Escàner de vulnerabilitats web

Nom Estudiant: Eric Bujeque Escorriola

Programa: Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Nom Consultor: Jordi Duch i Agusti Solanas

Centre: Universitat Oberta de Catalunya

Data Lliurament: 15-06-2015

Llicències alternatives (triar alguna de les següents i substituir la de la pàgina anterior)

A) Creative Commons:

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-SenseObraDerivada 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement-CompartirIgual 3.0 Espanya de Creative Commons

Aquesta obra està subjecta a una llicència de Reconeixement 3.0 Espanya de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © ANY EL-TEU-NOM.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free

C) Copyright

FITXA DEL TREBALL FINAL

Títol del treball:	*Escàner de vulnerabilitats web.*
Nom de l'autor:	*Eric Bujeque Escorriola*
Nom del consultor:	*Jordi Duch i Agusti Solanas*
Data de lliurament (mm/aaaa):	*06/2015*
Àrea del Treball Final:	*Seguretat en la programació*
Titulació:	Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Resum del Treball (màxim 250 paraules):

Escàner de vulnerabilitats web especialitzats en l'entrada de dades dels usuaris, ja sigui per URL, per formulari o tipus petició HTTP. Els usuaris registrats poden escanejar les web desitjades passant un seguit de tests que poden escollir. A més, poden especificar quins paràmetres en concret es volen escanejar. L'aplicació mostra els resultats dels tests i els guarda en la base de dades.

Alguns dels test de vulnerabilitats que es passen fan servir aplicacions de tercers, que l'aplicació executa i recull els resultats per mostrar-lo a l'usuari. Altres estan programats en la pròpia aplicació.

Abstract (in English, 250 words or less):

This project is a Vulnerability web scanner focused in analysing user input data from URL, forms or HTTP requests. The registered users can scan webpage vulnerabilities using a list of selected tests. They can also test and modify specific webpage parameters for the scan. The results are shown on the the application page and are saved into the database.

Some of the vulnerability tests are built-in and some use third party software, in which case they are run and results are processed by this application.

Paraules clau (entre 4 i 8):

Escàner, vulnerabilitats, web, xss, sqlinjection, html, python

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

Hi ha una clara tendència a fer servir la tecnologia web, no només per les aplicacions tradicionals com bitàcoles, fòrums, pàgines personals, corporatives etc... sinó també per les aplicacions. Des de fa uns anys, la tecnologia web li està guanyant terreny a les aplicacions d'escriptori, ja que són més fàcils de programar i de fer compatibles per a qualsevol sistema operatiu.

Existeixen aplicacions web que guarden documents o informació confidencial, que en cas de ser vulnerables quedarien a l'abast de qualsevol intrús. A més, no es tracta només de la pròpia web i el que guardi, una pàgina vulnerable i degudament atacada pot acabar infectant a milers de visitants.

Cada dia apareixen nous gestors de continguts, programats en diferents llenguatges i sobre *frameworks* diferents. És necessari passar un seguit de proves de seguretat en cada aplicació web que pengem, per tal d'assegurar-nos que no son vulnerables.

Per aquesta finalitat hi ha un alt conjunt d'eines efectives, gratuïtes i de codi obert molt ampli, que passen diversos tests especialitzats a les webs. A més, existeixen programes de pagament (com per exemple *Metasploit*) que passen un seguit de tests a una web, amb diferents nivells d'intrusió, i emmagatzemen els resultats.

La idea d'aquest projecte és fer servir aquestes eines gratuïtes i desenvolupar alguns tests propis ajuntant-los en una única aplicació web, i que a més sigui capaç d'emmagatzemar els resultats per a la seva posterior comparació.

1.2 Objectius del Treball

Els objectius del desenvolupament d'aquest projecte són els següents:

- Desenvolupar una aplicació que permeti escanejar pàgines i analitzar els resultats.
- Que l'aplicació sigui fàcil d'utilitzar, que no sigui necessari aprendre les comandes que utilitzen programes com *SQLmap*, sinó que tot això sigui transparent.
- Aconseguir un disseny agradable i intuïtiu.
- Que permeti guardar els resultats per la seva posterior comparació.

A més dels objectius propis del projecte, és pretén aprofundir en aquest tipus de vulnerabilitats i aprendre a desenvolupar una aplicació web en *Python* i *Django*.

1.3 Enfocament i mètode seguit

Les possibles estratègies a seguir eren varies:

- **Modificar una aplicació existent**. Per exemple, millorar algun altre projecte o desenvolupar algun *plugin* especialitzat en vulnerabilitats web per a aplicacions més grans i de qualitat, que ja tenen fama de funcionar bé i ser força efectives.

- **Desenvolupar una aplicació nova.** És a dir, crear de zero una aplicació que efectues tots els tests de vulnerabilitats web que existeixen en l'àmbit d'introducció de dades per part de l'usuari, voluntària o involuntàriament.

- **Desenvolupar una aplicació nova que es recolzi de programes existents**. Hi ha aplicacions especialitzades en certs aspectes webs que són difícils de millorar, per tant una altre alternativa per a desenvolupar el projecte és ajuntar les dues opcions anteriors.

Finalment s'ha decidit la última opció. Desenvolupar una aplicació nova totalment de zero requeria un volum de treball molt alt, que no es podia assumir si es pretenia que la aplicació fos competitiva. Per altra banda, desenvolupar els algoritmes per fer les proves de vulnerabilitats en els diferents aspectes d'una web, aportava un coneixement difícil de rebutjar. Per tant, algunes proves es fan a partir d'eines ja creades i altres s'han programat de zero.

Un altre enfocament que s'ha hagut de rumiar és sobre com fer la aplicació:

- **Aplicació d'escriptori**: Aquesta opció és més amigable amb el sistema de cara a executar programes de tercers, per exemple basats en línees de comandes Linux, però per fer-ho compatible amb línea de comandes Windows es tornava un problema. D'igual forma que el fet d'haver-ho de compilar amb un compilador diferent per a cada sistema operatiu. Es va desestimar aquesta opció bàsicament per temes de compatibilitat.

- **Aplicació web**: Aquesta és la opció triada ja que no oferia cap inconvenient en temes de compatibilitat entre sistemes operatius. Podria resultar un problema el executar comandes del sistema operatiu, però triant un llenguatge que facilita aquest tipus de tasca, com *Python*, mitiguem aquest inconvenient.

Aquesta última opció ha sigut la més adequada per aconseguir els objectius marcats, ja que ens permet desenvolupar una aplicació capaç d'escanejar una web en busca de vulnerabilitats, permet passar paràmetres sense problemes, es poden guardar els resultats fàcilment en la base de dades i ofereix millors opcions de disseny de la interfície.

1.4 Planificació del Treball

Per la planificació del treball s'ha de tenir clar quins son els recursos necessaris per a dur a terme el projecte i realitzar una planificació temporal de les tasques.

Els recursos necessaris són:

- Un ordinador amb Linux y servidor web Python.
- Un programador/analista.

Les diferents tasques a desenvolupar són:

- **Anàlisi de l'aplicació**
  - Anàlisi de requeriments de la aplicació.
  - Anàlisi de dades a tractar.
  - Anàlisi dels tests de vulnerabilitats a realitzar.
  - Anàlisi de seguretat i legalitat.
  - Documentació de l'anàlisi.

- **Preparació**
  - Adquisició de coneixements.
  - Lectura de la documentació.
  - Cerca d'aplicacions de tercers.

- **Disseny de l'aplicació**
  - Disseny de la base de dades.
  - Disseny de la estructura de l'aplicació.
  - Disseny gràfic de la aplicació.

- **Desenvolupament de l'aplicació**
  - Preparació de l'entorn de desenvolupament.
  - Configuració de la base de dades.
  - Maquetació del disseny gràfic de l'aplicació.
  - Programació de l'aplicació.
  - Programació dels tests de vulnerabilitats.

- **Proves**
  - Proves de seguretat.
  - Proves de funcionament de la aplicació.
  - Proves de funcionament dels tests.

- **Altres**
  - Generació de la memòria.
  - Generació de la presentació.
  - Tancament del projecte.

La planificació és la següent. Com que no cabia tot sencer s'ha hagut de partir.

Nombre	Fecha de ini...	Fecha de fin
Anàlisi de requeriments	2/03/15	4/03/15
Anàlisi de dades a tractar	5/03/15	9/03/15
Anàlisi de test a realitzar	10/03/15	13/03/15
Anàlisi de seguretat i legalitat	13/03/15	13/03/15
Adquisició de coneixements	16/03/15	24/03/15
Lectura de documentació	24/03/15	1/04/15
Cerca d'aplicacions de tercers	2/04/15	2/04/15
Disseny de la BBDD	7/04/15	8/04/15
Disseny de la estructura	9/04/15	13/04/15
Disseny gràfic de l'aplicació	14/04/15	16/04/15
Preparació de l'entorn	17/04/15	22/04/15
Configuració de la BBDD	23/04/15	23/04/15
Maquetació del disseny	24/04/15	28/04/15
Programació de l'aplicació	28/04/15	11/05/15
Programació dels tests	12/05/15	21/05/15
Proves de seguretat	22/05/15	25/05/15
Proves de funcionament	26/05/15	28/05/15
Proves dels tests	29/05/15	1/06/15
Generació de la memòria	2/06/15	10/06/15
Generació de la presentació	11/06/15	11/06/15
Tancament del projecte	12/06/15	15/06/15

Figura 1. Programació de tasques

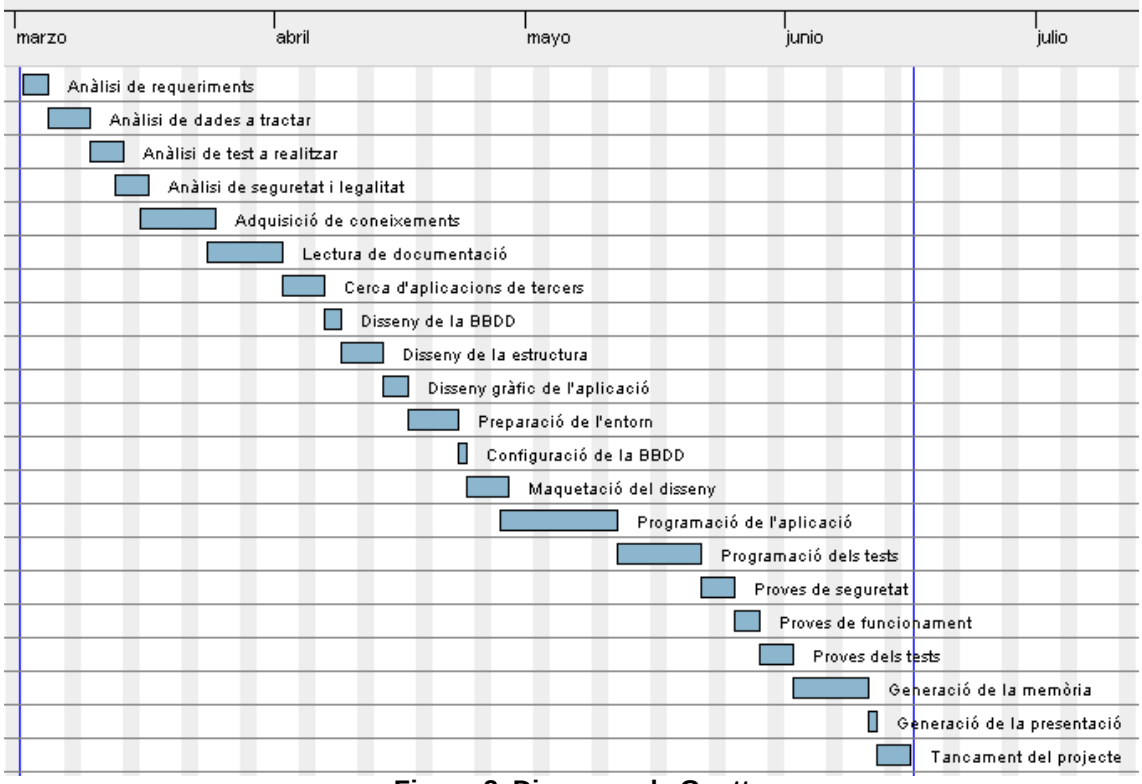


Figura 2. Diagrama de Gantt

1.5 Breu sumari del producte obtingut

El producte obtingut és un escàner de vulnerabilitats web especialitzat en l'entrada de dades dels usuaris, ja sigui per direcció del navegador, per formulari o tipus petició *HTTP*. Els usuaris registrats poden escanejar les web desitjades passant un seguit de tests de vulnerabilitats que poden escollir. A més, poden especificar quins paràmetres en concret es volen escanejar. L'aplicació mostra els resultats dels tests i els guarda en la base de dades.

Alguns dels test de vulnerabilitats que es passen fan servir aplicacions de tercers, que l'aplicació executa i recull els resultats per mostrar-lo a l'usuari. Altres estan programats en la pròpia aplicació fent peticions HTTP sobre la pàgina a escanejar.

1.6 Breu descripció dels altres capítols de la memòria

Aquesta memòria s'estructura en els següents capítols generals:

- **Fonaments teòrics:** Es tracta de la descripció dels coneixements bàsics que són necessaris per desenvolupar i entendre l'aplicació.

- **Anàlisi:** La aplicació és analitzada des del punt de vista funcional i no funcional.

    - **Funcional:** S'explica els requeriments tècnics, requeriments funcionals, els usuaris del sistema i finalment una explicació del diagrama de casos d'us de l'aplicació.

    - **No funcional:** S'analitza la compatibilitat amb els diferents navegadors, el compliment de la llei de protecció de dades i la accessibilitat.

- **Disseny:** Es centra en el disseny de la aplicació, des de la estructura del codi fins al model de dades i el disseny gràfic.

- **Implementació:** Es descriu tot el necessari per fer funcionar l'aplicació, a més d'explicar la seva estructura de carpetes i el seu funcionament.

- **Seguretat:** Informació referent a la seguretat pròpia de tota l'aplicació.

- **Conclusions:** Lliçons apreses, reflexió sobre el projecte, anàlisi crític del seguiment de la planificació i línees de treball futur per expandir la aplicació.

2. Fonaments teòrics

En aquest apartat es descriu tot allò que es dona per sabut i que no es detallarà en altres apartats de la memòria. També s'exposa el llenguatge i les eines utilitzades per fer el projecte i les seves alternatives.

2.2 Coneixements bàsics.

A continuació es llisten els coneixements bàsics que s'han de saber per la correcta comprensió d'aquesta memòria.

Linux

Sistema operatiu el qual el seu codi font pot ser utilitzat, modificat i distribuït lliurement per qualsevol.

Python

Llenguatge de programació interpretat amb una sintaxis que afavoreix un codi llegible. S'aconsella seguir uns estàndards per tal de que un codi sigui "*python friendly*", que vol dir fer els algorismes més fàcils d'entendre, fins i tot sense estar familiaritzat amb el llenguatge.

Django

Framework de desenvolupament web de codi obert, escrit en ***Python***, que respecta el patró de disseny *MVC* (Model - Vista - Controlador).

Framework

Podem definir un *framework* com un conjunt de llibreries, funcionalitats i eines ben estructurades, pensades per facilitar el treball de desenvolupament d'un software. Per exemple en aquest projecte s'ha fet servir el *framework* anomenat **jQuery**, que ens facilita molt la feina escrivint codi **Javascript**.

Base de dades

Són un conjunt de dades d'un mateix context emmagatzemades sistemàticament per a seu posterior us. Entre els sistemes més destacats de gestió de base de dades trobem **MySQL**, **PosgreSQL** i **Oracle**, entre molts altres. El sistema que s'ha fet servir en aquest projecte és el **SQLite**, un mini sistema de base de dades, lleuger i de fàcil ús.

CSS

Sigles de Cascading Style Sheets (Fulles d'estil en cascada). És un llenguatge usat per definir la presentació de pàgines web **HTML**. La seva funció principal es separar el disseny d'una pàgina del seu codi o estructura.

Servidor Web

És un programa informàtic que processa una aplicació del costat del servidor realitzant connexions bidireccionals o unidireccionals i síncrones

o asíncrones amb el client, a través del protocol **HTML**. El servidor web que es fa servir en aquest projecte és el que ve incorporat amb ***Django***.

Javascript

És un llenguatge de programació interpretat. S'executa en el client implementat com a part del navegador, permitent millores en la interfície d'usuari i pàgines web dinàmiques. Aquest llenguatge s'inclou dintre de l'**HTML**.

JQuery

Llibreria o *framework* de **JavaScript**.

SQLite

Sistema de gestió de base de dades relacional.

2.3 Software utilitzat i alternatives.

A continuació es mostra el software utilitzat i les seves respectives alternatives.

Programació

Per programar el software s'ha fet servir **Python**, un llenguatge de programació interpretat que cada vegada s'està fent servir més en el món web, per la seva facilitat i potencia. Es va escollir per sobre del PHP perquè oferia la possibilitat de generar un codi molt més net i algoritmes més amigables. A més, es tracta d'un llenguatge fàcil de fer servir, el servidor es gratuït i hi ha molta documentació.

Alternatives de programació

PHP: Ha sigut durant molts anys el llenguatge número 1 en webs, molts servidors incorporen un servidor *PHP* per defecte. Es un llenguatge molt permissiu a l'hora de definir variables (hi ha pocs conflictes entre tipus) i tolerant a errors. A més, al ser tant popular hi ha molta documentació al respecte.

Java: Llenguatge orientat a objectes, molt potent, multi plataforma amb servidor *apache* gratuït, molt popular i amb molta documentació. És menys permissiu que *PHP* o *Python* i més tediós de programar, sota el punt de vista personal.

Servidor

Pel servidor s'ha decidit anar per la via ràpida i fàcil, per aquest projecte es fa servir el servidor web que incorpora *Django*, que òbviament està preparat i optimitzat per fer córrer aquest *Framework*.

Alternativa al Servidor

Apache: Era sense dubte la alternativa més clara, ja que és força conegut gràcies a *PHP* i hi ha molta documentació al respecte. Per defecte no suporta *Python*, s'ha d'instal·lar i configurar un mòdul perquè el reconegui. Per evitar problemes i guanyar temps s'ha desestimat.

Base de Dades

En aquest projecte s'ha fet servir **SQLite** per la seva senzillesa, a més Django ve configurat per defecte amb aquesta base de dades i no s'ha d'instal·lar ni configurar res al respecte.

Alternativa de Base de Dades

MySQL: Gestor de base de dades molt popular gràcies a *PHP*, compta amb molta documentació i és ideal per projecte petits i mitjans. Finalment s'ha desestimat perquè el projecte té un model de dades força simple i amb un gestor com SQLite en era suficient. Tot i així, no seria difícil passar-ho a *Mysql*, i seria la solució ideal si el projecte acaba creixent molt.

Editors

Per a programar la aplicació s'ha fet servir **IntelliJ Idea**, un editor de pagament amb el que comptem amb llicencia legal, i que incorpora infinitat d'utilitats. Entre elles destaquem una consola de comandes integrada i un connector de base de dades que ens permet gestionar les taules i fer consultes, tot això sense haver de sortir del programa.

Editors alternatius

Sublime Text 3: Un editor gratuït molt bo que compte amb molts *plugins* que faciliten la edició. Tot i així, no és tant complet com l'editor escollit.

Sistema Operatiu

S'ha fet servir **Linux** per a fer el projecte, concretament la distribució **Fedora**. És gratuït, molt estable i amigable amb Python.

Alternativa al sistema operatiu

Windows: Sense dubte el sistema operatiu més popular avui dia. És necessari llicencia i no és tan amigable amb *python*, necessita més configuració i instal·lació de software extra per poder fer el mateix que amb *linux*. S'ha desestimat per comoditat i per seguretat.

3. Anàlisi

A continuació s'exposa l'anàlisi de l'aplicació, tant els aspectes funcionals com els no funcionals, tot el que s'ha tingut en compte abans de la implementació del projecte.

3.1 Anàlisi funcional

L'anàlisi funciona comprèn tot el que s'ha de tenir en compte perquè l'aplicació funcioni de forma òptima.

Requeriments tècnics

Es tracta d'un seguit d'especificacions tècniques que ha de complir l'equip sobre el que anirà muntada l'aplicació.

Servidor físic
La màquina que fa servir de servidor ha de tenir unes especificacions mínimes per poder fer anar el sistema operatiu, el programes necessaris i la pròpia aplicació de forma òptima.

Requeriments mínims:

Component	Requeriment
Sistema Operatiu	Linux (Fedora, Ubuntu, CentOS)
Processador	Intel Pentium® D 2.8 GHz o AMD Athlon™ 64 X2 4400+
Ram	2GB de RAM
Disc dur (capacitat)	20GB d'espai lliure
Disc dur (velocitat)	5.400 rpm
Connexió a Internet	Ample de banda de 10mb

Taula 1. Requeriments mínims

Requeriments recomanats:

Component	Requeriment
Sistema Operatiu	Linux (Fedora, Ubuntu, CentOS)
Processador	Intel® Core 2 Duo 2.4 GHz o AMD Athlon™ 64 X2 5600+ 2.8 GHz
Ram	4GB de RAM
Disc dur (capacitat)	50GB d'espai lliure
Disc dur (velocitat)	7.200 rpm (o SSD)
Connexió a Internet	Ample de banda de 100mb

Taula 2. Requeriments recomanats

Servidor en el núvol

Una bona alternativa al servidor físic seria contractar un servidor en el núvol, ja que ens permet disposar d'un servidor amb les especificacions que contractem de forma remota. Les especificacions són les mateixes que per un servidor físic.

Requeriments funcionals

Es tracta de tot els software necessari perquè l'aplicació funcioni correctament.

Sistema Operatiu
Per al sistema operatiu sobre el qual correrà la aplicació em optat per Linux per ser potent, per la seva bona gestió de la memòria i per ser totalment gratuït. Aquest projecte s'ha desenvolupat íntegrament sobre *Fedora*, però no hi hauria problema en muntar-ho sobre qualsevol altre *Linux*.

Python
És el llenguatge en que està escrita l'aplicació sencera, per tant necessitem que la versió sigui compatible amb *Django*. En principi es suportaria la versió 2.7 de *Python*, però s'ha decidit fer servir la versió 3 per ser més actual i estar recomanada per *Django*.

Django
La versió de Django que s'ha fet servir és més nova en el moment d'escriure la memòria, 1.8.2.

Servidor y base de dades
Tant pel servidor com per la base de dades es fa servir els que venen per defecte amb la instal·lació completa de *Django*, per tant el servidor és el propi de la versió 1.8.2 de *Django* i la base de dades és *SQLite* 3.8.10.2.

Usuaris del sistema

Hi ha tres tipus d'usuaris en l'aplicació:

Administrador
És l'únic usuari del sistema que té accés al panell d'administració, on pot crear, modificar o eliminar dades, resultats o usuaris del sistema.

Usuari del sistema
Es tracta d'un usuari donat d'alta per l'administrador. Una vegada autenticat en l'aplicació, pot fer-la servir sense limitació.

Usuari no registrat
Aquest usuari no podrà passar de la pantalla d'autenticació. No podrà fer res fins que un administrador no li creï un usuari dintre del sistema.

Especificació de casos d'us

Tot seguit es mostra el diagrama de casos d'ús que s'ha seguit per crear l'aplicació, i es detalla cada cas.

Diagrama

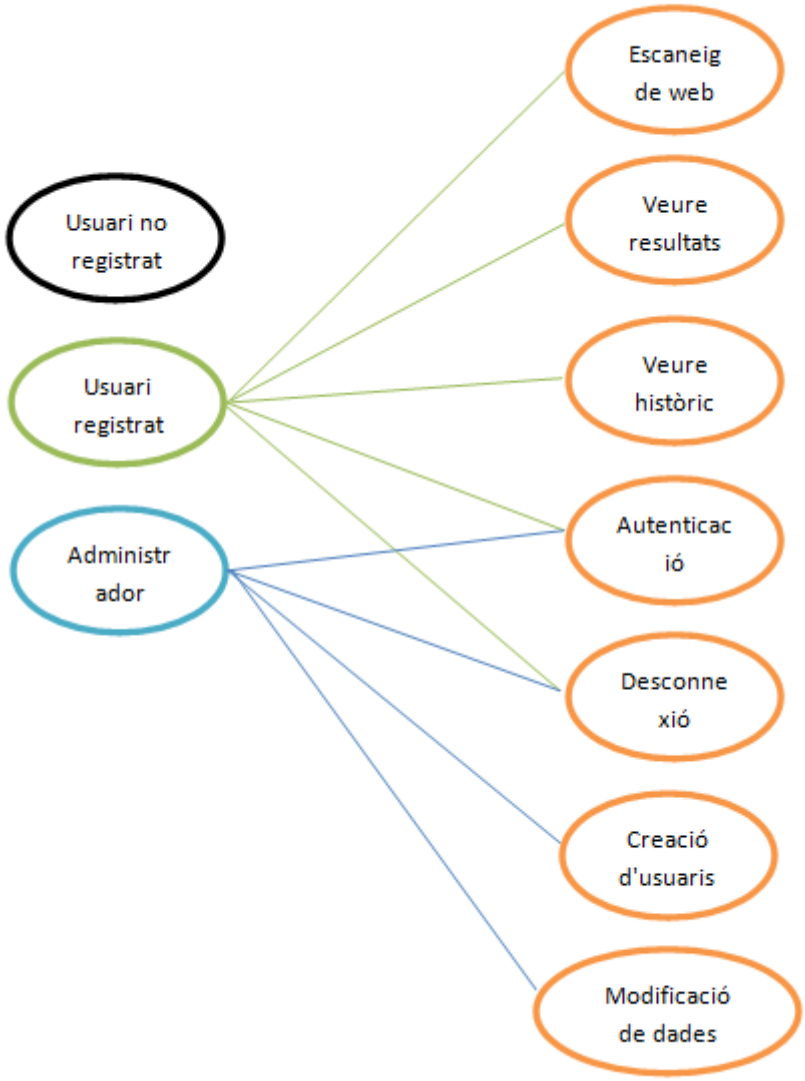


Figura 3. Diagrama de casos d'ús

Actors

Usuari no registrat: Qualsevol persona que no estigui registrat al sistema no podrà fer res més que veure la pantalla d'autenticació.

Usuari registrat: Es tracta de l'usuari principal del projecte i el que explotarà tota la capacitat del programa.

Administrador: És l'únic autoritzat per accedir a l'àrea d'administració.

Casos d'ús

Escaneig web: És l'acció d'omplir el formulari principal amb les dades de la web, els paràmetres opcionals i els tests desitjats i executar les proves.

Veure resultats: Visualització de resultats de cada test passat.

Veure històric: Acció de veure els resultats per pantalla amb el número de vulnerabilitats i tests passats.

Autenticació: L'acció de autenticar-se en el sistema.

Desconnexió: Es refereix a tancar la sessió. Per fer-ho has haver-te autenticat abans.

Creació d'usuaris: Afegir nous usuaris al sistema. Al ser una aplicació potencialment perillosa en males mans, no es permet el registre lliure d'usuaris.

Modificació de dades: Modificació, creació o eliminació de qualsevol dada del sistema.

3.2 Anàlisi no funcional

A continuació s'exposen els aspectes no funcionals de l'aplicació, la compatibilitat, la llei de protecció de dades i la accessibilitat.

Compatibilitat amb els navegadors

Perquè l'aplicació sigui compatible amb qualsevol navegador de qualsevol sistema operatiu, s'ha optat per maquetar la web fent servir **Bootstrap**. *Bootstrap* és un *framework* per al disseny de pàgines web, proporciona un seguit d'eines, plantilles, tipografies i efectes que són compatibles amb qualsevol navegador, i a més, facilita molt la tasca de maquetació i disseny de l'aplicació web.

Llei de protecció de dades (LOPD)

Aquesta llei té com a objectiu garantir i protegir, en lo referent al tractament de les dades personals, les llibertats públiques y els drets fonamentals de les persones. Per aquesta raó s'han pres mesures en la aplicació per prevenir la subtracció indesitjada de dades alienes.

En aquesta aplicació no es fa servir un volum de dades personal molt gran, tant sols l'usuari i contrasenya, de totes maneres s'han pres mesures per complir aquesta llei.

Contrasenya: La clau d'accés dels usuaris es guarda xifrada a la base de dades, per tant ningú que tingui accés a la base de dades podrà saber mai la contrasenya de l'usuari. L'algorisme de xifratge que fa servir Django per defecte és el *PBKDF2*.

Base de dades: La pròpia base de dades també està protegida sota usuari i contrasenya, evitant l'accés directe de tercers a les dades dels usuaris i resultats dels test.

Accessibilitat

S'han pres mesures perquè l'aplicació tingui un disseny atractiu a l'hora de que sigui llegible e intuïtiva.

Font
S'ha fet servir les fonts i dissenys per defecte de Bootstrap, ja que estan pensades perquè sigui atractiva i llegible. D'aquesta manera facilitem que un usuari amb dificultats visuals pugui fer servir la aplicació sense cap tipus d'augment visual.

Colors

Per facilitar la visualització de les lletres s'ha triat uns colors clars pels fons dels texts i un ressaltat pels títols, juntament amb un icona perquè sigui més intuïtiu.

4. Disseny

Per disseny ens referim tant en l'aspecte visual i estructura de les pàgines com al disseny del model de dades de l'aplicació i la seva estructura de codi.

4.1 Estructura de carpetes

Per estructurar el codi s'ha seguit les estàndard de *Django* per evitar problemes de configuració. La estructura queda de la següent manera:

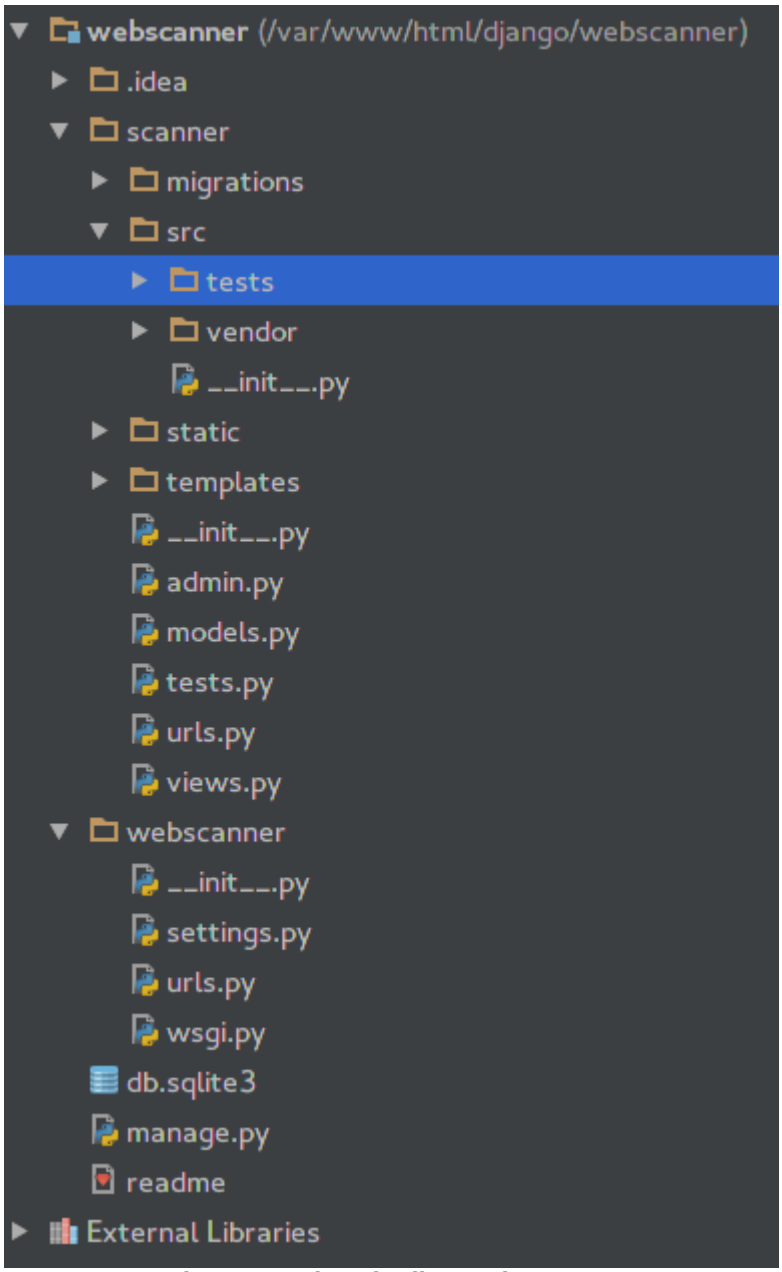


Figura 4. Arbre de directoris

Per **Django** existeix primer el projecte (en el nostre cas *webscanner*) i les aplicacions (*scanner* en aquest cas) que es poden crear tantes com es vulgui dintre d'un mateix projecte.

Els fitxers de configuració de la aplicació estan dintre de la carpeta *webscanner*, però ens centrarem en les carpetes dintre de la aplicació scanner, que és on està tot el codi del projecte.

Migrations
En aquesta carpeta es guarden les migracions de les bases de dades amb tots els canvis que s'han fet. Quan es vol actualitzar la base de dades, s'actualitza el fitxer *models.py* i mitjançant la consola de comandes, s'executa una comanda que genera una nova migració i actualitza la base de dades. Aquesta migració és la que es guarda en aquesta carpeta.

src
Aquí és on guardem tot el codi de la aplicació referent als tests de vulnerabilitats que passem. En la carpeta "*tests*" hi ha un fitxer per a cada prova que seguretat que hi ha al sistema. En la carpeta "*vendor*" és on està el codi dels programes de tercers que fem servir en alguns tests.

static
En aquesta carpeta, Django guarda tot el contingut estàtic de la aplicació, es refereix al codi *javascript*, imatges o fulles d'estils.

templates
Carpeta amb tots els fitxers *HTML* de l'aplicació. *Django* fa servir un sistema de plantilles propi que permet l'herència, inclusió i pas de paràmetres a cada plantilla, per tant, les plantilles estan escrites amb la finalitat de treure profit a aquestes característiques.

Fitxers generals

- **admin.py**: Es el fitxer de configuració de l'administrador. Quan es creen nous models, s'ha de modificar aquest fitxer perquè apareguin. També és possible canviar l'aparença de l'administrador.

- **models.py:** Fitxer on es defineix el model de dades explicat en aquest mateix capítol.

- **urls.py:** En aquest fitxer es controla les peticions que es reben de la url i, segons el que es rebi, res s'encaminarà a una vista o altre.

- **views.py:** Es defineixen les vistes de l'aplicació. Aquest fitxer és el que acaba reben la informació del formulari de la web a escanejar, i executa el codi de dintre de la carpeta *src* segons necessiti. Cada vista acaba retornant una plantilla junt a les dades a mostrar.

4.2 Estructura de la web

L'estructura de la web és força simple, ja que s'ha intentat que fos intuïtiu. La complexitat en aquesta part no pren tanta importància com la que tenen la programació de tests.

Autenticació

Aquesta es la pàgina inicial de l'aplicació. Si s'intenta entrar en qualsevol altre secció de la web sense estar autenticat l'aplicació redirigirà a aquesta pàgina.

Figura 5. Pantalla d'autenticació

Pàgina d'escaneig

Aquesta és la pàgina principal, i és la mateixa pàgina que quan fem clic en "*nou escaneig*". Consta d'un text explicatiu i un formulari per omplir les dades de la web a escanejar i els tests que volem aplicar.

Figura 6. Pantalla principal

Pàgina de resultats

Quan escanegem una pàgina, arriben a la següent pantalla on es mostren els resultats agrupats per cada test triat. Per a cadascun tenim totes les vulnerabilitats trobades, o un missatge confirmant que no s'han trobat vulnerabilitats.

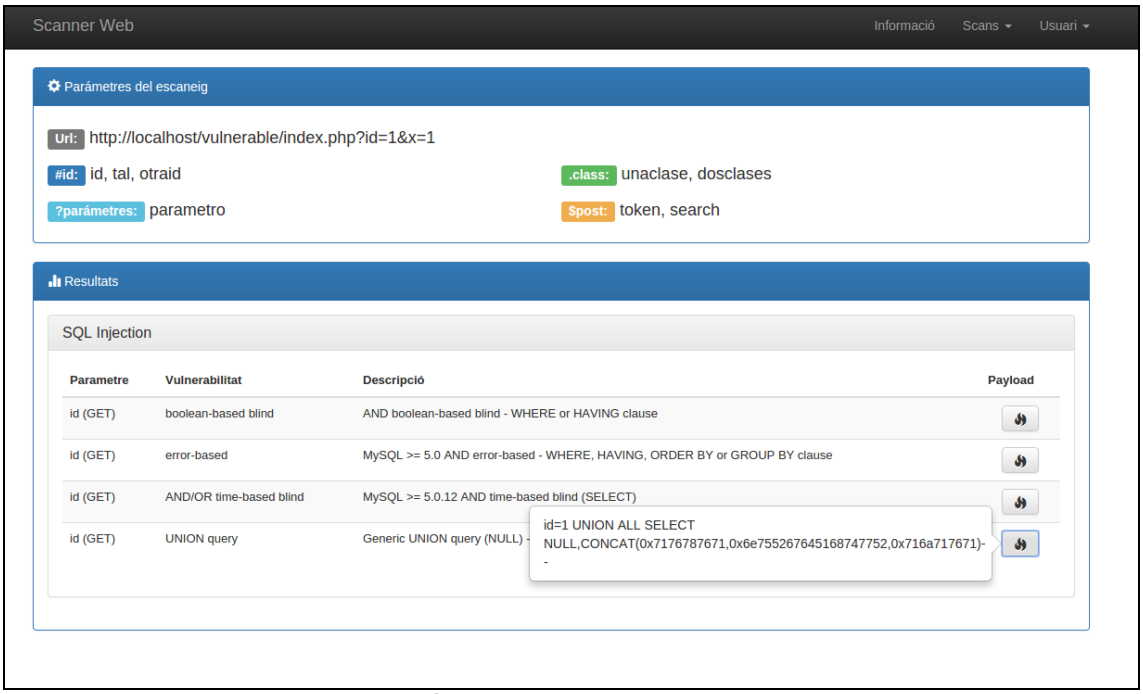
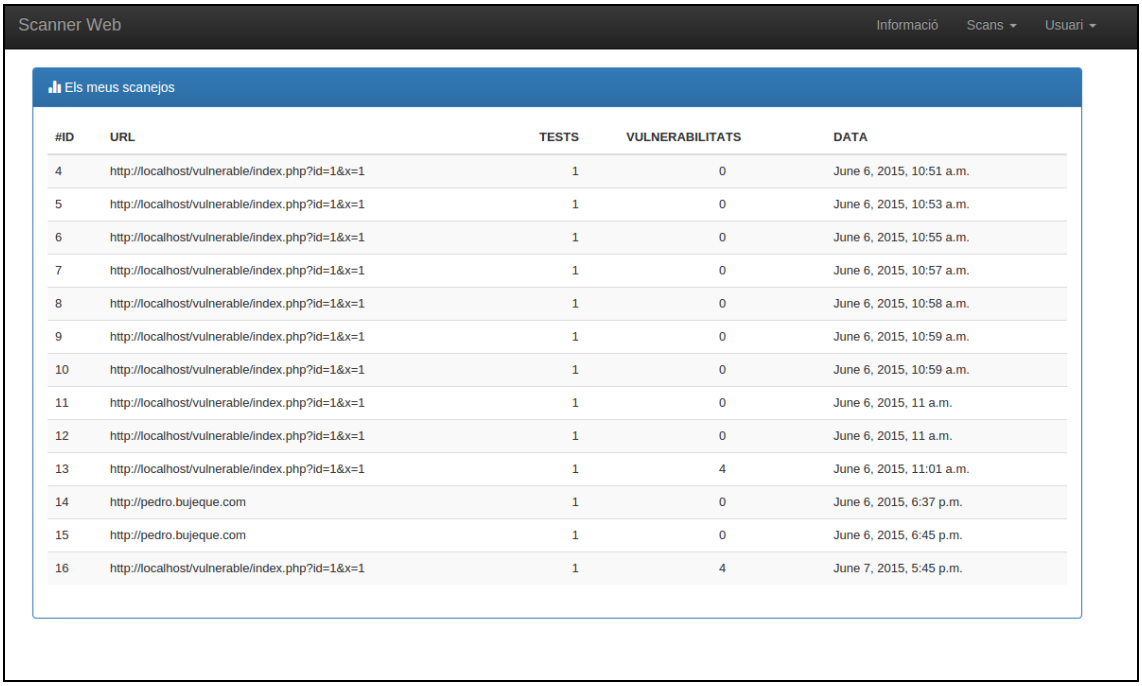


Figura 7. Pàgina de resultats

Els meus escanejos

Aquesta pàgina mostra els resultats dels escanejos enregistrats amb l'usuari autenticat. Podem veure quants test s'han passat i el total de vulnerabilitats trobades.



#ID	URL	TESTS	VULNERABILITATS	DATA
4	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:51 a.m.
5	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:53 a.m.
6	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:55 a.m.
7	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:57 a.m.
8	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:58 a.m.
9	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:59 a.m.
10	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 10:59 a.m.
11	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 11 a.m.
12	http://localhost/vulnerable/index.php?id=1&x=1	1	0	June 6, 2015, 11 a.m.
13	http://localhost/vulnerable/index.php?id=1&x=1	1	4	June 6, 2015, 11:01 a.m.
14	http://pedro.bujeque.com	1	0	June 6, 2015, 6:37 p.m.
15	http://pedro.bujeque.com	1	0	June 6, 2015, 6:45 p.m.
16	http://localhost/vulnerable/index.php?id=1&x=1	1	4	June 7, 2015, 5:45 p.m.

Figura 8. Pantalla dels meus escanejos

Al fer clic sobre qualsevol test, arribarem a la mateixa pàgina de resultats de com si haguéssim acabat d'executar el test, però en aquest cas recuperem les dades de la base de dades.

Administració

Al panell d'administració s'accedeix des d'una direcció diferent que la resta de la aplicació, ja que es tracta del panell d'administració propi de *Django* i que crea per defecte. Per entrar ens hem d'identificar a través del següent formulari:

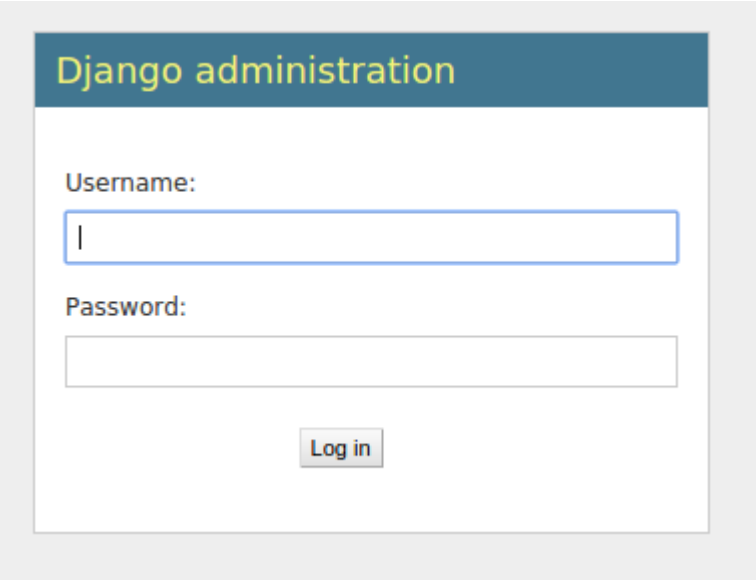


Figura 9. Pantalla d'autenticació de l'administrador

Una vegada autenticat, podem veure la informació de totes les taules de la base de dades, y a més, les pròpies de *Django* sobre usuaris.

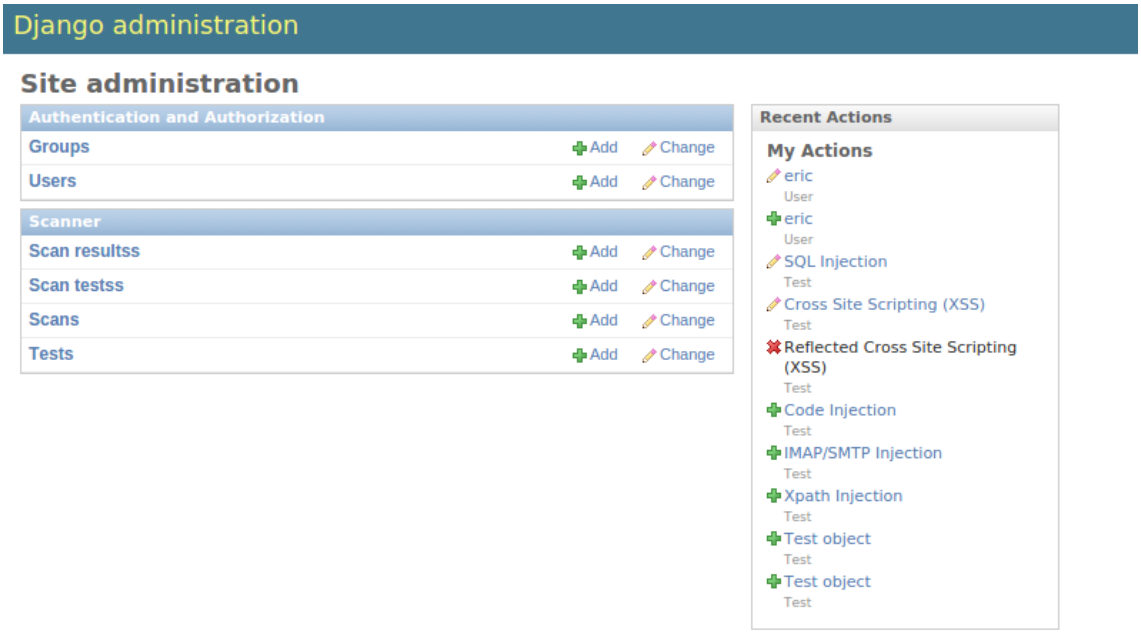


Figura 10. Pantalla de l'administrador

Si entrem a usuaris, per exemple, podem modificar-los, esborrar-los o crear-ne de nous.

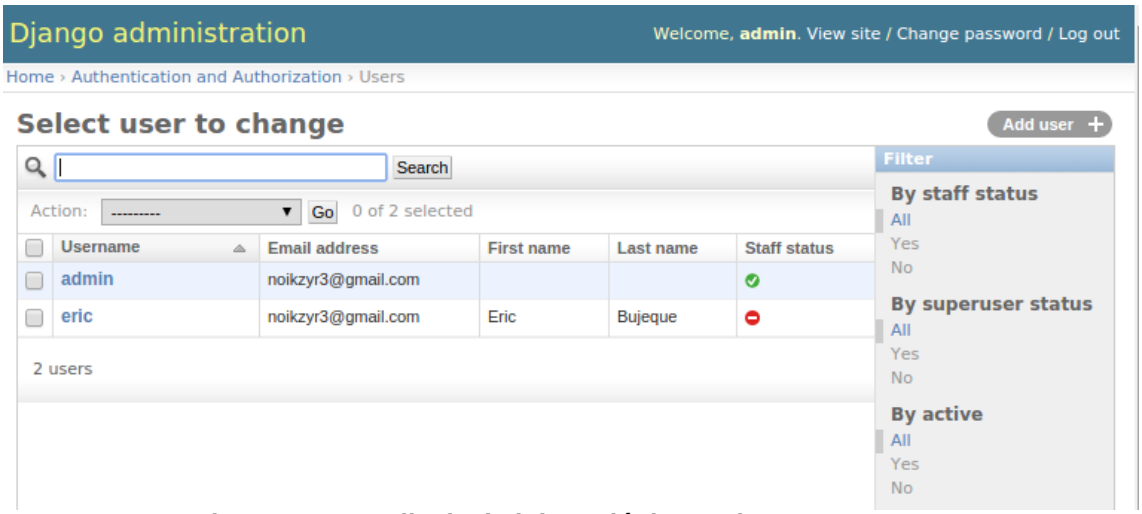


Figura 11. Pantalla d'administració d'usuaris

4.3 Model de dades

A continuació es mostra el model generat, el diagrama de la base de dades i la corresponent explicació de cada taula.

Model

Quant em parlat de l'estructura de fitxers, em vist que l'arxiu *models.py* contenia tota la informació dels models, que finalment són les taules de la base de dades. *Django* crea una taula d'usuaris i grups per defecte, així que podem obviar aquestes taules i afegir només el que necessitem. A més, per qualsevol classe o taula que creem, *Django* ens crea un camp *id* per defecte, per tant no ens cal posar-ho. El fitxer ens queda de la següent manera:

```
class Test(models.Model):
    test = models.CharField(max_length=50)
    clase = models.CharField(max_length=50, default='common')
    pub_date = models.DateTimeField('date published')

    def __unicode__(self):
        return self.test


class Scan(models.Model):
    user = models.ForeignKey(User, default=0)
    url = models.CharField(max_length=150)
    posts = models.CharField(max_length=150, null=True)
    ids = models.CharField(max_length=150, null=True)
    clases = models.CharField(max_length=150, null=True)
    params = models.CharField(max_length=150, null=True)
    pub_date = models.DateTimeField('date published')


class ScanTests(models.Model):
    idscan = models.ForeignKey(Scan)
    test = models.ForeignKey(Test)


class ScanResults(models.Model):
    idscanTest = models.ForeignKey(ScanTests)
    param = models.CharField(max_length=150, null=True)
    vulnerability = models.CharField(max_length=150, null=True)
    description = models.CharField(max_length=150, null=True)
    payload = models.CharField(max_length=150, null=True)
    pub_date = models.DateTimeField('date published')
```

Figura 12. Fitxer models.py

Test
Aquesta taula conté la informació sobre els tests, que serà el que es mostrarà a la llista del formulari per escanejar una web.

Scan
Guarda tota la informació referent a la petició d'escaneig realitzat per l'usuari, és a dir, tot el que l'usuari ha demanat escanejar. Fa servir l'usuari com a clau forana.

ScanTests
Aquesta taula és la que relaciona cada escaneig de la taula "*Scan*" amb els tests demanats per l'usuari. Conté les claus foranes de la taula "*Scan*" i "*Test*" per relacionar-les.

ScanResults
Aquí es guarda cadascun dels resultats dels tests de vulnerabilitats obtinguts. Tenim l'identificador de *ScanTests* com a clau forana, ja que cada resultat estarà relacionat amb un Test.

Users
Taula d'usuaris pròpia de *Django*, on es guarda la contrasenya i se li dona un identificador a cada usuari. A més, també guarda informació opcional com el correu electrònic, nom y cognoms. Així es mostra la taula en la base de dades:

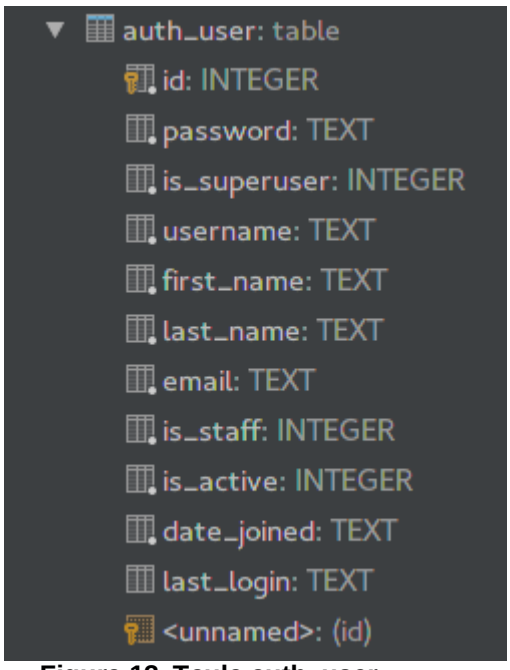


Figura 13. Taula auth_user

Diagrama

El diagrama explica clarament la relació entre les taules, s'inclou també la taula usuaris, que encara que no aparegui en el model existeix i és vital pel funcionament de la resta.

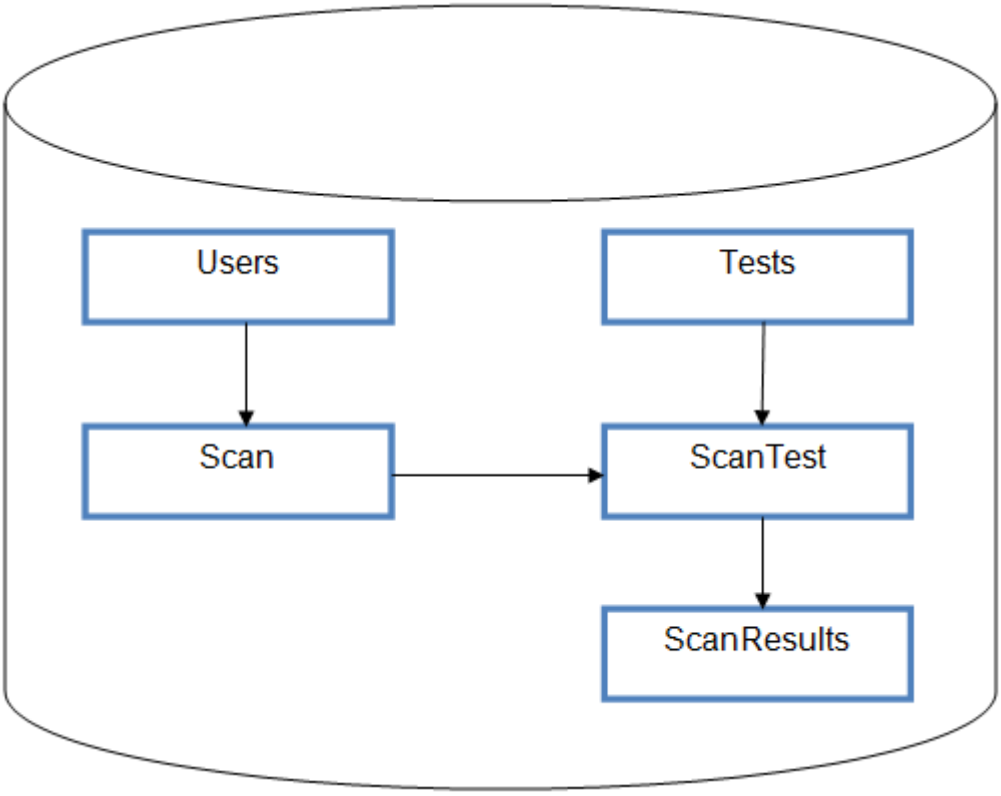


Figura 14. Diagrama model de dades

5. Implementació

En l'apartat d'implementació s'exposa la instal·lació, configuració i funcionament general de la aplicació.

5.1 Instal·lació

El primer de tot serà instal·lar l'eina *pip* per facilitar les tasques d'instal·lació, si estem en *Fedora* obrim una consola de comandes i escrivim:

```
sudo yum install pip
```

Si és *Ubuntu*:

```
sudo apt-get install pip
```

El següent que necessitem és instal·lar un entorn virtual per python i la nostre aplicació. Aquí s'instal·laran tots els components que necessitarem per la nostre aplicació, com per exemple la llibreria Requests, encarregada de fer peticions http i recollir el resultat. L'instal·larem amb la següent línea de comandes:

```
sudo pip install virtualenv
```

Una vegada instal·lat l'hem de crear amb la següent línea:

```
virtualenv webscanner
```

Un cop tenim el servidor on anirà instal·lada l'aplicació complint amb els requeriments exposats en els capítols anteriors referents als requeriments funcionals i no funcionals, haurem d'instal·lar l'aplicació començant per activar l'entorn virtual de python:

```
source venv/bin/activate
```

L'aplicació s'entrega en un paquet *webscanner-project-1.0.tar.gz* que, si tenim preparat l'entorn i hem instal·lat l'eina *pip*, només haurem d'executar la línea de comandes següent:

```
pip install --user webscanner-project-1.0.tar.gz
```

I ens muntarà tota l'estructura del projecte amb la configuració per defecte.

Per arrancar l'aplicació haurem d'estar dintre de la carpeta del projecte i escriure la següent línea (s'arrancarà en el port 8000 per defecte):

```
python manage.py runserver
```

Per finalitzar, instal·larem les dependencies de paquets que necessita el nostre projecte per funcionar. Anem a l'arrel del projecte i escrivim la següent comanda:

```
pip install -r requirements.txt
```

5.2 Configuració

La base de dades **SQLite** és un fitxer *SQLite3.db* que ja està inclòs dintre del directori. En l'arxiu *settings.py* ja està configurat perquè l'aplicació faci servir aquest arxiu com a base de dades, per tant no hauríem de tocar res més al respecte.

Si arriba un dia en que l'aplicació creix i es vol canviar de sistema gestor de base de dades, o simplement canviem la contrasenya del nostre **SQLite**, hauríem d'obrir el fitxer *settings.py* del projecte i canviar la connexió, usuari o contrasenya allà.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Figura 15. Fragment del fitxer settings.py

5.3 Funcionament de la aplicació

En l'apartat de disseny s'ha explicat les diferents pàgines que té l'aplicació. El que s'explica en aquest capítol és com funciona l'aplicació a nivell més tècnic.

Django està considerat un *framerwork* que fa servir *MVC* (model - vista - controlador) com a patró d'arquitectura de software. Tot i que ells mateixes afirmen que és més aviat un *framework MTV* (model - *template* - vista), ja que considera que l'aplicació en sí fa de controlador.

A continuació s'explicarà breument com funciona el projecte i com s'ha implementat el codi propi dintre de *Django*.

Ja em vist en l'apartat de disseny l'estructura de carpetes que té el projecte i els arxius més importants. Comencem doncs des de que s'efectua una petició al navegador web.

urls.py

Aquest arxiu recull la direcció enviada pel navegador i la tracta segons unes normes escrites amb expressions regulars. Per ejemple, si escrivim:

http://localhost:8000/webscanner/scanner/**scan/**

La informació que li arriba a *urls.py* serà "*scan/*", el qual s'ha decidit que vagi a la vista *scan*. Si en canvi fem la següent petició:

http://localhost:8000/webscanner/scanner/**scan/10/**

La informació que arriba en aquest cas es "scan/10/" i la vista que es crida es diferent, ja que no correspon amb el mateix patró que l'anterior petició.

```
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^scan/$', views.scan, name='scan'),
    url(r'^scan/(?P<scan_id>[0-9]+)/$', views.scan_detail, name='scan_detail'),
    url(r'^login/$', views.login_vista, name='login'),
    url(r'^login_action/$', views.login_action, name='login_action'),
    url(r'^logout/$', views.logout_action, name='logout'),
    url(r'^myscans/$', views.myscans, name='myscans'),
    url(r'^info/$', views.info, name='info'),
```

Figura 16. Fragment del fitxer urls.py

views.py

Aquests és el fitxer de vistes, l'anterior fitxer redirigeix la petició a una vista o una altra, definides en aquest fitxer, segons l'expressió regular que capti la direcció.

En aquest fitxer s'importen tots els models i les classes de tots els tests que passarem. El funcionament d'una vista és bàsicament el següent:

- **Recollir paràmetres**: Només en les vistes que reben paràmetres d'un formulari o paràmetres en la *url*.

- **Consulta a la BBDD**: Per exemple, per llistar tots els tests en el formulari principal de l'aplicació, es fa una consulta per extreure'ls de la base de dades.

- **Crida als tests**: Aquesta acció només la efectua la vista "*scan*". Al recollir paràmetres, revisa quins són els tests sol·licitats i els crida en conseqüència, passant-li la resta de paràmetres recollits.

- **Retornar dades**: Tant amb la consulta a la BBDD com a les crides al tests, es necessari generar una estructura adequada amb la informació obtinguda per tal de que es pugui mostrar per pantalla. Això serà així sempre i quan la vista que estem cridant ho necessiti, per exemple, la pàgina d'autenticació no necessita cap informació de test o de la base de dades, però sí que ho necessita la vista que verifica si l'usuari o contrasenya introduïts corresponen o no.

- **Retornar una plantilla o redirigir:** Aquest últim pas és el que les vistes faran de manera obligatòria en aquest projecte. Si es crida la vista de *login* es retornarà la plantilla login.html. Si es crida a la vista que verifica l'autenticació, redirigirà a la vista del formulari principal de l'aplicació si la informació és correcte, o redirigirà al la vista de *login* altre vegada en cas d'introduir una informació errònia.

Un exemple de vista de verificació d'autenticació:

```
def login_vista(request):
    return render(request, 'scanner/login.html')


def login_action(request):
    user = request.POST['user']
    passwd = request.POST['password']
    user = authenticate(username=user, password=passwd)
    if user is not None and user.is_active:
        login(request, user)
        return redirect('index')
    else:
        return redirect('login')
```

Figura 17. Vistes d'autenticació

El funcionament de l'aplicació, des del punt de vista de flux de dades, ve a ser el següent:

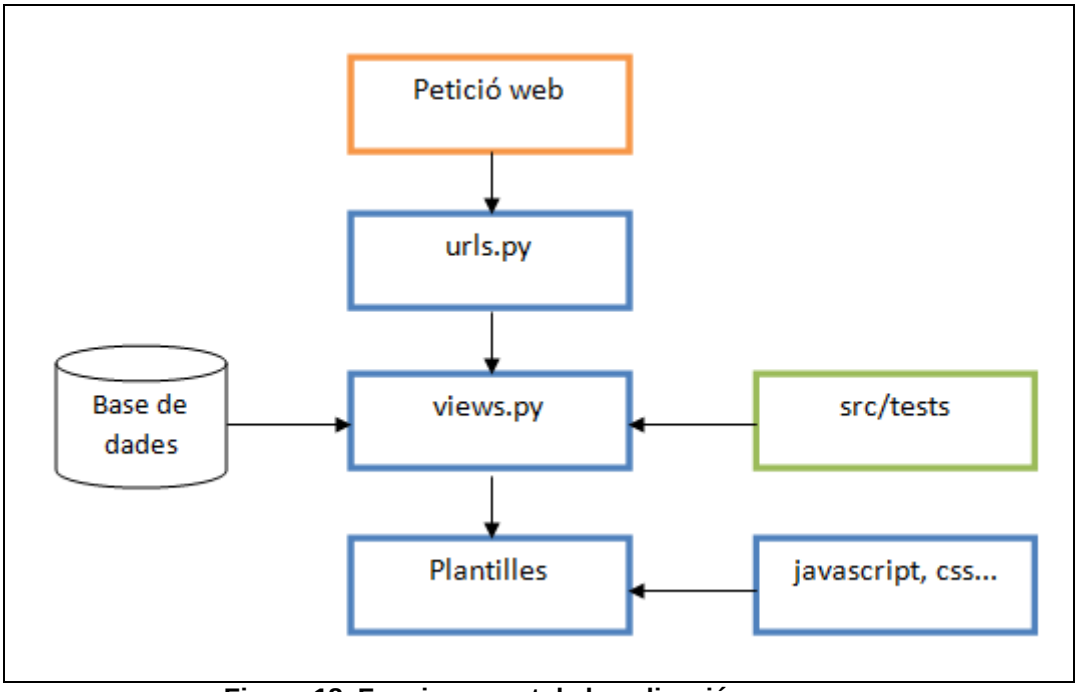


Figura 18. Funcionament de la aplicació

5.4 Tests

Tot el codi dels tests està dintre de la carpeta *src* del projecte. El que fa cadascun és executar un codi per testejar la web introduïda i retornar el resultat a les vistes, concretament a la vista "*scan*". El resultat dels codis és la mateixa per tots, per facilitar el pas de paràmetres a les plantilles.

```
{
        'parámetre': [{
                'type': 'Tipus',
                'payload': 'payload',
                'title': 'Descripció'
        }]
}
```

En termes de *Python*, generem una un diccionari on la clau es el paràmetre, i per a cada paràmetre hi ha una llista de vulnerabilitats trobades. El format està basat en els resultats que dona *SQLmap*, ja que va ser el primer test a fer.

SQL Injection

Per la complexitat d'aquest test s'ha optat per fer servir una eina externa com ***SQLmap***. Es tracta d'una eina molt eficaç d'injecció de codi *SQL* que s'ha utilitzat a l'assignatura de bases de dades. A més, al estar escrita en *python*, fa que els requeriments siguin els mateixos.

Per executar aquest test el que fem és recollir els paràmetres web i executem la següent comanda per consola, a través de la funció de *python* ***subprocess***:

```
sqlmap.py -u 'direccio_web' --batch
```

-u 'url': Especifiquem la web que volem escanejar, si la direcció conté paràmetres com ?id=1 o similar, els agafarà automàticament a menys que especifiquem quins paràmetres volem testejar.

--batch: Execució en mode Batch, és a dir que no pregunti res, simplement s'executarà i donarà el resultat.

Una vegada tenim el resultat, el passem per un "*parser*" dintre de la mateixa classe, i generem l'estructura comú de resultats de tots els tests, per enviar-li a la vista.

XSS

Per fer proves de *Cross site scripting* (*XSS*) hi ha eines molt bones de codi obert disponibles, però per la dificultat d'integració o pels alts

requeriments de llibreries o altre software, s'ha optat per fer el nostre propi test per *XSS* només amb la llibreria ***Requests*** de *Python*.

Hi ha dos tipus generals de vulnerabilitats XSS, la que es guarda en l'aplicació (*stored*) i la que només es mostra per pantalla (*reflected*).

Reflected XSS
Normalment aquesta vulnerabilitat ve donada per no validar correctament els paràmetres de la direcció rebuts, per exemple una direcció amb un paràmetre index.php?**page=2**, el programador genera els enllaços de les pàgines amb números sense pensar que un usuari maliciós pot canviar el número per un codi *XSS*. La prova més fàcil es canviar aquest número per un codi *javascript* com per exemple:

```
<script>alert('xss');</script>
```

Aquest codi mostrarà una alerta per pantalla. Imaginem que la aplicació mostra la pàgina actual a través del paràmetre enviat ("Vostè està a la pàgina 2"). Si substituïm el número de la *url* pel codi *javascript*:

```
index.php?page=<script>alert('xss');</script>
```

Si la aplicació no comprova que el paràmetre és un número, i ho executa sense filtrar-ho, hauríem d'obtenir una alerta amb el text "*xss*", i significaria que no ha passat el test i seria vulnerable a aquest tipus d'atac *XSS*. Per passar el test, el que fem és crear una marca de temps i canviar-la en el valor del paràmetre, fer la petició web amb el *requests* amb el nou format, i buscar si realment apareix la marca de temps en el codi junt al codi *javascript*.

Stored XSS
Aquesta vulnerabilitat és típica de formularis. Per exemple, en una web on es pot omplir un perfil amb les dades perquè la gent ho vegi, si el formulari és vulnerable, podríem escriure el mateix codi que hem fet servir en el *Reflected XSS*, però afegint alguna dada per no deixar el camp visiblement vuit. Per exemple, en el camp "aficions":

```
Motociclisme i arts marcials.<script>alert('xss');</script>
```

Si el programador de l'aplicació no filtra els camps introduïts contra etiquetes *javascript*.

Per provar-ho, es fa un "*submit*" del formulari amb codi XSS en els camps. Si quant es torna a fer la petició web apareix el codi maliciós, significa que és vulnerable.

HTTP Verb Tampering

Aquesta és una vulnerabilitat senzilla de testejar, segons les especificacions de OWASP els servidors webs accepten per defecte peticions de tipus GET i POST. Però també existeixen altres tipus de peticions que és possible que el servidor accepti (per un descuit, o per dotar una aplicació d'una funcionalitat extra), són les següents:

- OPTIONS
- HEAD
- PUT
- DELETE
- TRACE
- CONNECT
- PROPFIND
- PROPPATCH
- MKCOL
- COPY
- MOVE
- LOCK
- UNLOCK

Per testejar aquesta vulnerabilitat només hem de construir una capçalera HTTP amb cadascun dels tipus anteriors i mirar si la petició retorna un codi 200 o no. En cas de retornar un codi 200 significaria que és vulnerable, si obtenim un codi d'error 501 voldria dir que està capat.

Contaminació de paràmetres HTTP

Aquesta vulnerabilitat (en anglès *HTTP parameter pollution*) es basa en duplicar un paràmetre de la direcció de la web. Per provar-ho s'ha de seguir les següents passes:

- Fem una petició amb els paràmetres normals, per exemple: *index.php?param=valor1*
- Fem una segona petició amb un altre valor, per exemple: *index.php?param=valor2*
- Fem una tercera petició combinant els dos paràmetres: *index.php?param=valor1&param=valor2*

Si el resultat de la tercera petició és diferent a la primera o la segona petició, es considerarà vulnerable.

XML Injection

Aquest test l'executarem si la web en qüestió fa consultes sobre un fitxer XML. En aquesta ocasió, el test és més simple que un atac d'injecció

SQL perquè el format XML és força simple, i si aconseguim introduir un caràcter "<" o " ' " de més, el programa petarà al consultar de nou l'arxiu XML, ja que es trencaria la estructura del fitxer. Per exemple:

Tenim la estructura següent:

```
<usuaris>
	<usuari>
		<nom>Eric</nom>
		<edat>27</edat>
	</usuari>
</usuaris>
```

Si al introduir un nou usuari en comptes d'escriure 'Eric' escric:

```
Nom: Eric'
Edat: <27
```

El nou usuari generaria la estructura següent al fitxer XML:

```
<usuaris>
	<usuari>
		<nom>Eric'</nom>
		<edat><27</edat>
	</usuari>
</usuaris>
```

En notació XML les cometes només formen part dels atributs, i la clau "<" indica el començament d'una etiqueta, per tant el fitxer XML es tornaria inconsistent i donaria error al llegir-ho.

SSI Injection

Aquest test (*Server side includes*) es tracta d'incloure fitxers del propi servidor com a codi de l'aplicació. Per tant, és semblant al test XSS, però en aquest cas intentarem mostrar informació més enllà de la pròpia aplicació. Perquè aquest test sigui efectiu, el servidor ha de tenir habilitat el suport SSI, que substitueix un *string* per la informació demanada. Per exemple:

```
<!--#include virtual="/etc/passwd" -->
```

Aquesta línea retornaria el els usuaris i contrasenyes xifrades del servidor on està allotjada la aplicació. També és possible construir una capçalera HTTP amb aquesta informació i llegir la capçalera HTTP resultant.

Code Injection

Aquest test intenta executar comandes o pàgines del servidor web passant la informació per paràmetres de la *url*. Per exemple, un codi que agafa per paràmetre la web que obrirà:

```
index.php?p=users
```

El codi web farà una inclusió de *users.php*. Si canviem el paràmetre podem incloure un fitxer del servidor o una pàgina externa. Per exemple:

```
index.php?p=../../../../etc/passwd
```

El que es prova en el codi de l'aplicació és un arxiu remot que conté una alerta en *javascript*. Ens queda el següent:

```
index.php?p=http://eric.bujeque.com/alert.js
```

Si recollim la alerta de *javascript* voldrà dir que la web compleix aquesta vulnerabilitat.

6. Seguretat

El projecte ha d'estar protegit, tant les dades com el propi codi, per complir amb la LOPD i per impedir que la pròpia aplicació sigui utilitzada per finalitats que no ha estat pensada.

6.1 Base de dades

Tot i ser **SQLite**, un sistema de base de dades simple i lleuger, és possible protegir l'accés amb usuari i contrasenya, tot i que utilitza un xifrat "*Caesar cipher*", un sistema de xifrat força simple que corre les posicions dels caràcter X posicions. Per exemple, si tenim una A i està configurat perquè corri 2 posicions, el resultat serà una C. Pel projecte s'ha prescindit d'usuari i contrasenya.

Al ser un arxiu local, dintre del propi projecte, necessitem donar-li els permisos adients perquè només l'aplicació pugui accedir a ella.

A més, hem de vigilar que la pròpia aplicació no sigui vulnerable a atacs maliciosos que permetin apoderar-se de la base de dades en qüestió.

Les consultes a la base de dades es gestionen amb *Django*, que s'encarrega d'obrir i tancar les sessions, així com de filtrar els paràmetres que s'envien, per evitar atacs d'injecció SQL.

6.2 Control de sessions

Les sessions en *Django* es passen a les vistes a través de la variable *request* que reben per defecte. Aquesta variable també conté la informació dels paràmetres enviats per GET o POST.

Per a cada vista que necessiti estar autenticat (és a dir, totes menys la del propi formulari d'autenticació), hi ha posat un "*decorator*" de *Django* anomenat ***@login_required(),*** que dota a les vistes del codi encarregat de validar si l'usuari que està executant la vista està autenticat en el sistema o no a través de la variable *request*.

6.3 Xifratge de contrasenyes

El propi Django també s'encarrega de xifrar les contrasenyes dels usuaris donats d'alta a través del formulari del panell d'administració.

El sistema que fa servir és diu **PBKDF2** (*Password-Based Key Derivation Function 2*). Un algoritme molt segur elaborat pels creadors de l'estàndard de xifratge de clau pública RSA. Aquest sistema de xifratge aplica una funció *seudo* aleatòria de *hash* sobre la contrasenya, junt a un "*salt*", diverses vegades fins a obtenir el resultat final.

6.4 Protecció CSRF per formularis

El propi *Django* també incorpora una protecció contra CSRF. Aquesta protecció és obligatòria implementar-la, ja que si no s'aplica als formularis no els deixa enviar i retorna un error. El que s'ha d'enviar dintre dels formularis de les plantilles és el següent codi:

```
{% csrf_token %}
```

Només posant això Django ja s'encarrega de protegir el sistema d'aquesta amenaça.

7. Proves

A continuació s'exposen totes les proves realitzades en l'aplicació.

7.1 Proves d'autenticació

Per començar aquesta prova, sense haver-nos autenticat, intentem accedir a la següent direcció:

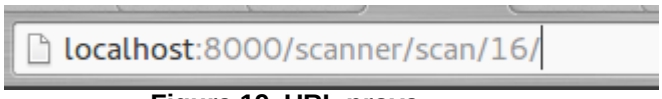


Figura 19. URL prova

Veiem que automàticament som redirigits a la pàgina d'autenticació:

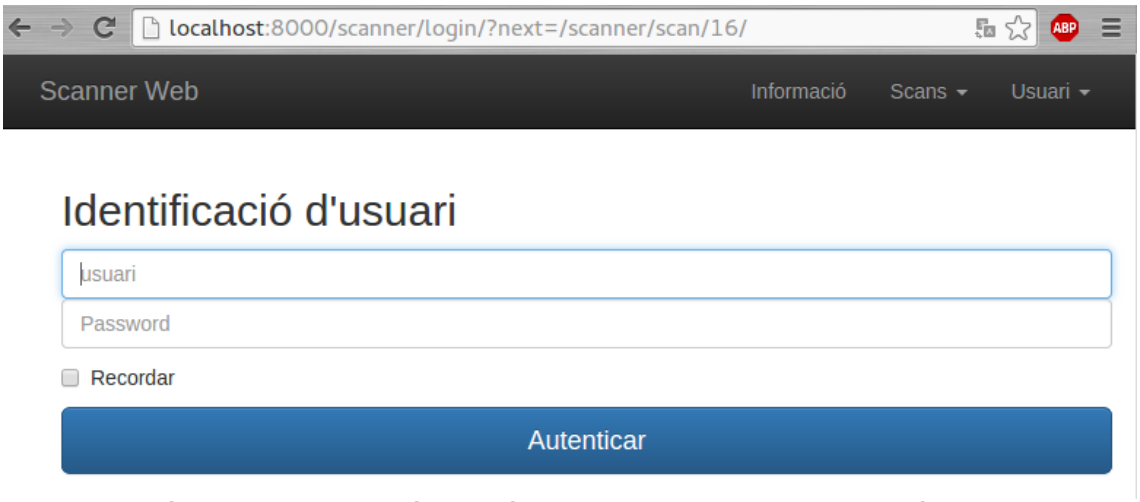


Figura 20. Pàgina d'autenticació després d'intentar accedir sense permís

A més, si intentem posar un usuari incorrecte ens surt el següent avís:

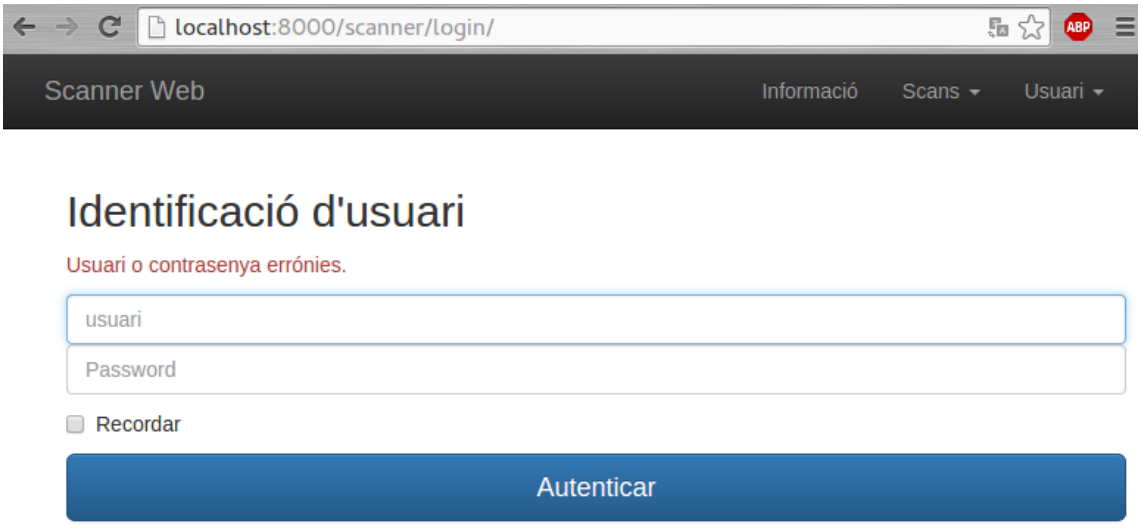


Figura 21. Autenticació fallida

En canvi quan el posem correctament ens apareix la pàgina d'inici amb el formulari d'escaneig:

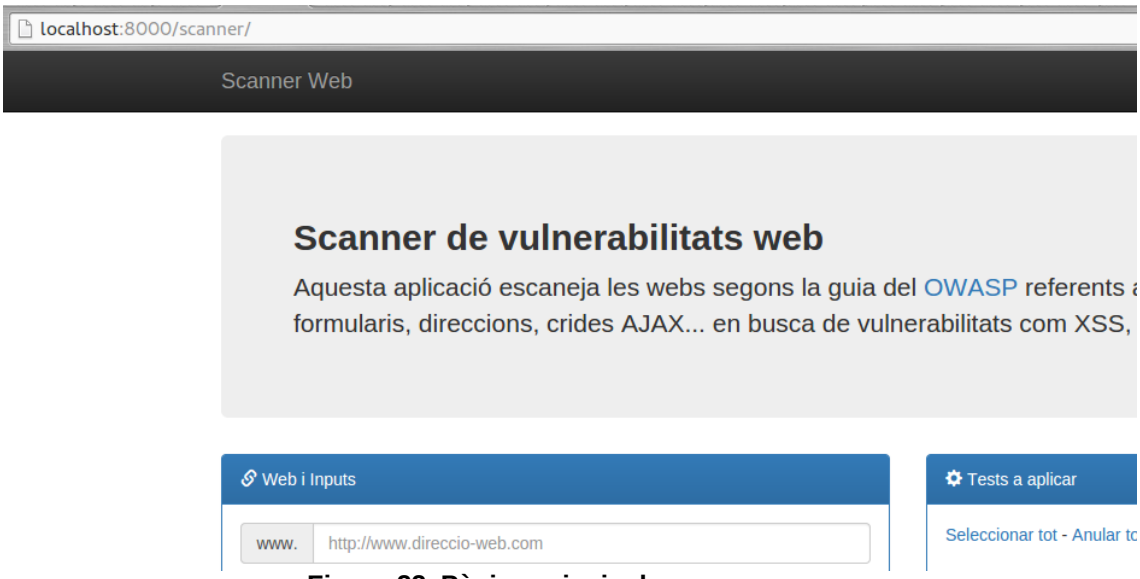


Figura 22. Pàgina principal

7.2 Prova dels tests

Per assegurar-nos de que els tests funcionen correctament, s'ha creat un servidor ***Apache*** per ***PHP*** amb base de dades ***MySQL*** i s'han generat pàgines web vulnerables. Després s'han passat els tests un a un.

SQL Injection

Omplim el formulari amb la pàgina vulnerable a *SQL Injection*:

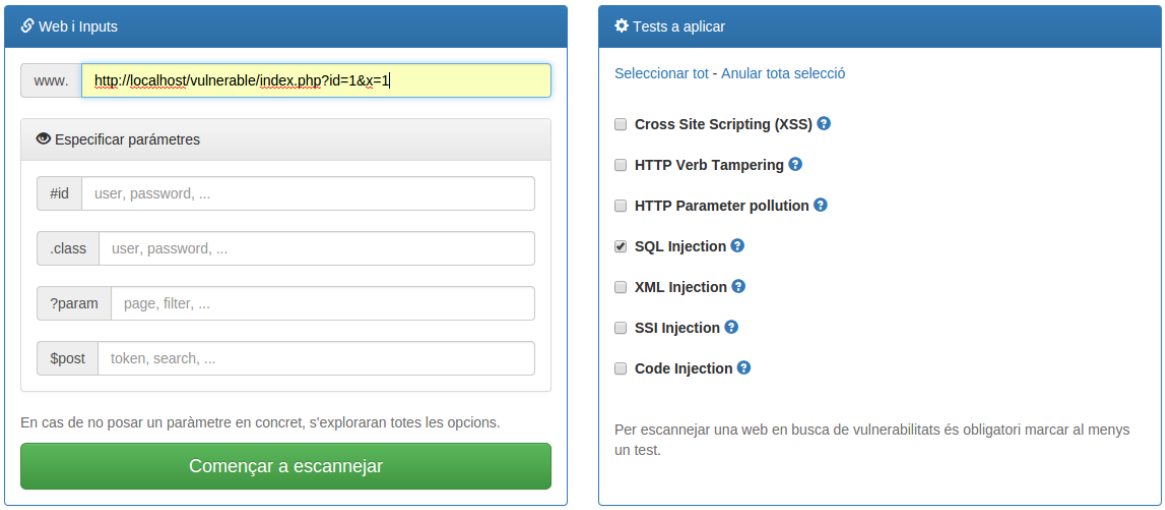


Figura 23. Formulari SQL Injection

El resultat és el següent:

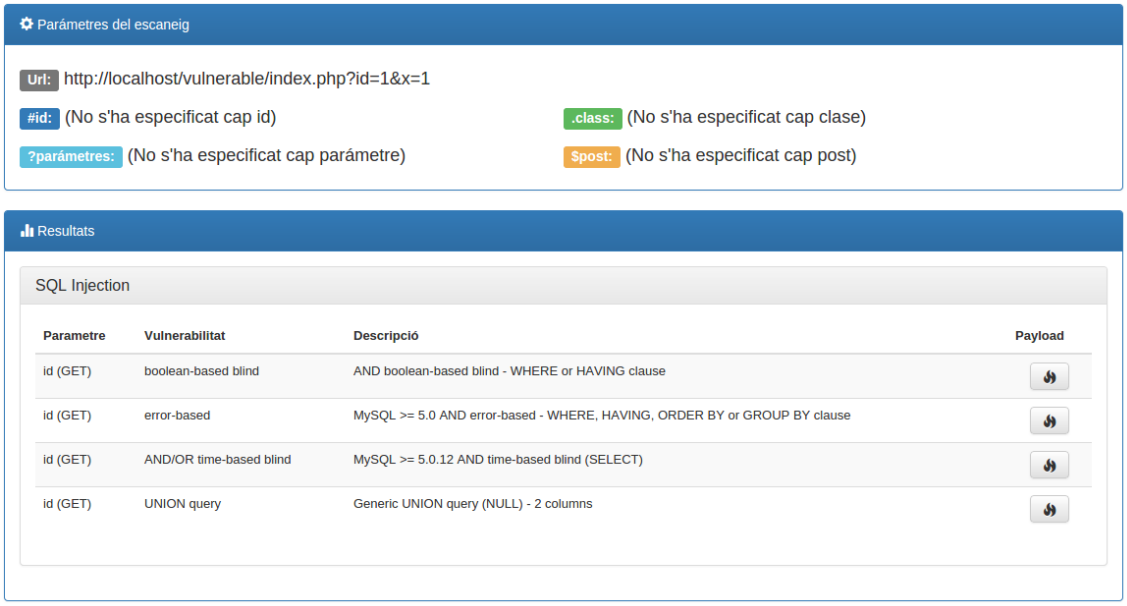


Figura 24. Resultat SQL Injection

XSS

Omplim el formulari amb la pàgina vulnerable a XSS. En aquest cas és la mateixa que l'anterior.

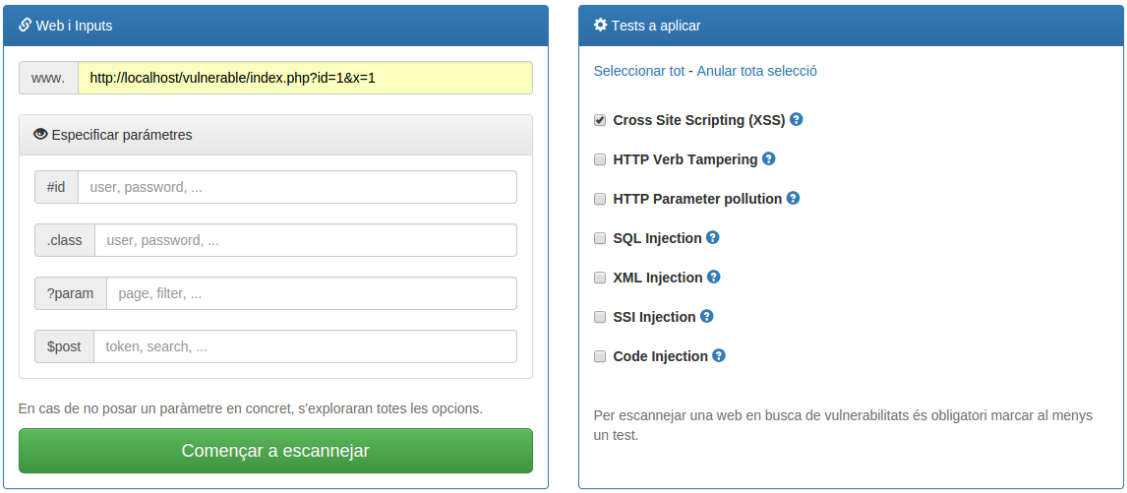
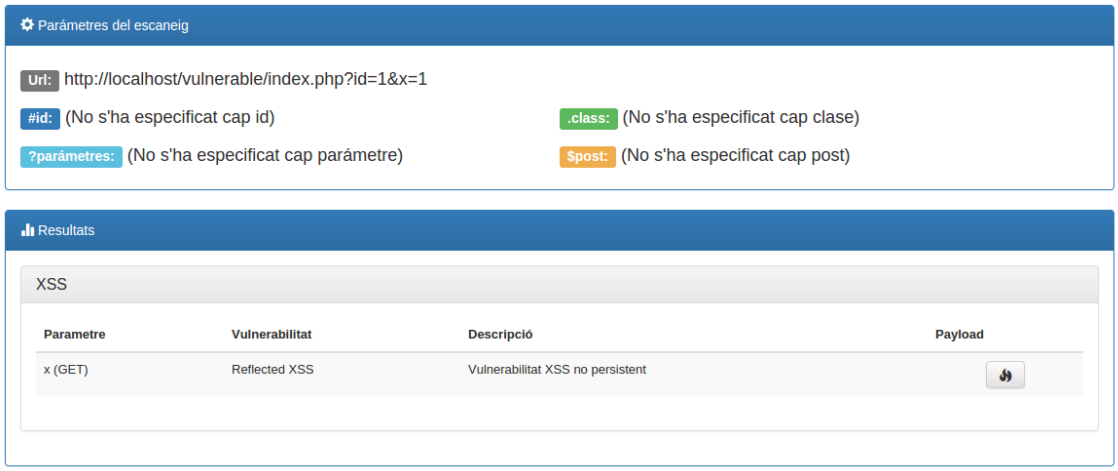


Figura 25. Formulari XSS

Obtenim el següent resultat:



HTTP Verb Tampering

Per aquest test tampoc ha fet falta generar una nova pàgina vulnerable, ja que més aviat es tracta d'una vulnerabilitat de la configuració del servidor, però aquesta vegada no cal que li passem cap paràmetre per URL.

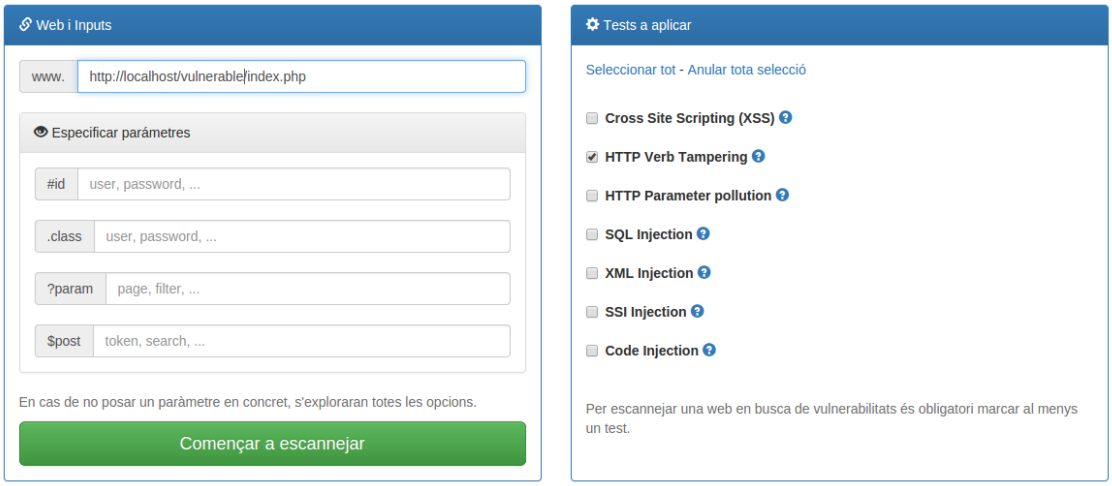


Figura 26. Formulari HTTP verb tampering

El resultat és el següent:

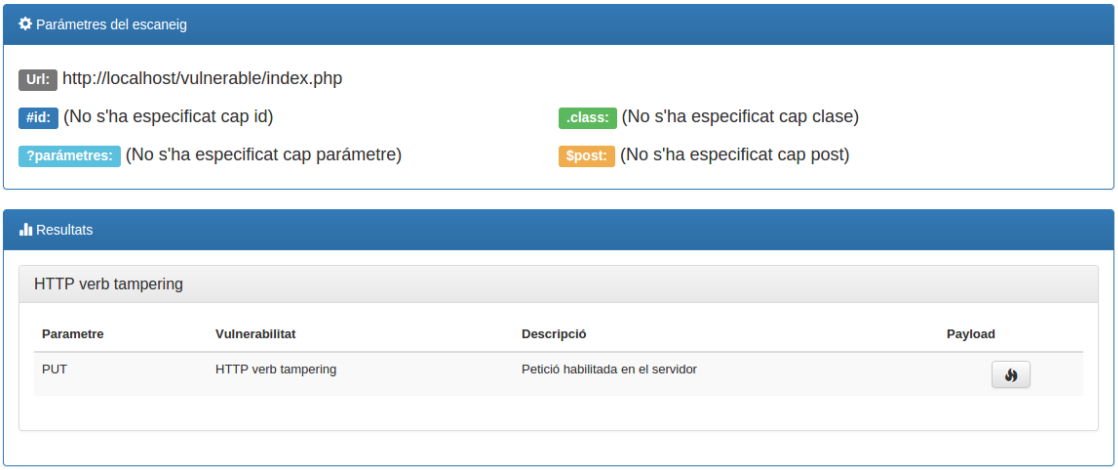


Figura 27. Resultat HTTP Verb Tampering

HTTP Parameter Pollution

Per aquesta vulnerabilitat s'ha creat una nova pàgina vulnerable amb un nou paràmetre.

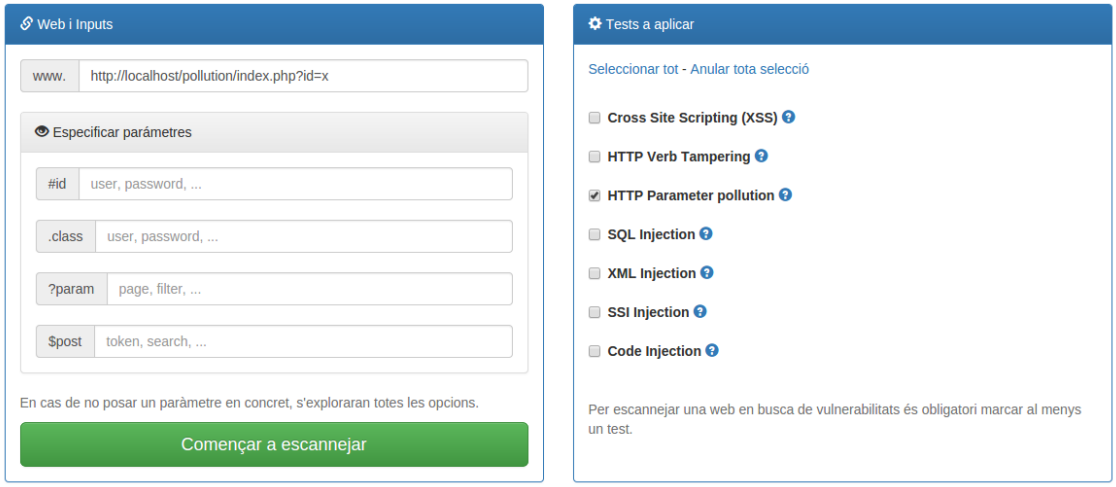


Figura 28. Formulari HTTP Parameter Pollution

El resultat és el següent:

Figura 29. Resultat HTTP Parameter Pollution

XML Injection

Per aquest test s'ha hagut de crear una nova pàgina vulnerable que insereix un registre en un fitxer XML d'aquest estil:

```
<usuaris>
	<usuari>
		<nom>Eric</nom>
		<edat>27</edat>
	</usuari>
</usuaris>
```

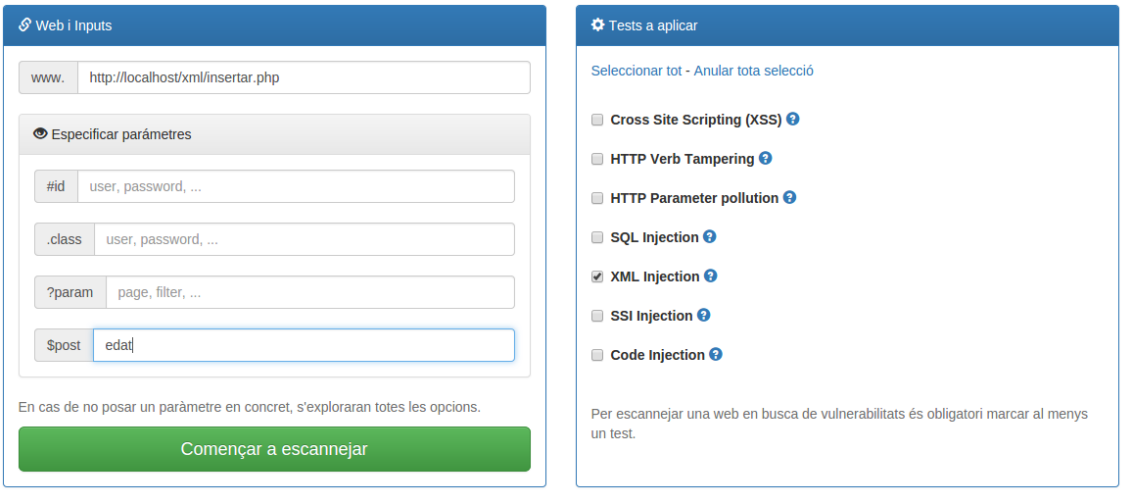


Figura 30. Formulari XML Injection

El resultat és el següent:

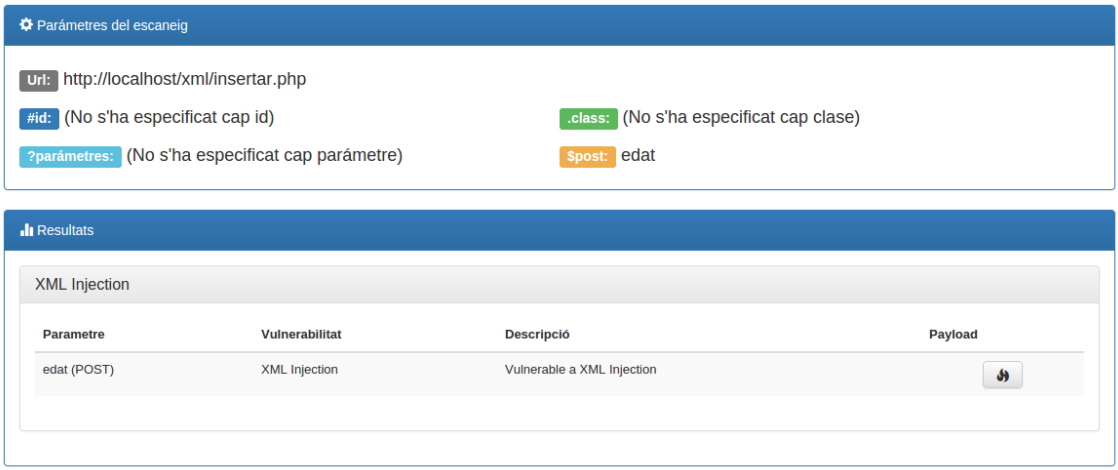


Figura 31. XML Injection

SSI Injection

Per fer aquesta prova s'ha hagut d'habilitar el SSI en el nostre servidor Apache.

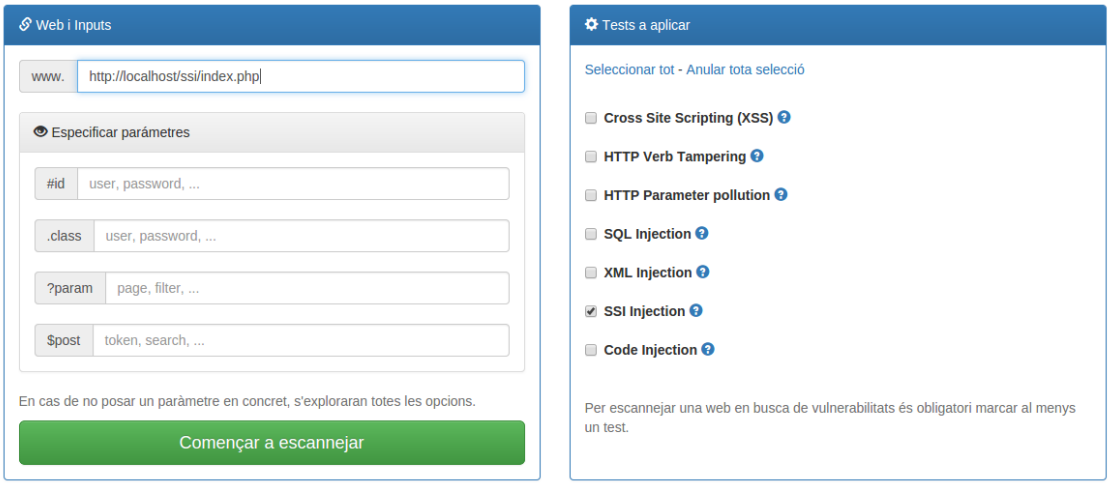


Figura 32. Formulari SSI Injection

El resultat és el següent:

Figura 33. Resultat SSI Injection

Code Injection

Per provar aquest tests, s'ha generat una pàgina PHP que, a través del paràmetre "*p*", crida a la pàgina del valor del paràmetre més "*.php*", sense cap tipus de validació.

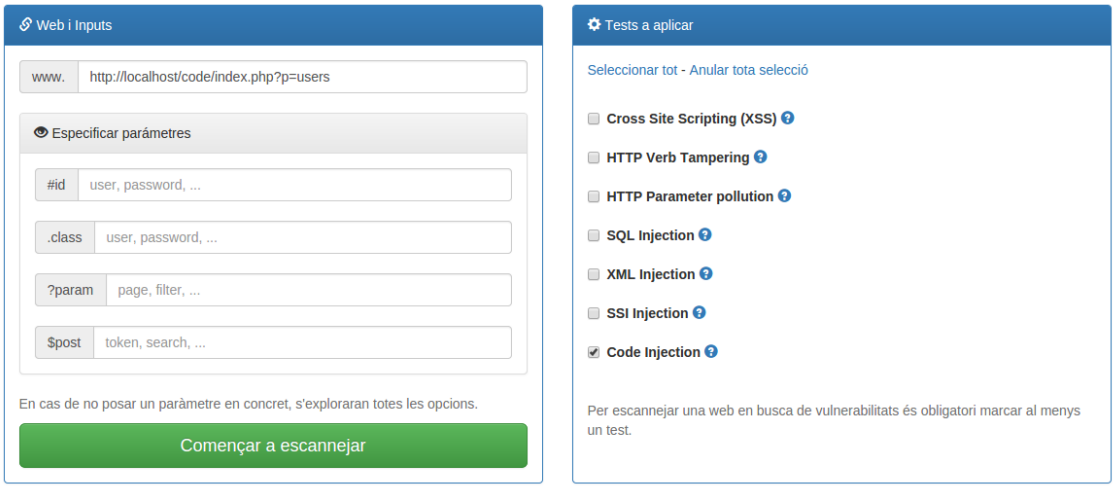


Figura 34. Formulari Code Injection

El resultat és el següent:

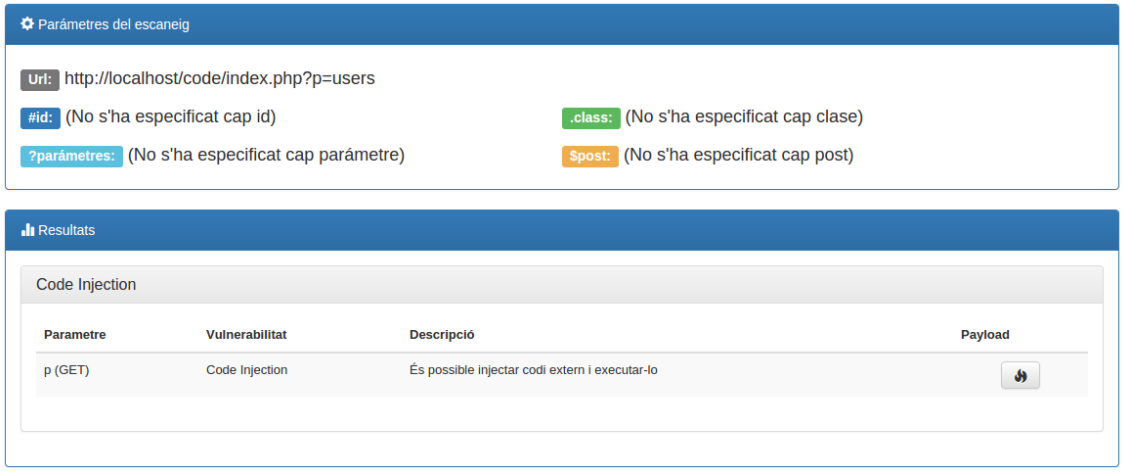


Figura 35. Resultat Code Injection

7.3 Proves de maneig de la informació

Bàsicament el que es vol provar aquí és que guarda correctament la informació a la base de dades i la recupera sense problemes. Per fer-ho, només hem de passar un test anterior (en aquest cas ha estat el primer de tots, amb *SQL injection*) i veure que recuperem la informació sense problemes.

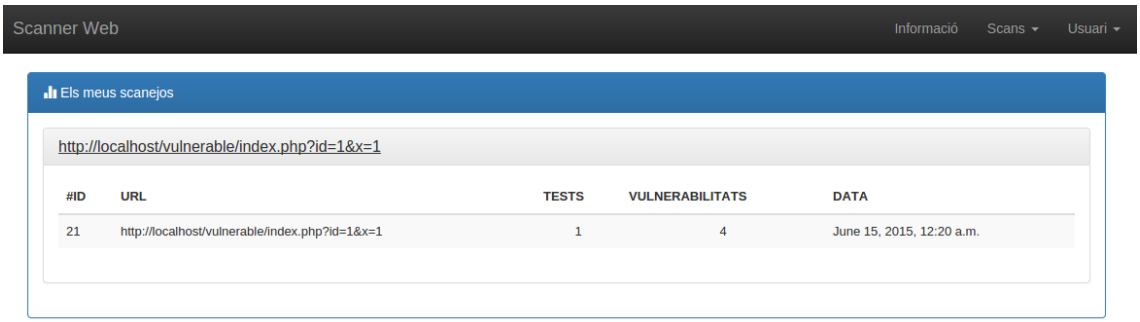


Figura 36. Els meus escanejos

8. Conclusions

A continuació s'exposen les conclusions extretes una vegada finalitzat el projecte.

8.1 Conclusions del treball

Aquest treball ha estat molt enriquidor a nivell personal, he estat programant amb PHP tota la vida fins que vaig començar a programar *Python* per raons professionals, Em va agradar molt i per això vaig decidir fer el projecte final de màster amb aquest llenguatge.

Fa pocs anys que em dedico a la programació web, però mai havia vist un *framework* tant senzill i potent com ***Django***, no m'ha donat cap problema des de que vaig començar i ha resultat molt fàcil d'aprendre i de programar.

Com a conclusió final puc extreure que *Python* y *Django* han resultat estar per sobre de les meves expectatives, tant que he decidit deixar de programar en PHP pels meus projectes personals. I per altre banda, m'ha agradat molt el fet de poder aprofundir en el tema de les vulnerabilitats web, m'he adonat de que la seguretat en general no es té en compte com hauria, i que pren més importància que l'aplicació faci el que ha de fer per davant de que pugui ser utilitzada per tercers amb finalitats les quals no està pensada, com per exemple l'enviament d'*spam*.

Aquest projecte m'ha donat la visió sobre com programar pensant en la seguretat del codi en tot moment.

8.2 objectius

Finalment s'han assolit tots els objectius, tot i que ha faltat temps per poder programar més tests. En un principi pensava que hi hauria més eines programades en *python* per l'anàlisi de vulnerabilitats però ha resultat que no era així, o les que hi havia necessitaven més llibreries o software extern del que s'esperava.

8.3 Planificació

La planificació sincerament ha estat un fracàs, ja que s'esperava poder fer molta més feina els primers mesos del que realment s'ha pogut. Ha estat impossible seguir la planificació i sempre s'ha anat assolint les tasques amb retràs. Això ha sigut degut a pics de treball inesperats, tant de l'empresa com propis, i alguns problemes personals que m'han tret molts més dies del que espera.

Finalment, per assolir els objectius, s'han hagut de concentrar tots els recursos i hores en l'últim mes i mig.

8.4 Líneas futures

Les Líneas futures són força clares:

- Acabar de definir tots els tests de OWASP.
- Millorar la interacció de l'usuari amb els seus escanejos, poder esborrar resultats o comparar-los de forma més visual.
- Gràfiques evolutives de vulnerabilitats trobades en una mateixa web.
- Millorar l'eficiència dels tests.
- Posar una barra de percentatge dels tests que es van executant, i que vagin apareixen els resultats conforme els tests van acabant.

Aquestes són les tasques amb les que m'he quedat amb ganes de poder fer i que possiblement s'acabi duen a terme en un futur. A més, he anat millorant les habilitats de programació en *Python* y *Django* i estic segur que el codi es pot optimitzar, així que seria quelcom a tenir en compte en un futur també.

9. Glossari

A continuació s'exposa la definició dels termes i acrònims més rellevants utilitzats dins la Memòria.

- **OWASP**: "*Open Web Application Security Project*", és un projecte dedicat a determinar i combatre les causes que fan que el software sigui insegur.
- **URL**: "*Uniform Resource Locator*", en aquesta memòria s'utilitza per fer referència a la direcció web de les pàgines.
- **Python**: Llenguatge de programació interpretat.
- **PHP**: Un altre llenguatge de programació web interpretat.
- **SQL**: "*Structured Query Language*", és un llenguatge declaratiu d'accés a base de dades.
- **Django**: *Framework* web escrit en *python*.
- **Framework**: Conjunt de biblioteques i organització de codi per facilitar la programació.
- **Spam**: Correu brossa.
- **Decorator**: Característica de *python* que afegeix funcionalitat a les classes.
- **Requests**: Llibreria de *python* per fer peticions web.
- **String**: Tipus de dades, cadena de caràcters.
- **Parser**: Analitzador sintàctic.

10. Bibliografia

Els recursos bibliogràfics han estat majorment webs, ja que no s'han consultat llibres ni articles de revista.

1. **http://stackoverflow.com/** - No es pot definir la data ni es pot incloure tots els enllaços, la he estat visitant varies vegades al dia durant tot el desenvolupament del projecte.

2. **https://docs.djangoproject.com/en/1.8/intro/tutorial01/** - És l'enllaç al primer capítol del *tutorial* oficial de *Django* que he seguit, des de que vaig començar amb el projecte. (01-03-2015 - 13-06-2015).

3. **http://docs.python-requests.org/** - (12-05-2015)

4. **http://sqlmap.org/** - (10-03-2015, 12-05-2015)

5. **https://docs.python.org/2/library/subprocess.html** - Documentació de la funció *subprocess* de *python*. (13-05-2015).

6. **https://docs.python.org/2/library/urlparse.**html - Documentació de la funció *urlparse* de *python*. (15-05-2015)

7. **https://www.owasp.org/index.php/OWASP_Testing_Guide_v3_Table_of_Contents** - (01-03-2015 i gairebé cada dia de la programació dels tests, de fet, es va imprimir i enquadernar en PDF per facilitar la consulta).