVSS. Aquesta taula conté aquestes avaluacions.

Registrant

- **RegistrantAnalysis**: Cada entrada d'aquesta taula representa un nou anàlisi d'aquest tipus.

Subdominis

- **SubdomainsAnalysis**: Cada entrada d'aquesta taula representa un nou anàlisi d'aquest tipus.
- **Subdomain**: Conté la informació de cada subdomini trobat. Cada subdomini està vinculat a un **SubdomainsAnalysis**.

Tecnologies

- **TechnologiesAnalysis**: Cada entrada d'aquesta taula representa un nou anàlisi d'aquest tipus.
- **WebsiteTechnology**: Conté informació sobre la versió concreta d'una tecnologia trobada en una pàgina web.
- **Technology**: Conté informació general sobre una tecnologia.
- **Category**: Conté informació sobre les categories de les tecnologies.

D’altra banda, l’aplicació té un panell d’administració en el que es poden consultar les dades del sistema. A més, es poden modificar les dades (molt útil per corregir errors), o fins i tot es poden eliminar.

Figura 4.1: Panell d’administració de l’aplicació core

Aquest panell és molt fàcil de crear. De fet, amb Django és pràcticament automàtic. Per cada model es crea un panell que conté un llistat de registres amb les accions que hem comentat abans. A continuació podem veure el panell que s’ha creat per les webs:

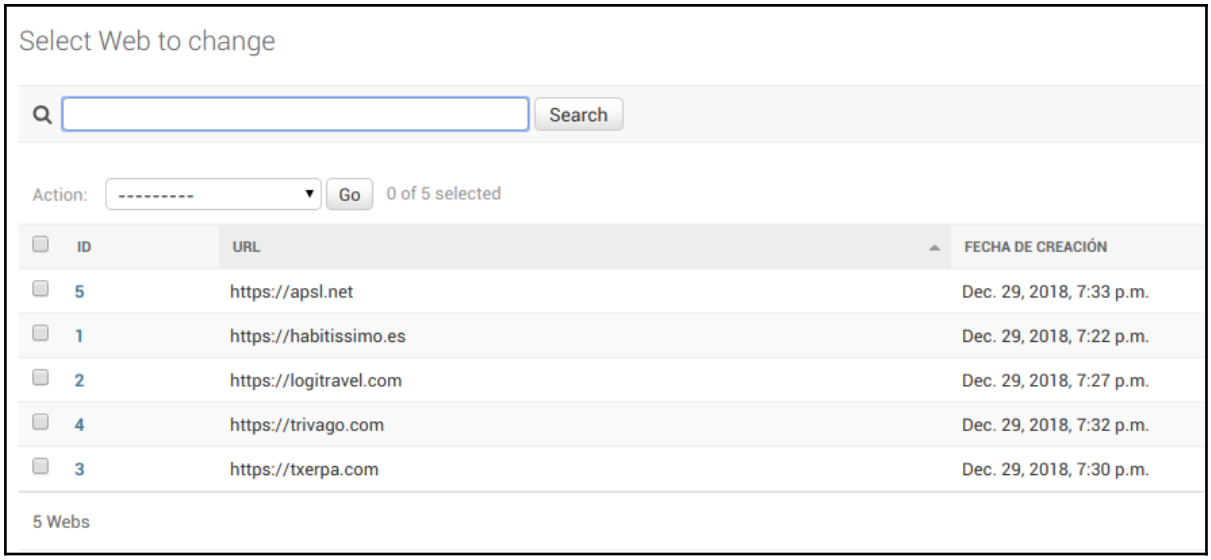
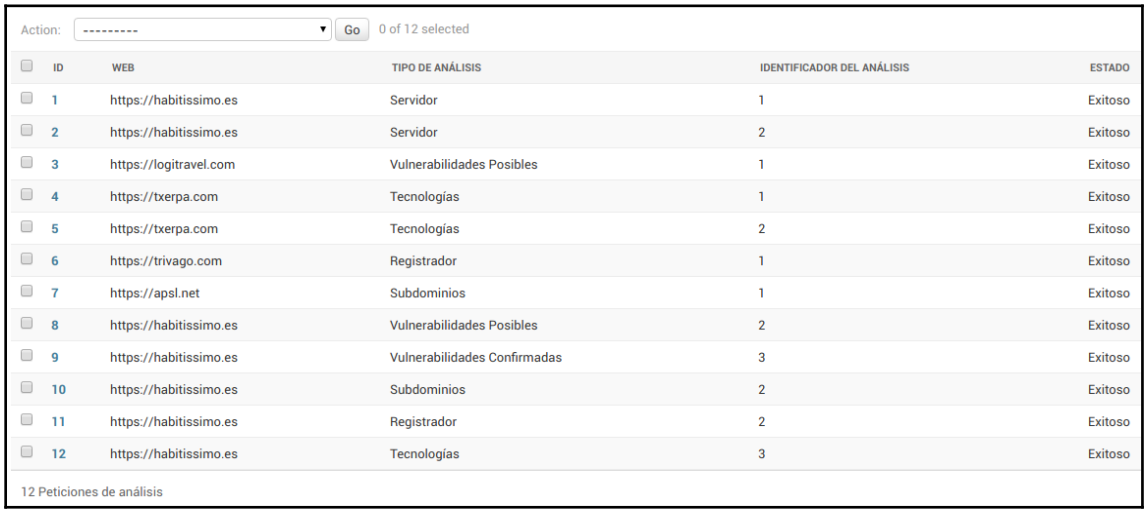


Figura 4.2: Panell d’administració de les webs analitzades

A més, aquests panells són molt útils durant el desenvolupament, ja que es poden veure fàcilment els registres sense necessitat de revisar la base de dades.

Tot i així, una de les parts més interessants d'aquest panell d'administració és la secció de les peticions d'anàlisi. En la pròxima secció entrarem en detall en aquest tema, però bàsicament hem de saber que per realitzar un anàlisi es fa una petició. Després, aquesta petició es processa asíncronament i es queda en estat 'pendent'. Si l'anàlisi es realitza satisfactòriament es modifica l'estat i es marca com 'exitós'. Però si l'anàlisi falla, es marca com 'fallit'.


Action: --------- Go 0 of 12 selected

ID	WEB	TIPO DE ANÁLISIS	IDENTIFICADOR DEL ANÁLISIS	ESTADO
1	https://habitissimo.es	Servidor	1	Exitoso
2	https://habitissimo.es	Servidor	2	Exitoso
3	https://logitravel.com	Vulnerabilidades Posibles	1	Exitoso
4	https://txerpa.com	Tecnologías	1	Exitoso
5	https://txerpa.com	Tecnologías	2	Exitoso
6	https://trivago.com	Registrador	1	Exitoso
7	https://apsl.net	Subdominios	1	Exitoso
8	https://habitissimo.es	Vulnerabilidades Posibles	2	Exitoso
9	https://habitissimo.es	Vulnerabilidades Confirmadas	3	Exitoso
10	https://habitissimo.es	Subdominios	2	Exitoso
11	https://habitissimo.es	Registrador	2	Exitoso
12	https://habitissimo.es	Tecnologías	3	Exitoso

12 Peticiones de análisis

Figura 4.3: Panell d'administració de les peticions d'anàlisi

Aquesta naturalesa asíncrona fa que sigui més difícil de seguir el seu estat i encara més trobar possibles errors. Per tant, tenir un panell on en tot moment podem saber l'estat d'aquests anàlisis és molt profitós.

4.3 Analitzadors

En el capítol 2 hem fet un repàs de distintes eines d'anàlisi i n'hem seleccionat 6. A continuació veurem com s'han integrat aquestes eines dins l'aplicació i com s'ha dissenyat perquè sigui molt fàcil afegir-ne noves.

En primer lloc s'ha creat una classe genèrica anomenada **BaseAnalyzer**. Aquesta classe conté totes les característiques que ha de complir un analitzador per funcionar correctament dins el sistema. Es tracta d'una classe abstracta, és a dir, no es pot instanciar. El seu objectiu és que tots els analitzadors parteixin d'aquesta base.

A continuació podem veure les seves característiques:

- **Constructor**: La classe disposa d'un constructor pel moment en que es crea una nova instància. Aquest constructor necessita que li passem una **WebsiteAnalysis**. D'aquesta forma, l'analitzador ja sap quin és l'usuari que vol realitzar l'anàlisi i sobre quina web el vol realitzar.
- **get_target** (l'ha d'implementar cada analitzador): Aquest mètode serveix per indicar l'objectiu de l'anàlisi. Amb el **WebsiteAnalysis** ja sabem la URL completa, però cada analitzador pot requerir l'objectiu en un format distint.
- **analyze** (l'ha d'implementar cada analitzador): Aquest mètode rep per paràmetre el resultat del mètode anterior (l'objectiu) i ha de tornar en format de diccionari de python el resultat de l'anàlisi. En aquest mètode és on cada analitzador haurà d'executar-se.
- **get_analysis_model** (l'ha d'implementar cada analitzador): Aquest mètode serveix per indicar quin tipus d'anàlisi es registrarà a base de dades.
- **execute**: Aquest mètode és l'únic que implementa aquesta classe. És el mètode que es crida des de l'exterior quan s'ha de realitzar una anàlisi. En primer lloc, crida al mètode **get_target** per obtenir l'objectiu de l'anàlisi. Seguidament, executa el mètode **analyze** passant-li com a paràmetre l'objectiu. Després, obté la classe del model que haurà de registrar amb el mètode **get_analysis_model**. A continuació, executa el mètode **build** de la classe del model passant com a paràmetres el **WebsiteAnalysis** i el resultat de l'anàlisi. Finalment, retorna el registre creat a base de dades.

En el capítol anterior hem comentat l'existència d'una classe genèrica per tots els models dels anàlisis per facilitar la seva integració. Doncs bé, aquesta classe conté el mètode abstracte **build** que acabam de mencionar. Bàsicament, obliga a tots els models dels anàlisis a implementar aquest mètode en el que s'ha de crear un nou registre mitjançant un **WebsiteAnalysis** i el resultat de l'anàlisi. Òbviament, cada model implementa aquest mètode de la forma requerida per registrar el seu tipus d'anàlisi.

Per tant, acabam de veure que el sistema ja està molt pensat per afegir nous analitzadors de forma senzilla. Tan sols hem de crear una nova classe que sigui filla del **BaseAnalyzer** i implementar els mètodes que ens demana.

A continuació veurem breument com s'ha realitzat aquest procés per cada un dels analitzadors que conté actualment l'aplicació.

Servidor

En primer lloc, s'ha hagut d'instal·lar **nmap** en el sistema. Però recordem que utilitzam **docker**, per tant, el que s'ha fet és instal·lar la dependència dins la imatge.

En segon lloc, s'ha creat la classe **ServerAnalyzer** i s'han implementat els mètodes abstractes de la classe **BaseAnalyzer**:

- **get_target**: Es retorna la URL sense el protocol, ja que és com s'ha de passar a l'eina **nmap**.
- **get_analysis_model**: Es retorna la classe **ServerAnalysis**.
- **analyze**: Executa **nmap** mitjançant una llibreria per python i processa el resultat de forma que genera un diccionari de python amb les dades.

Finalment, s'ha implementat el mètode **build** de la classe **ServerAnalysis** de forma que transforma el resultat d'un anàlisi en els registres corresponents per aquest tipus d'anàlisi (ServerAnalysis i Port).

Vulnerabilitats

En primer lloc, s'han hagut d'afegir els scripts de nmap (**vulners** i **vulscan**) dins l'aplicació. D'altra banda, **nmap** ja es trobava instal·lat.

En segon lloc, s'ha creat la classe **VulnerabilitiesAnalyzer** i s'han implementat els mètodes abstractes de la classe **BaseAnalyzer**:

- **get_target**: Es retorna la URL sense el protocol, ja que com hem comentat abans és la forma que requereix **nmap**.
- **get_analysis_model**: Es retorna la classe **VulnerabilitiesAnalysis**.
- **analyze**: Executa **nmap** de forma similar a l'analitzador anterior però afegint com a paràmetre un dels dos scripts. Es decideix quin dels dos scripts s'ha de passar a **nmap** en funció del paràmetre **confirmed**. A continuació, es processa el resultat de **nmap** i es completa amb informació addicional per cada vulnerabilitat gràcies a APIs externes.

Finalment, s'ha implementat el mètode **build** de la classe **VulnerabilitiesAnalysis** de forma que transforma el resultat en els registres corresponents per aquest tipus d'anàlisi (VulnerabilitiesAnalysis, Vulnerability, VulnerabilityReference i CVSS).

Registrant

En primer lloc, s'ha hagut d'afegir la dependència **whois** dins la imatge de docker. A més, s'ha afegit una llibreria de python per poder interactuar des de l'aplicació.

En segon lloc, s'ha creat la classe **RegistrantAnalyzer** i s'han implementat els mètodes abstractes de la classe **BaseAnalyzer**:

- **get_target**: Es retorna la URL sense el protocol, ja que és la forma en que s'ha de passar al programa **whois**.

- **get_analysis_model**: Es retorna la classe **RegistrantAnalysis**.
- **analyze**: Executa **whois** mitjançant la llibreria instal·lada per python passant-li com argument la URL de l'objectiu. Després, processa el resultat i ho transforma en un diccionari de python.

Finalment, s'ha implementat el mètode **build** de la classe **RegistrantAnalysis** de forma que transforma el diccionari de python amb les dades del resultat en els registres corresponents (RegistrantAnalysis).

Subdominis

En primer lloc, s'ha hagut d'afegir la llibreria per python de **sublist3r**.

En segon lloc, s'ha creat la classe **SubdomainsAnalyzer** i s'han implementat els mètodes abstractes de la classe **BaseAnalyzer**:

- **get_target**: Es retorna la URL sense el protocol, ja que és la forma en que s'ha de passar al **sublist3r**.
- **get_analysis_model**: Es retorna la classe **SubdomainsAnalysis**.
- **analyze**: Executa **sublist3r** mitjançant la llibreria per python passant-li com a paràmetre la URL de l'objectiu. En aquest cas no s'han de processar les dades ja que l'eina ja retorna un llistat de subdominis tal i com nosaltres ho volem.

Finalment, s'ha implementat el mètode **build** de la classe **SubdomainsAnalysis** de forma que transforma les dades de l'analitzador en els registres corresponents per aquest tipus d'anàlisi (SubdomainsAnalysis, Subdomain).

Tecnologies

En primer lloc, s'ha hagut d'integrar el software de **Wappalyzer** dins el sistema. Ha estat el cas més complicat de tots, ja que es tracta d'una dependència per node i nosaltres treballam en un entorn python. Per tal de solucionar-ho, s'ha creat un microservei apart basat en express (un framework web per node). D'aquesta forma, tenim un altre servei aixecat en el sistema al qual podem contactar via API per realitzar escanejos de tecnologies.

En segon lloc, s'ha creat la classe **TechnologiesAnalyzer** i s'han implementat els mètodes abstractes de la classe **BaseAnalyzer**:

- **get_target**: Es retorna la URL amb el protocol, ja que és la forma en la que s'ha de passar a **Wappalyzer**.
- **get_analysis_model**: Es retorna la classe **TechnologiesAnalysis**.
- **analyze**: Realitza una petició al microservei que hem creat per escanejar les tecnologies de la pàgina web. Després, filtra els resultats i els torna en format de diccionari de python.

Finalment, s'ha implementat el mètode **build** de la classe **TechnologiesAnalysis** de forma que transforma el resultat de l'analitzador en els registres adequats per aquest tipus d'anàlisi (TechnologiesAnalysis, WebsiteTechnology, Technology i Category).

La darrera cosa que hem de tenir en compte dels analitzadors és que la seva execució pot tardar varis minuts en alguns casos. Això fa que s'hagi de cercar una solució asíncrona per no bloquejar les peticions entrants. És en aquest punt on entra **Celery**. Gràcies a **Celery** podem derivar algunes execucions lentes a un **worker**, i per tant, es pot processar independentment del fil principal de l'aplicació.

D'aquesta forma, el cicle de vida dels anàlisis dins la nostra aplicació ha quedat així:

1. S'indica mitjançant l'API que es vol realitzar un nou anàlisi.
2. Es crea una petició (**AnalysisRequest**) amb la informació de l'usuari que vol realitzar l'anàlisi, la web que es vol analitzar i el tipus d'anàlisi que es vol fer.
3. S'executa la funció **analyze** asíncronament passant-li com argument la petició creada en el punt anterior. Com a resposta de l'API es retorna l'identificador de la petició mentre l'anàlisi es du a terme asíncronament.
4. Aquesta funció crea una nova instancia de l'analitzador requerit (en funció del tipus d'anàlisi indicat) i l'executa.
5. Aquesta execució pot acabar de 3 formes distintes:
    a) L'anàlisi acaba satisfactòriament, es creen els registres relacionats amb l'anàlisi i es modifica la petició per indicar que ha acabat correctament.
    b) L'anàlisi acaba degut a un error i es modifica la petició per indicar que s'ha produït un error.
    c) L'anàlisi supera el màxim temps permès i es modifica la petició per indicar que l'anàlisi ha superat el límit de temps.

4.4 API

Un cop ja hem vist com és el model i com funciona el cicle de vida dels analitzadors, anem a veure el disseny de l'API.

Com hem comentat a l'inici del capítol, hem utilitzat Django Rest Framework per desenvolupar l'API. Aquesta eina ens ha permès ser molt àgils a l'hora de desenvolupar l'API per la nostra aplicació. És increïble com en tan poques línies de codi es pot fer tant.

D'aquesta forma, s'han hagut de definir 3 tipus de dades distintes:

- **Rutes**: Les rutes constitueixen les adreces de l'aplicació amb les quals podem interactuar per obtenir la informació que necessitam o realitzar certes accions. Cada ruta representa un recurs distint, sobre el qual es poden realitzar accions de consulta, creació, modificació o eliminació.
- **Vistes**: Les vistes és on es defineix la lògica a seguir per cada acció dins cada ruta. És a dir, és el lloc on es determina el comportament de cada combinació de ruta i mètode HTTP. Per exemple, és on es defineix que s'ha de fer si es realitza una petició amb el mètode PUT sobre la ruta d'usuaris.
- **Serialitzadors**: Els serialitzadors són els encarregats de transformar les dades que tenim a base de dades en representacions aptes per ser transmeses a través de l'API.

Així doncs, mitjançant l'API es poden realitzar fins a 24 accions distintes. A continuació farem un breu repàs d'aquestes accions.

Usuaris

Ruta	Mètode	Acció
/users	POST	Crea un nou usuari
/token	POST	Obtenim un token per autenticar a l'usuari
/token/refresh	POST	Permet actualitzar el token un cop expirat

Aquest token és molt important, ja que l'hem de passar com a capçalera per autenticar a l'usuari en totes les peticions que veurem a continuació.

Anàlisis de webs

Ruta	Mètode	Acció
/website/analysis	GET	Llista les webs analitzades per un usuari
/website/analysis	POST	Indica que un usuari comença a monitoritzar una web
/website/analysis/<id>	GET	Obté l'estat d'una web analitzada per l'usuari
/website/analysis/<id>	PUT	Modifica dades relacionades amb una web analitzada

Peticions d'anàlisi

Ruta	Mètode	Acció
/website/analysis/<id>/requests	GET	Llista les peticions d'anàlisi d'una web
/website/analysis/<id>/requests/<id>	GET	Obté l'estat d'una petició d'anàlisi

Anàlisis de servidor

Ruta	Mètode	Acció
/website/analysis/<id>/server	GET	Llista els anàlisis de servidor d'una web
/website/analysis/<id>/server	POST	Realitza un nou anàlisi de servidor sobre la web indicada
/website/analysis/<id>/server/<id>	GET	Obté les dades d'un anàlisi de servidor

Anàlisis de vulnerabilitats

Ruta	Mètode	Acció
/website/analysis/<id>/vulnerabilities	GET	Llista els anàlisis de vulnerabilitats d'una web
/website/analysis/<id>/vulnerabilities	POST	Realitza un nou anàlisi de vulnerabilitats
/website/analysis/<id>/vulnerabilities/<id>	GET	Obté les dades d'un anàlisi de vulnerabilitats

Anàlisis de registrant

Ruta	Mètode	Acció
/website/analysis/<id>/registrant	GET	Llista els anàlisis de registrant d'una web
/website/analysis/<id>/registrant	POST	Realitza un nou anàlisi de registrant
/website/analysis/<id>/registrant/<id>	GET	Obté les dades d'un anàlisi de registrant

Anàlisis de subdominis

Ruta	Mètode	Acció
/website/analysis/<id>/subdomains	GET	Llista els anàlisis de subdominis d'una web
/website/analysis/<id>/subdomains	POST	Realitza un nou anàlisi de subdominis
/website/analysis/<id>/subdomains/<id>	GET	Obté les dades d'un anàlisi de subdominis

Anàlisis de tecnologies

Ruta	Mètode	Acció
/website/analysis/<id>/technologies	GET	Llista els anàlisis de tecnologies d'una web
/website/analysis/<id>/technologies	POST	Realitza un nou anàlisi de tecnologies
/website/analysis/<id>/technologies/<id>	GET	Obté les dades d'un anàlisi de tecnologies

5. Client web

En aquest capítol s'analitza en detall l'aplicació del client web. En la secció 5.1 repassam les tecnologies utilitzades per l'aplicació. En la secció 5.2 veim com es produeix la comunicació amb l'API. Finalment, en la secció 5.3 es mostra el disseny de la web i s'explica el funcionament dels components.

5.1 Tecnologies

A continuació farem un repàs de les tecnologies més importants que s'han usat pel desenvolupament del client web.

Docker

Al igual que en l'aplicació core utilitzam docker per desenvolupar software en un entorn aïllat i després desplegar-lo fàcilment sense possibilitats de conflictes.

Express

L'estructura principal de l'aplicació és Express. Express és un framework web basat en JavaScript que permet desenvolupat webs de forma molt ràpida i elegant. És el framework web preferit per la comunitat de JavaScript. És molt similar a Django quant a la facilitat per desenvolupar sobre aquest framework.

React

React és una llibreria per JavaScript que permet crear interfícies d'usuari basades en components. Aquesta llibreria permet desenvolupar molt àgilment aplicacions d'una sola pàgina com la nostra. Facilita molt dissenyar interaccions dins la pàgina.

Material UI

Per evitar haver de maquetar des de zero la pàgina web hem utilitzat aquest framework CSS. Gràcies a aquest framework hem pogut dedicar molt poc temps a la maquetació ja que t'ho dona quasi tot fet. A més, és el framework CSS òptim per treballar amb React.

Axios

Axios és una llibreria que ens permet fer peticions HTTP en entorns JavaScript. És la més famosa de la comunitat, i de fet, és molt fàcil d'emprar.

5.2 Connector de l’API

Abans de veure com s’ha dissenyat la web i quins són els seus components, és important analitzar com es fa la comunicació amb l’API.

Com hem comentat en la secció anterior, s’utilitza **axios** per realitzar les peticions HTTP dins la web. Ara bé, no és gens recomanable utilitzar **axios** a pel cada vegada que haguem de fer una consulta a l’API. A continuació podem veure alguns dels motius:

1. **L’API pot canviar**: Un moment donat podem decidir fer petites modificacions a l’API. Si hem utilitzat **axios** a pel, haurem de fer tantes modificacions com vegades cridem als endpoints modificats.
2. **Possibles errors**: Si ens equivoquem en com feim una petició, segurament ens haurem equivocat en moltes altres i les haurem de cercar i modificar.
3. **Llegibilitat**: Òbviament, realitzar peticions a endpoints per tota l’aplicació fa que sigui un poc difícil entendre el context.

Per aquest motiu s’ha creat un client per l’API dins la pròpia aplicació. Aquest client bàsicament utilitza **axios** però ho presenta amb un nivell d’abstracció. D’aquesta forma podem definir les crides als endpoints un sol pic i despreocupar-nos de quines són les rutes i com reben els paràmetres.

Per fer-ho encara més còmode, s’han creat tantes classes com recursos té l’API. Així doncs, per crear un usuari tenim la classe **User** i per crear una petició d’anàlisi tenim la classe **AnalysisRequest**.

A continuació podem veure les classes que s’han creat per interactuar amb l’API:

<u>User</u>

- **create**: Permet crear un usuari
- **get**: Obté les dades d’un usuari
- **createToken**: Retorna un token per autenticar a l’usuari
- **refreshToken**: Permet actualitzar el token de l’usuari

<u>WebsiteAnalysis</u>

- **list**: Llista les webs analitzades per l’usuari
- **create**: Indica que l’usuari vol analitzar una web
- **get**: Retorna les dades d’una web analitzada per l’usuari

AnalysisRequest

- **list**: Llista les peticions d'anàlisi d'una web
- **get**: Obté les dades d'una petició d'anàlisi

ServerAnalysis

- **list**: Llista els anàlisis de servidor d'una web
- **analyze**: Realitza un nou anàlisi de servidor a una web
- **get**: Obté les dades d'un anàlisi de servidor

VulnerabilitiesAnalysis

- **list**: Llista els anàlisis de vulnerabilitats d'una web
- **analyze**: Realitza un nou anàlisi de vulnerabilitats a una web
- **get**: Obté les dades d'un anàlisi de vulnerabilitats

RegistrantAnalysis

- **list**: Llista els anàlisis de registrant d'una web
- **analyze**: Realitza un nou anàlisi de registrant a una web
- **get**: Obté les dades d'un anàlisi de registrant

SubdomainsAnalysis

- **list**: Llista els anàlisis de subdominis d'una web
- **analyze**: Realitza un nou anàlisi de subdominis a una web
- **get**: Obté les dades d'un anàlisi de subdominis

TechnologiesAnalysis

- **list**: Llista els anàlisis de tecnologies d'una web
- **analyze**: Realitza un nou anàlisi de tecnologies a una web
- **get**: Obté les dades d'un anàlisi de tecnologies

5.3 Disseny i components

Fins ara hem vist com s'han seleccionat els analitzadors, com s'ha dissenyat l'arquitectura i l'estructura de l'aplicació core. Doncs bé, ara ha arribat el moment de veure l'aspecte del client web, que serà la part visible de tot allò que hem fet fins ara.

Com s'ha vist en l'apartat de tecnologies, hem utilitzat React pel disseny de la interfície. Amb React el procés d'estructuració d'una web és molt simple, ja que es creen components amb la lògica auto-continguda que es poden reutilitzar per tots els llocs que nosaltres volguem. Aquest aspecte és important, ja que a mesura que veiem les distintes parts de l'aplicació web anirem analitzant per quins components estan formades.

Així doncs, podem començar a veure quin aspecte té el client web.

Inici de sessió

Només accedir a la pàgina web apareix un diàleg per iniciar sessió. Aquest diàleg ens demana el nom d'usuari i la contrasenya. Un cop els introduïm, fa una petició a l'API per tal d'obtenir el token d'autenticació. Si les dades són vàlides, obtenim el token i som redirigits al dashboard de l'aplicació. Però si no són vàlides, apareix un missatge d'error indicant que ens hem equivocat.

Figura 5.1: Inici de sessió de l'aplicació web

En aquest cas, tot el que veiem està format per un sol component. Aquest component es nom **SignIn** i tan sols s'utilitza en aquest punt de l'aplicació.

Dashboard

Un cop hem aconseguit iniciar sessió satisfactòriament som redirigits al dashboard principal de l'aplicació.

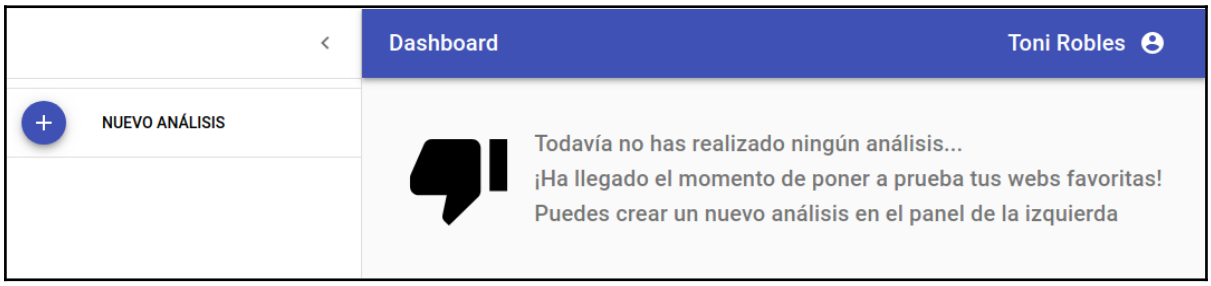


Figura 5.2: Dashboard principal de l'aplicació

En aquest dashboard és on gestionam totes les funcionalitats de l'aplicació. El dashbord representa un component, concretament el component **Dashboard**. Ara bé, aquest component està format per varis components que anirem veient més endavant.

D'altra banda, si ens fixem bé, apareix un missatge que ens diu que encara no hem realitzat cap anàlisi i ens indica com fer-ho. Aquest missatge també és un component anomenat **NoAnalysisBoard**.

Llistat de webs analitzades

A l'esquerra tenim un panell on apareix el llistat de webs analitzades. Aquest panell és un component anomenat **WebsiteList**. Com de moment no hem analitzat cap web, el panell està buit. Tot i així, sempre hi ha un botó per començar a analitzar una web. Si feim clic sobre el botó en qüestió apareix un input per indicar quina web volem analitzar.

Figura 5.3: Input per indicar quina web es vol analitzar

Quan es confirma el contingut d'aquest input s'envia una petició a l'API indicant que es vol analitzar aquesta web. En aquest punt, l'API pot indicar que s'ha creat correctament o que la web no existeix. En cas de no existir, el contorn de l'input es pinta de color vermell. Ara bé, si la web existeix, es crea un nou registre i s'actualitza el panell amb la web que hem indicat.

Normalment aquest procés és molt ràpid. Tot i així, en ocasions les peticions a l'API es poden veure afectades per les condicions de la xarxa i tardar un poc més de l'esperat. Per tant, per indicar a l'usuari que la petició s'està realitzant, apareix la típica animació de càrrega de contingut.

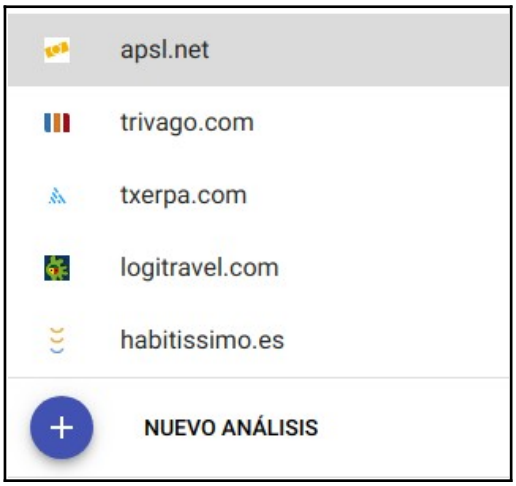


Figura 5.4: Llistat de webs analitzades

Si ens fixem en la figura 5.4 podem veure que una de les webs està ressaltada. Això vol dir que el contingut del dashboard farà referència a aquesta web. A continuació veurem en detall aquest aspecte.

Detall d'una web analitzada

Ara acabam de veure que en el panell de l'esquerra del dashboard apareixen les webs analitzades per l'usuari. A més, acabam de comentar que una d'elles està seleccionada. Doncs bé, a la resta del dashboard podem veure informació relativa a aquesta web. Aquesta informació representa un component anomenat **WebsiteDetail**.

Figura 5.5: Detall d'una web quan encara no s'ha realitzat cap tipus d'anàlisi

Com es pot observar a la figura 5.5, quan s'indica que es vol analitzar una web apareix al dashboard un panell amb 6 àrees que representen els 6 tipus d'anàlisis que podem realitzar. Com encara no hem realitzat cap tipus d'anàlisi, tots ens indiquen que no l'hem executat.

Cada una d'aquestes àrees és un component distint, les quals repassarem a continuació. D'altra banda, són molt similars entre elles. De fet, el contingut que mostren sempre segueix el mateix format, però la lògica és distinta. Per aquest motiu tots els components utilitzen un component comú per representar les dades anomenat **AnalysisPreview**.

Resum d'un anàlisi de servidor

La primera de les àrees que hem mencionat abans consisteix en el resum d'un anàlisi de servidor. El component en qüestió s'anomena **ServerPreview**. En la figura 5.5 té aquell aspecte ja que no s'ha realitzat cap anàlisi. Ara bé, si feim clic sobre el botó d'analitzar i esperem uns segons, apareix el resum de l'anàlisi realitzat.

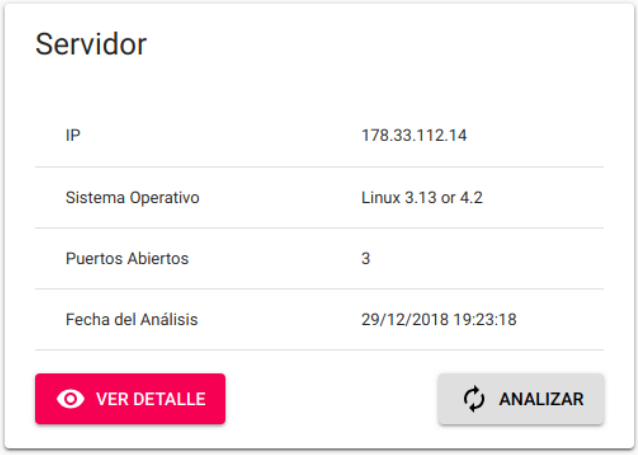


Figura 5.6: Resum d'un anàlisi de servidor

Dins aquest resum podem veure la IP, el sistema operatiu, el nombre de ports oberts i la data de l'anàlisi. A més, podem fer un nou anàlisi de forma que s'actualitzarien les dades. Finalment, hi ha un botó que ens permet veure l'anàlisi en detall, però aquest aspecte el veurem més endavant.

Resum d'un anàlisi de vulnerabilitats possibles

La segona de les àrees que hem mencionat abans consisteix en el resum d'un anàlisi de vulnerabilitats possibles. El component en qüestió s'anomena **VulnerabilitiesPreview**.

Figura 5.7: Resum d'un anàlisi de vulnerabilitats possibles

Un cop hem realitzat el primer anàlisi el component tindrà un aspecte similar al de la figura 5.7. Dins aquest resum podem veure el nombre de possibles vulnerabilitats detectades, quantes d'aquestes són greus, quantes d'aquestes són recents i la data de l'anàlisi. Les accions possibles són les mateixes que en el cas anterior.

Resum d'un anàlisi de vulnerabilitats confirmades

Per les vulnerabilitats confirmades s'utilitza el mateix component que per les vulnerabilitats possibles. La única diferència és que se li indica mitjançant un paràmetre que les vulnerabilitats que ha de gestionar són les confirmades.

Resum d'un anàlisi de tecnologies

La quarta de les àrees que hem mencionat abans consisteix en el resum d'un anàlisi de tecnologies. El component en qüestió s'anomena **TechnologiesPreview**.

Figura 5.8: Resum d'un anàlisi de tecnologies

Un cop hem realitzat el primer anàlisi el component tindrà un aspecte similar al de la figura 5.8. Dins aquest resum podem veure el nombre de tecnologies detectades, quantes d'aquestes estan confirmades, quantes d'aquestes tenen versió i la data de l'anàlisi. Les accions possibles són les mateixes que en els casos anteriors.

Resum d'un anàlisi del registrant

La quinta de les àrees que hem mencionat abans consisteix en el resum d'un anàlisi del registrant. El component en qüestió s'anomena **RegistrantPreview**.

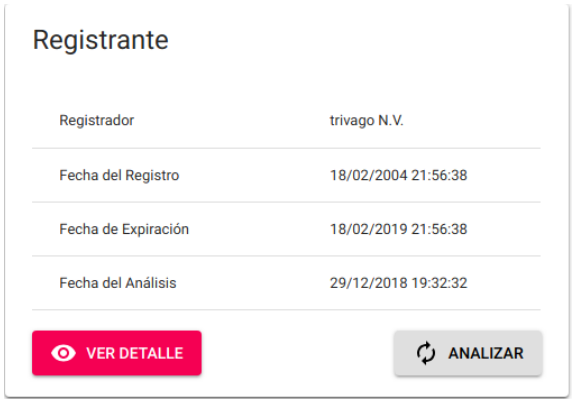


Figura 5.9: Resum d'un anàlisi del registrant

Un cop hem realitzat el primer anàlisi el component tindrà un aspecte similar al de la figura 5.9. Dins aquest resum podem veure qui és el registrador, la data de registre, la data d'expiració i la data de l'anàlisi. Les accions possibles són les mateixes que en els casos anteriors.

Resum d'un anàlisi de subdominis

La darrera de les àrees que hem mencionat abans consisteix en el resum d'un anàlisi de subdominis. El component en qüestió s'anomena **SubdomainsPreview**.

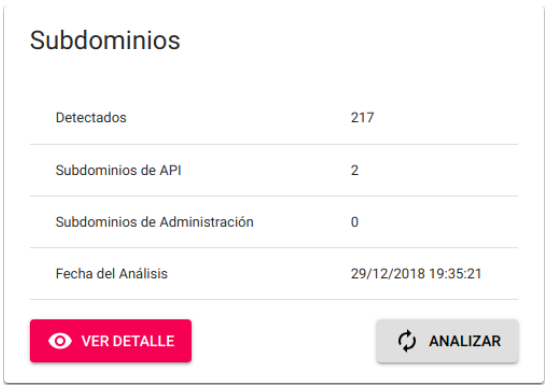


Figura 5.10: Resum d'un anàlisi de subdominis

Un cop hem realitzat el primer anàlisi el component tindrà un aspecte similar al de la figura 5.10. Dins aquest resum podem veure quants subdominis han estat detectats, quants corresponen a una API, quants corresponen a un panell d'administració i la data de l'anàlisi. Les accions possibles són les mateixes que en els casos anteriors.

Selector del detall d'un anàlisi

Tots els resums dels anàlisis que hem vist fins ara tenien un botó per veure el detall. Aquest botó el que fa és incloure dins el dashboard, concretament a la part inferior, un selector de detalls d'anàlisi. Aquest selector permet navegar entre tots els detalls dels anàlisis realitzats.

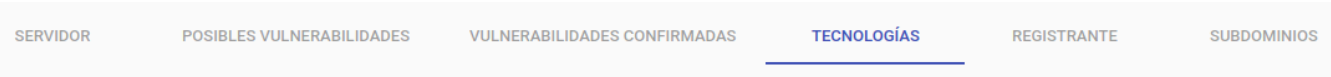


Figura 5.11: Selector de detall d'un anàlisi

A més, quan indicam que volem veure un detall, es desplega automàticament aquell detall en el selector. Aquesta part de l'aplicació es correspon al component anomenat **WebsiteAnalysisSelector**. A continuació veurem els components d'aquests detalls d'anàlisi.

Detall de l'anàlisi de servidor

Quan indicam que volem veure el detall de l'anàlisi de servidor apareix un panell a continuació del selector amb tota la informació. Podem veure la IP, el sistema operatiu i la informació dels ports detectats. A més, hi ha un historial mitjançant el qual podem revisar qualsevol dels anàlisis anteriors que hem realitzat.

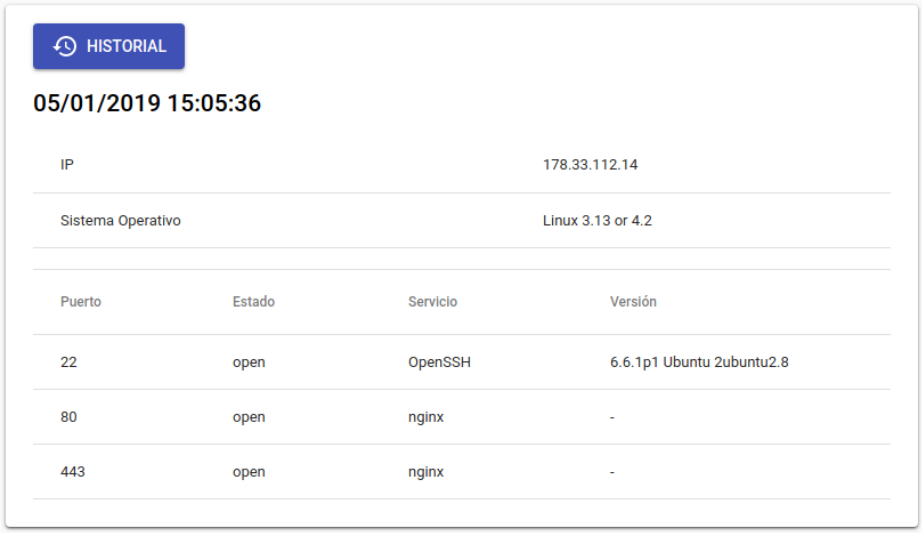


Figura 5.12: Detall d'un anàlisi de servidor

Detall de l’anàlisi de vulnerabilitats

Quan indicam que volem veure el detall de l’anàlisi de vulnerabilitats apareix un panell a continuació del selector amb tota la informació. En aquest panell apareix una taula paginada on a cada pàgina apareixen els detalls de cada vulnerabilitat. En primer lloc, es mostra un resum de la vulnerabilitat indicant el CVE, el nom del producte, la descripció i altres dades d’interès. En segon lloc, en els casos en els que es disposi de la informació, es mostra una taula amb la informació del CVSS. Finalment, es llisten les distintes referències que es té de la vulnerabilitat.

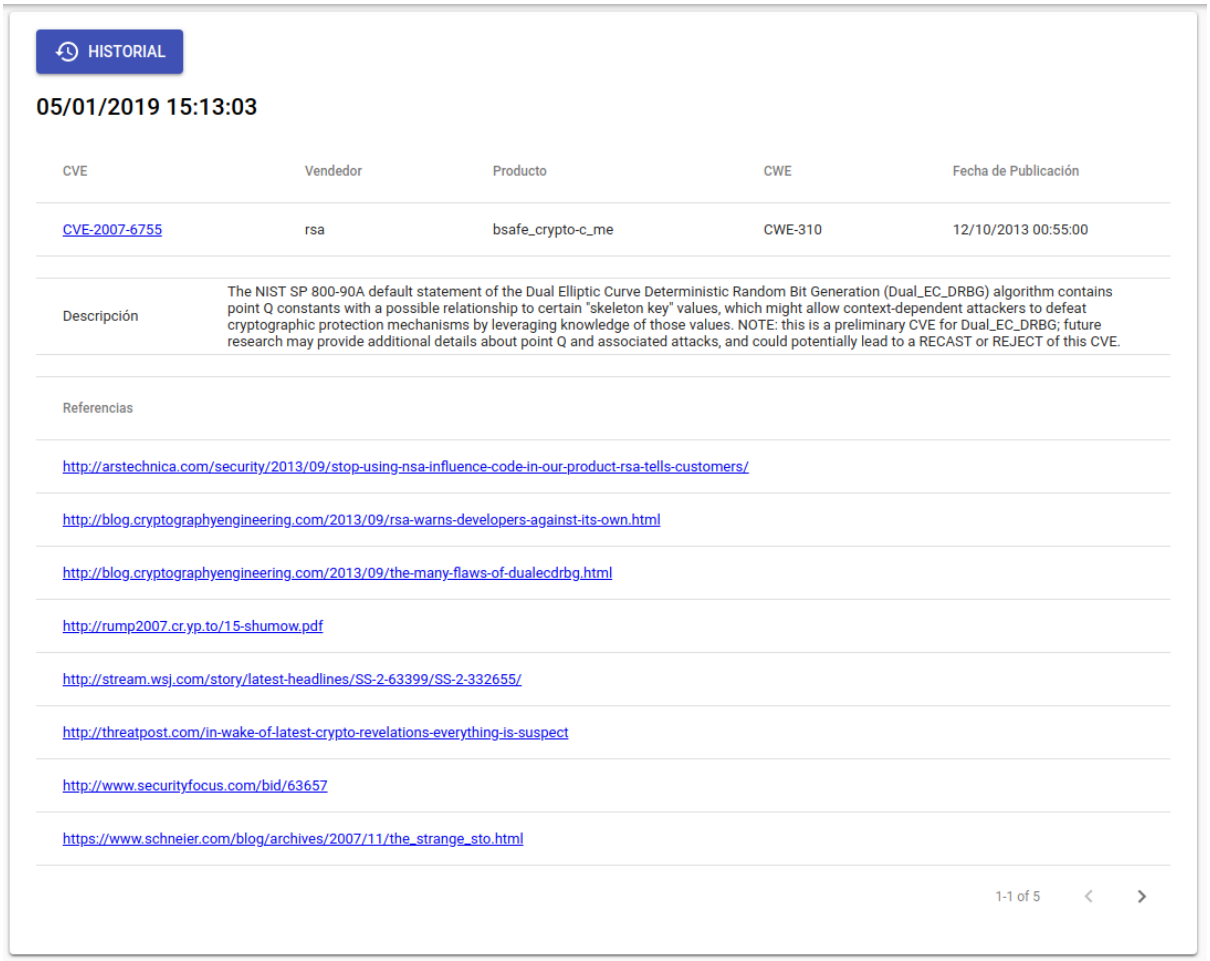


Figura 5.13: Detall d’un anàlisi de vulnerabilitats

Detall de l’anàlisi de tecnologies

Quan indicam que volem veure el detall de l’anàlisi de tecnologies apareix un panell a continuació del selector amb tota la informació. En aquest panell es llisten les tecnologies detectades indicant el nom, la versió, la categoria i el grau de confiança. A més, disposam dels enllaços a la pàgina principal de cada una de les tecnologies. D’altra banda, al igual que en els altres anàlisis, es pot consultar l’historial.

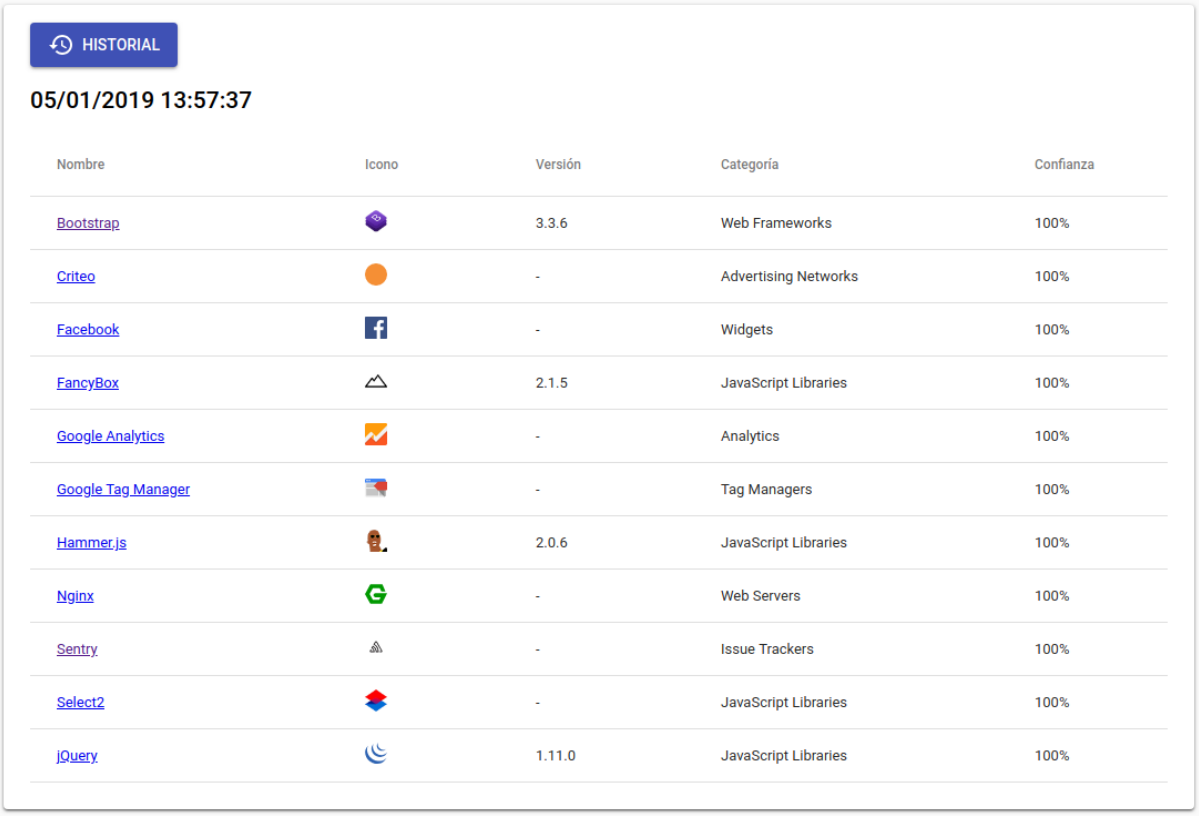


Figura 5.14: Detall d'un anàlisi de tecnologies

Detall de l'anàlisi del registrant

Quan indicam que volem veure el detall de l'anàlisi del registrant apareix un panell a continuació del selector amb tota la informació. Aquesta informació inclou el nom del registrant, data de registre, data d'expiració, adreça, etc.

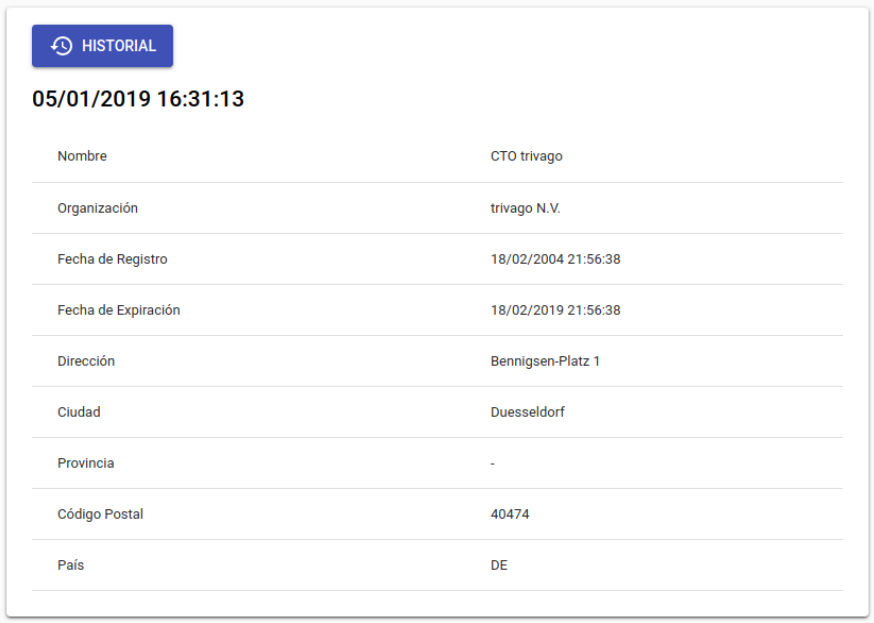


Figura 5.15: Detall d'un anàlisi del registrant

Detall de l’anàlisi de subdominis

Quan indicam que volem veure el detall de l’anàlisi de subdominis apareix un panell a continuació del selector amb tota la informació. En aquest panell hi ha una taula paginada on a cada pàgina hi ha un subconjunt del llistat total de subdominis.

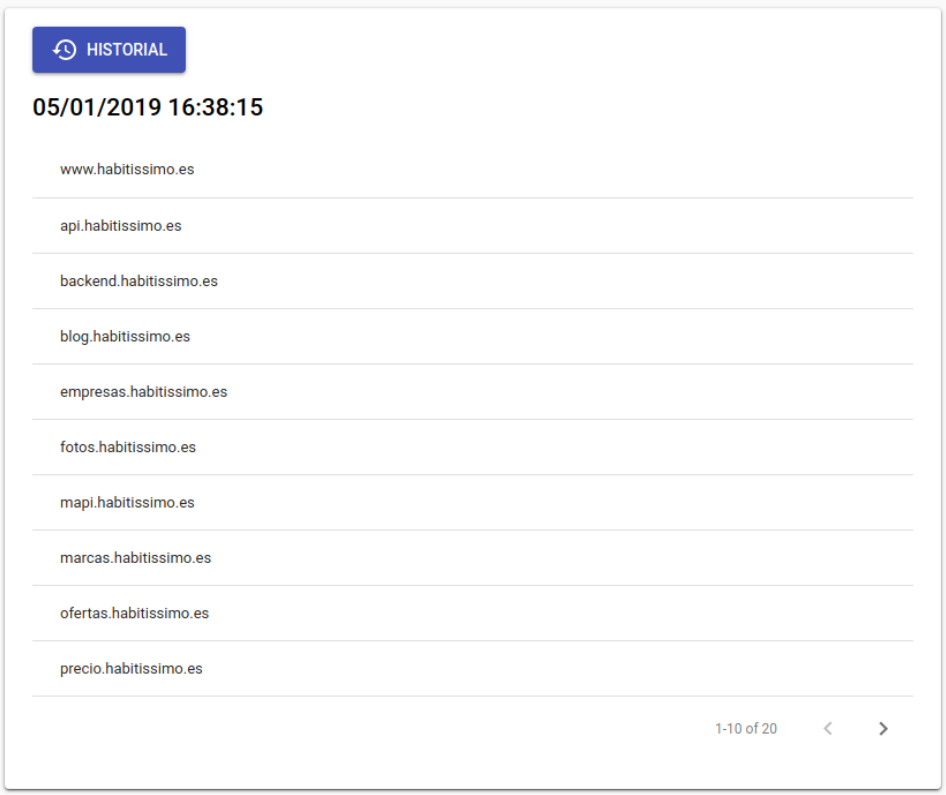


Figura 5.16: Detall d’un anàlisi de subdominis

6. Conclusions

En aquest capítol s'analitzen els resultats obtinguts i es treuen les conclusions. En la secció 6.1 es dona una opinió personal de com s'ha desenvolupat el projecte i quines sensacions ens ha donat. En la secció 6.2 s'analitzen els resultats obtinguts. En la secció 6.3 s'analitza la dedicació del projecte i es compara amb la planificació establerta. Finalment, en la secció 6.4 es veuen possibles ampliacions i millores del projecte.

6.1 Opinió personal

Aquest projecte és el que m'ha fet sentir més orgullós fins ara. Les sensacions finals de tot el treball dedicat són excel·lents.

Gràcies a aquest projecte he aprés molt sobre el món de la seguretat informàtica. També he conegut moltíssimes eines relacionades amb la seguretat que desconeixia. Algunes m'han sorprès molt.

D'altra banda, aquest projecte m'ha permès desenvolupar tot el meu potencial quant al desenvolupament web. Feia molt de temps que cercava l'oportunitat de desenvolupar un projecte tan gran des de zero. Gràcies a això, he pogut posar en pràctica els meus coneixements de backend i també aprendre molt de la part de frontend.

En resum, aquest projecte m'ha permès millorar molt les meves competències quant a la seguretat informàtica i consolidar les meves competències de desenvolupament web.

6.2 Anàlisi dels resultats

L'objectiu d'aquest projecte era aconseguir un sistema completament modular a partir del qual poder integrar distintes eines d'anàlisi de vulnerabilitats i altres analitzadors similars.

En aquest sentit, els resultats són més que satisfactoris. Hem aconseguit una API molt completa que permet integrar nous anàlisis i modificar els existents de forma molt senzilla. A més, s'ha desenvolupat un client web que utilitza aquesta API demostrant tot el potencial de l'aplicació. De fet, al tractar-se d'una API, ens permet desenvolupar qualsevol tipus de client que faci ús d'ella sense refer la lògica.

D'altra banda, els objectius quant al client web també s'han aconseguit. L'aplicació permet analitzar les vulnerabilitats de les pàgines webs indicades. Aquests anàlisis es queden registrats i vinculats al nostre usuari, de forma que els podem consultar en qualsevol moment. A més, també tenim un historial per consultar tots els anàlisis anteriors que hem realitzat.

6.3 Dedicació

Setmanes	Tasca
1 - 3	Investigació de les distintes eines per analitzar vulnerabilitats
4	Definició de l'arquitectura del projecte
5	Disseny de l'API
6 - 10	Implementació de l'API i integració dels analitzadors inicials
11	Disseny de la interfície web
12 - 13	Implementació del client web
14 - 15	Documentació

La dedicació real ha estat molt similar a la planificació realitzada. Les poques variacions que s'han produït són les següents:

1. S'ha dedicat una setmana menys a la investigació d'analitzadors ja que amb aquest temps es varen explorar totes les opcions.
2. S'han dedicat dues setmanes més a la integració dels analitzadors ja que alguns varen donar problemes.
3. La implementació del client web s'ha realitzat amb una setmana menys, ja que el seu desenvolupament ha estat més senzill del que s'esperava.

6.4 Ampliacions i millores

Aquest projecte s'ha realitzat de forma que permet moltes ampliacions i millores. A continuació podem veure algunes d'elles:

- Millorar l'analitzador de vulnerabilitats perquè mostri de forma més visual quines vulnerabilitats són més greus.
- Crear una aplicació mòbil que es connecti a l'API i permeti fer el mateix que el client web.
- Afegir nous tipus d'anàlisi.
- Millorar la representació dels resultats dels anàlisis utilitzant gràfiques.
- Assignar a la web un nivell d'alerta personalitzat en funció dels resultats dels anàlisis.
- Crear un bot de telegram que es connecti a l'API i permeti consultar els resultats d'un anàlisi.

7. Bibliografia

Wikipedia [en línia]. Palma:
nmap. [Data de consulta: 5 d'octubre de 2018]
<https://es.wikipedia.org/wiki/Nmap>

CIRT [en línia]. Palma:
nikto. [Data de consulta: 7 d'octubre de 2018]
<https://cirt.net/nikto2-docs/>

Wikipedia [en línia]. Palma:
Nessus. [Data de consulta: 7 d'octubre de 2018]
<https://es.wikipedia.org/wiki/Nessus>

Wikipedia [en línia]. Palma:
OpenVAS. [Data de consulta: 8 d'octubre de 2018]
<https://es.wikipedia.org/wiki/OpenVAS>

Wikipedia [en línia]. Palma:
whois. [Data de consulta: 8 d'octubre de 2018]
<https://es.wikipedia.org/wiki/WHOIS>

GitHub [en línia]. Palma:
nmap-vulners. [Data de consulta: 9 d'octubre de 2018]
<https://github.com/vulnersCom/nmap-vulners>

GitHub [en línia]. Palma:
nmap-vulscan. [Data de consulta: 9 d'octubre de 2018]
<https://github.com/scipag/vulscan>

Wikipedia [en línia]. Palma:
BuiltWith. [Data de consulta: 10 d'octubre de 2018]
<https://en.wikipedia.org/wiki/BuiltWith>

GitHub [en línia]. Palma:
WAScan. [Data de consulta: 12 d'octubre de 2018]
<https://github.com/m4ll0k/WAScan>

GitHub [en línia]. Palma:
sublist3r. [Data de consulta: 12 d'octubre de 2018]
<https://github.com/aboul3la/Sublist3r>

freeCodeCamp [en línia]. Palma:
Client Side VS Server Side Rendering. [Data de consulta: 21 de novembre de 2018]
<https://bit.ly/2AVlGrE>