Säkerhet i webbapplikationer

Roger Hinders, 35648

Kandidatavhandling i datavetenskap
Åbo Akademi
2017
Handledare: Dragos Truscan

Referat

Dom senaste decennierna har Internet och World Wide Web fått en allt större spridning i samhället. Detta har lett till att kritisk och konfidentiell data har blivit allt mer tillgänglig genom system kopplade till Internet. Webbaserade system är väldigt flexibla och enkla att underhålla samt utveckla, och därför har såna system blivit väldigt populära och används för allt från Internetbanker till diskussionsforum. Det har dock genom åren varit många problem med intrång och stöld utav data vilket har lett till stora problem. Denna avhandling undersöker dom vanligaste säkerhetshålen som används för att bryta sig in i dåligt säkrade webbapplikationer. Avhandlingen kommer också att analysera Python ramverken Django och Flask för att jämföra hur dom två olika ramverken är byggda för att förhindra dessa problem.

Innehåll

1. Inledning

Denna avhandling kommer att gå igenom dom vanligaste säkerhetsproblemen som en programmerare måste ta i beaktande under utvecklingen av en webbapplikation. Den kommer att ta upp två olika ramverk för webbapplikationer att analyseras, med målet att se vilka metoder som används för att förhindra dessa problem, och hur dom skiljer sig åt i det avseendet.

En webbapplikation är en applikation som erbjuder anpassat innehåll till klienter som ansluter över HTTP-protokollet till en webbserver med en webbläsare. Till en webbapplikation hör olika delar. HTML beskriver hur applikationen skall se ut grafiskt. Varje förfrågan till servern skapar dynamiskt ett HTML-dokument genom att webbserver kör ett program som genererar sidan enligt valda parametrar, som sedan skickas till klienten för att visas. Ett sådant program kan vara programmerat i många olika språk, men som exempel i denna avhandling används Python. För att höja en användares upplevelse kan JavaScript användas. JavaScript kör kod lokalt i webbläsaren för att utföra kosmetiska effekter på HTML-dokumentet och göra sidan mer interaktiv. En webbapplikation måste i många fall ha möjlighet att spara och läsa data. Detta kan göras med enkla filer sparade direkt på filsystemet, men det är inte väldigt effektivt och därför behövs en mera effektiv lösning. Som lösning för detta kan olika typer av databaser användas. I denna avhandling kommer en relationsdatabas användas som lagringsteknologi. Relationsdatabaser kan snabbt lagra och läsa stora mängder av information och gör en webbapplikation mer skalbar.

När en webbapplikation planeras och utvecklas är det väldigt viktigt att ta i beaktande säkerheten. Att förbise säkerheten kan ha förödande konsekvenser och kan orsaka att viktig och konfidentiell information hamnar i fel händer. Vanligtvis så används nuförtiden nån form utav ramverk när webbapplikationer utvecklas. Dessa ramverk har oftast inbyggda funktioner och åtgärder för att hantera dom vanligaste typerna av säkerhetsproblem som man kan stöta på.

Ramverken som har valts för att analyseras i denna avhandling är Django, som är ett komplext ramverk, men som har mycket funktionalitet som redan är inkluderad som standard i ramverket, och Flask som är ett lättare ramverk som förlitar sig på att

utvecklaren skall använda tredjepartsmoduler för att lägga till behövlig funktionalitet. Båda av dessa ramverk är programmerade i programmeringsspråket Python.

2. Teknologier

World Wide Web bygger på tre grundstenar: transportering (*Hypertext Transfer Protocol*, HTTP), representering (HTML, JPEG, GIF etc.) och länkningsinformation (*Uniform Resource Information*, URI). För att hämta information ansluter en klient till server över HTTP-protokollet, och anger en resurs specifierad med en URI sträng, som anger under vilken identifiering på server som informationen finns lagrad. Server skickar sedan tillbaka informationen som hittas över samma protokoll, och klientens webbläsare kan visa informationen till användaren. [25]

HTTP är ett tillståndslöst klient/server-protokoll som fungerar enligt modellen förfrågan och svar. Varje HTTP-förfrågan är självständigt och det finns inte någon länk mellan två olika förfrågningar. HTTP-protokollet har definierat en samling med förfrågningskommandon som en klient kan använda för att utföra önskad förfrågan. För att göra ett anrop som hämtar en fil, till exempel ett HTML-dokument eller en bildfil, ansluter klienten till en HTTP-server och skickar ett *GET*-kommando med tillhörande URI. Server svarar sedan med att skicka filens data förutsatt att den finns. Efter att server har svarat avslutas anslutningen. För att skicka information till servern, som till exempel en fil, eller för att skicka text som har skrivits in i ett HTML-formulär används *POST*-kommandot. Klienten öppnar en förfrågan med kommandot till en URI på servern som hanterar lagring av data. I förfrågan inkluderas också datan som skall lagras eller behandlas. [23]

Hypertext Markup Language (HTML) är ett märkspråk som används för att beskriva hur en webbsida ser ut. Ett HTML-dokument är en enkel textfil som byggs upp genom att använda HTML-taggar som beskriver olika grafiska layouter. En viktigt HTML-tagg är <*form*>-taggen som används för att visa formulär på en webbsida som en användare kan fylla i och skicka informationen till server genom en POST-förfrågan.

Som exempel för att visa hur en enkel sida som skriver ut ”Hej!” byggs upp, kan den beskrivas i HTML som följande[24]:

```
<html>
    <head></head>
    <body>Hej!</body>
</html>
```

JavaScript skapades 1995 av Brendan Eich som då jobbade på företaget Netscape.[5] JavaScript är ett lättviktigt objektorienterat programmeringsspråk som vanligen används för webbsidor. Språket är prototypbaserat och dynamiskt. Olika paradigm av programmering stöds så som objektorienterad, imperativ och funktionell programmering. JavaScript-kod hämtas antingen som en extern JavaScript fil eller inbäddad i HTML-kod från en webbsida av webbläsaren, kör sedan koden lokalt i webbläsarens tolk, till skillnad mot Python som är serverbaserat.

JavaScripts primära bruk är för att göra webbsidor mer interaktiva och för att ge utvecklaren mera valmöjligheter till designen utav en webbsida. Objekt i JavaScript skapas programmatiskt genom att lägga till metoder till tomma objekt under körtid. Objekten kan sen användas som grund för att skapa nya objekt med samma egenskaper.

JavaScript misstas ofta för att vara besläktat med programmeringsspråket Java, men detta stämmer inte och språken är väldigt olika i funktionalitet, även om syntaxen har en viss liknelse. [6]

Ajax är en JavaScript-teknologi som kan användas i webbapplikationer för att asynkront göra anrop till webbserver från webbläsaren som görs i bakgrunden utan att huvudanslutningen till webbserver påverkas. För att utveckla interaktiva webbapplikationer laddas traditionellt sett alltid en ny sida när nytt innehåll behöver hämtas från webbserver. Detta skapar dock en sämre användarupplevelse, och gör att webbklienten laddar en massa onödig information som antagligen redan har laddats från förut. Genom att använda Ajax anrop kan webbläsaren till exempel periodvis hämta ny information från webbserver utan att behöva ladda om webbsidan. Detta är väldigt användbart för olika webbapplikationer som till exempel e-postapplikationer och nyhetsapplikationer. Ajax använder sig av objektet *XMLHttpRequest* i JavaScript

som kan konfigureras enligt preferens och sen användas för att göra ett anrop till webbserver. [17]

En webbserver kan skicka användarspecifik information till en klients webbläsare genom att lägga till parametern *Set-Cookie* i huvudet på serverns svar och inkludera data. Med detta så kan webbläsaren spara informationen som en variabel med namn och värde. En sån variabel kallas för en kaka (Engelska: Cookie). När webbläsaren sen skickar en förfrågan till servern kommer denna kaka att inkluderas genom parametern *Cookie* i huvudet på klientens förfrågan. Genom att använda denna kaka kan servern identifiera en klient och associera den till specifika förfrågningar. Nackdelen med att använda kakor är dock att användaren kan enkelt läsa och modifiera data som sparas. I vissa fall så får informationen inte ändras mellan förfrågningar. För att undvika detta kan sessioner användas. En session startas för att associera data på servern med förfrågningar. Varje session har en unik ID som vanligtvis sparas i en kaka, som sen automatiskt kommer skickas med i varje förfrågan till servern. Med detta så kan servern leta upp rätt information på servern som tillhör medskickat ID. Sessioner används ofta till att hålla reda på om en användare är autentiserad på servern genom att till exempel spara en boolesk variabel som indikerar att en lyckad autentisering har genomförts. Genom att använda sessioner så behöver användaren inte skicka med ett lösenord varje gång en förfrågan som kräver autentisering görs. [26]

Python är ett objektorienterat programmeringsspråk med en enkel och ren syntax. Det kan också enkelt kopplas till externa moduler och bibliotek för att bygga ut möjligheterna med språket. En stor fördel med Python är att det är portabelt. Det kan lätt installeras på olika operativsystem som till exempel olika varianter av Unix, Mac OS, Microsoft Windows och MS DOS.[4] Python kan användas som programmeringsspråk på en webbserver för att dynamiskt generera dokument och processera information när en klient gör en förfrågan. Python har valts som programmeringsspråk i denna avhandling för att språket är lätt att förstå och är ett ideellt språk för nybörjare som inte har så mycket erfarenhet i programmering. Detta hjälper till att lägga fokus på koncepten som presenteras i denna avhandling, istället för att fokusera på komplicerade syntaxer och metoder i programmeringsspråket.

Relationsdatabaser utgörs av en samling tabeller. För varje tabell specificeras ett visst antal kolumner. Dessa kolumner har alla en datatyp som till exempel text för

textsträngar, heltal, olika datumtyper för att representera datum och tid, binärdata och flyttal. Tabellerna är indelade i rader som innehåller data som motsvarar kolumnernas datatyper. Som exempel en tabell med namn *persons* som sparar information om personer. I Tabell 1 nedanför är ett exempel av en tabell som har kolumnerna *first_name*, *last_name*, *age*, *phone* med tre rader data som representerar tre olika personers uppgifter som finns lagrade i tabellen.

first_name(text)	last_name(text)	age(int)	phone(int)
Johan	Johansson	22	12452234
Peter	Petersson	45	55202345
Sven	Svensson	34	44245564

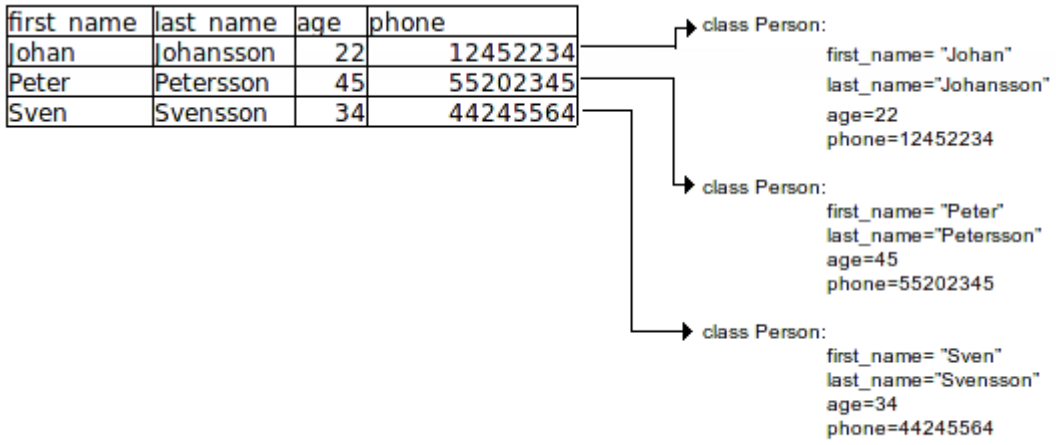
Tabell 1: Representation av en enkel databastabell

I relationsdatabaser används oftast ett förfrågningsspråk. Den vanligaste dialekten är SQL som är en klass med språk som är väldigt enkla att använda, men som erbjuder ett praktiskt och kraftfullt gränssnitt till databasen. Nedanför är ett exempel på en SQL-förfrågning som hämtar alla rader som har kolumnen *age* satt till 22 i Tabell 1:

SELECT * FROM tabell WHERE age=22;

SQL databaser erbjuder snabba förfrågningar och är därför ett bra alternativ att använda som lagring utav data som behöver lagras och användas i en webbapplikation.[20]

En Object-relational mapper (ORM) ger en abstraktion på hög nivå för interaktion med en relationsdatabas. Istället för att skriva anrop till SQL-databasen så representeras tabellerna som objekt, vilka har attribut som motsvarar kolumnerna i databasen. Objekten kan direkt manipuleras i kod för att komma åt data från databasen. [19]



Figur 1: Diagram hur databastabell kan representeras som Python-objekt. [19]

En ORM ger utvecklaren enkla funktioner för att på ett snabbt och smidigt sätt ändra, skapa och ta bort information i databasen genom kod. Fördelen är att programmeraren endast behöver använda programmeringsspråkets syntax, utan att blanda in en hel del med SQL-förfrågningar. Detta ger koden ett enhetligt utseende och struktur, och programmeraren behöver endast fokusera på att använda en syntax. En ORM gör också att utvecklingstiden blir snabbare jämfört med att använda rena SQL-förfrågningar.

ORM gränssnitt har dock en del nackdelar. Genom att ha gränssnittet att översätta objektmanipulering till SQL-förfrågningar skapar det möjlighet att prestandan inte är lika bra som den är jämfört med att manuellt göra en SQL-förfrågning. Att representera tabeller och relationer i objekt fungerar inte heller alltid optimalt eftersom hanteringen av data och relationer i en tabell inte fungerar på helt samma sätt som hur det görs i objektorienterad kod. Ett annat problem är att istället för att ha mycket av komplexiteten i SQL-förfrågningarna, så flyttas denna komplexitet ut i webbapplikationens kod.[19] Om en modell med namnet Person har definierats i Python för SQL-tabellen i Tabell 1, så kan en förfrågan för att hämta alla rader med *age* satt till *20* se ut som i följande exempel:

Persons.objects.filter(age=20)

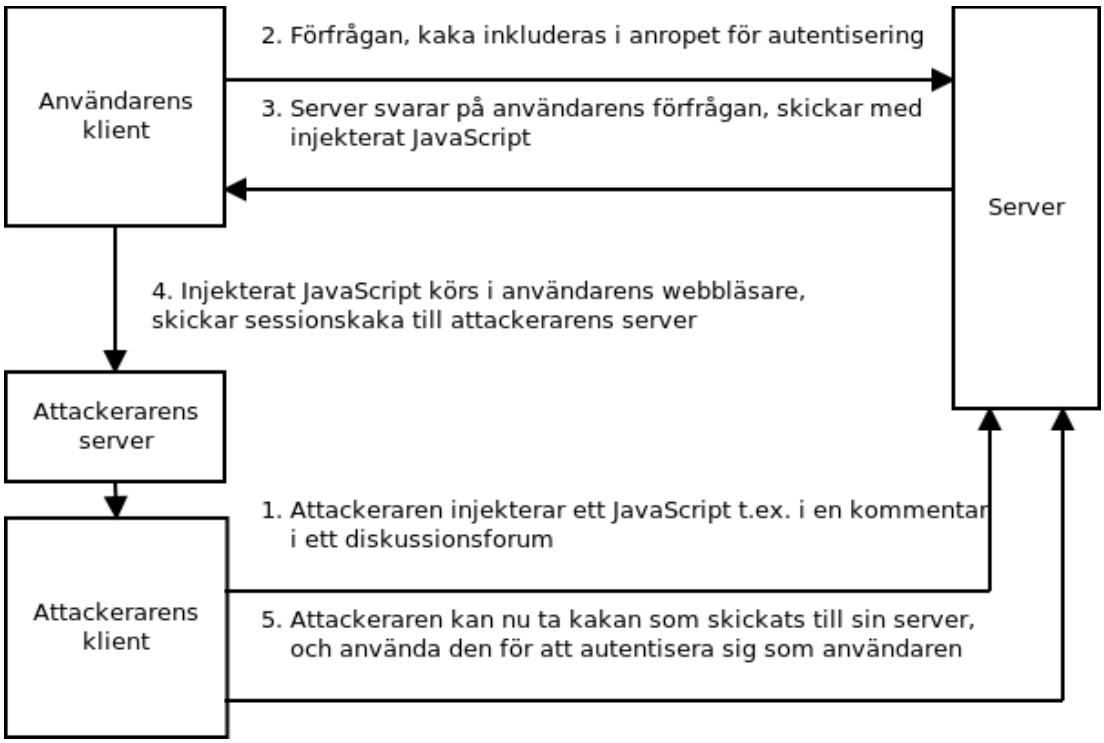
3. Vanliga säkerhetsproblem

I denna sektion kommer tre typer av säkerhetsproblemen att diskuteras. Problemen som har valts, har valts för att dom är vanligt förekommande problem som ofta används för att utnyttja svagheter i webbapplikationer. Dessa problem utnyttjar sårbarheter som har uppkommit eftersom att dom olika teknologierna som uppbygger World Wide Web inte från grunden har utvecklats med tanke på säkerhet, eftersom det inte behövdes när grundtanken var att endast visa statiska webbsidor.

3.1 Cross site scripting

En av dom vanligaste typen av säkerhetshål är Cross Site Scripting (XSS). Flera stora aktörer som Google, Facebook, eBay med flera har utsatts för denna typ av attack genom åren. För att en XSS attack skall vara möjlig förutsätter det att webbapplikationen inte utför korrekt sanering på data som skickas till servern utav användaren. Idén bakom en XSS attack är väldigt enkel. Genom att skicka specialtecken eller taggar till servern, så kommer webbläsarens tolk att köra kod istället för att endast skriva ut data.

Som exempel så kan en person skriva in JavaScript kod i ett HTML-formulär på en webbsida. Om servern sen inte sanerar informationen genom att filtrera bort <script> taggarna, och om informationen kommer att skrivas ut till en annan användare, så kommer webbläsaren att köra JavaScript-koden. Detta medför att den som skapade koden kan utföra olika uppgifter på användarens webbläsare, så som att stjäla kontodata, kakor, Denial of Service(DoS) attacker eller manipulera webbsidans innehåll. För att skydda användarna av en webbapplikation mot denna typ av attack krävs det att all indata som skickas till servern genomgår en strikt sanering före lagring i databas, för att göra det omöjligt att skicka med exekverbar kod.[1] För att illustrera hur en attack sker stegvist, se *Figur 2*.



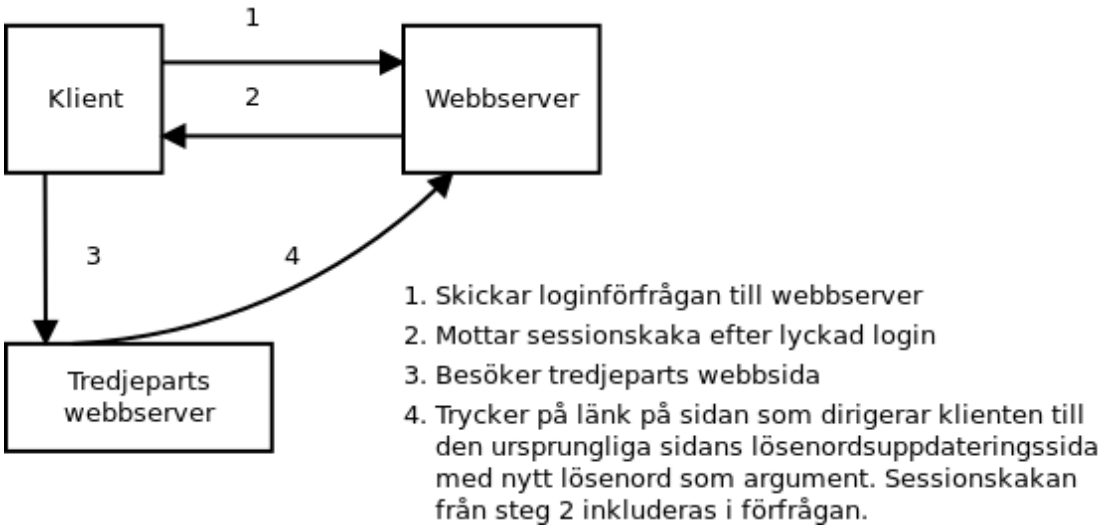
Figur 2: Diagram över hur en lyckad XSS-attack kan gå till.

3.2 Cross site request forgery

Genom att utföra en lyckad Cross Site Request Forgery (CSRF) attack kan en utomstående person skicka HTTP anrop från en annan användare till en webbapplikation. Om den användaren är autentiserad på webbapplikationen som har en CSRF sårbarhet, så kan den som utför attacken göra sånt som endast den autentiserade användaren normalt har tillgång till. Detta beror förstås på webbapplikationen, men som exempel kan nämnas att skicka meddelanden i användarens namn, ändra personlig data, radera eller ändra filer och texter.

Sessioner och kakor som diskuterades i sektion 2 är väldigt användbara men öppnar upp för sårbarhet genom CSRF attacker. Webbservern kan ej skilja på om en förfrågan är medveten från användaren, eller om någon har lurat användaren att öppna en länk i sin webbläsare som går till den autentiserade servern. Som exempel kan en webbapplikation med adressen *www.domain.com/edit/* ta parametern *password* för att ändra en användares lösenord. Den som utför attacken kan då konstruera en länk som *www.domain.com/edit/?password=nyttlosenord*. Om den länken öppnades av en autentiserad användare, så skulle användarens lösenord bytas

till *nyttlosenord*. Detta är en allvarlig attack och kan lätt utföras genom att lura användaren att trycka på länken, till exempel genom att skicka användaren en e-post med länken, eller skapa en webbsida med länken på som sen användaren luras att gå in på.[2] Se *Figur 3* som illustrerar hur denna attack kan ske.



Figur 3: Exempel på hur lyckad CSRF-attack som ändrar en användares lösenord kan gå till.

3.3 SQL-injektioner

SQL injektion är en vanlig attack som kan göra väldigt mycket skada och kan hända om en webbapplikation inte sanerar indata från en användare som skall inkluderas i ett SQL kommando till en relationsdatabas som till exempel MySQL. SQL injektioner är endast ett potentiellt problem på webbapplikationer som använder en relationsdatabas för att spara och läsa data.

För att demonstrera hur en injektion konstrueras används pseudokod. Låt säga att variabeln *input* innehåller ett sökord som en användare har angett i ett HTML formulär. I koden så konstrueras SQL kommandot genom att definiera en sträng *s*. Strängen byggs genom att klistra ihop olika strängar för att bilda ett kommando:

var s = ”SELECT * FROM produkter WHERE name=’” + input + ”’”

Denna sträng kommer sen skickas till relationsdatabasen för att söka efter en produkt med namnet *input.* Om till exempel sökordet som skickats med variabeln *input* till

koden är *dator* så kommer det genererade kommandot som skickas till databasen att set ut som följande:

SELECT * FROM produkter WHERE name='dator'

Och då kommer alla produkter med namnet dator att returneras från databasen. Det finns ett allvarligt säkerhetshål i detta. Om variabeln *input* inte har blivit korrekt sanerad så kan en användare missbruka detta för att skicka egenkonstruerade SQL kommandon till databasen. Som exempel om en användare skickar indata som *dator'; DROP TABLE produkter* – skulle det resultera i följande kommando:

SELECT * FROM produkter WHERE name='dator'; DROP TABLE produkter –'

Detta kommando skulle orsaka att hela tabellen *produkter* raderades från relationsdatabasen. Genom att modifiera indata kan användaren utan lov köra i princip vilket kommando som helst och ställa till stora problem. [3]

4. Säkerhetsanalys av ramverk

I följande sektion kommer två ramverk att analyseras. Ett ramverk är en samling funktioner och strukturer som gör det enklare att skriva och underhålla webbapplikationer. Ramverket ger grundläggande funktioner som förenklar vanliga metoder som används vid utveckling av webbapplikationer. Orsaken till varför man vill använda ett ramverk när man utvecklar webbapplikationer är för att göra processen snabbare och enklare. Genom att använda ett ramverk får man en väldefinierad struktur. Eftersom ett ramverk används av många olika utvecklare i olika projekt, betyder det att man får en vältestad bas att bygga vidare på. Då kan utvecklaren koncentrera sig på applikationslogiken och inte på grundläggande funktioner och strukturer. [22]

Ramverken Django och Flask använder arkitektursmönstret Model-View-Controller (MVC). Det betyder att applikationen är separerad i tre olika delar. Vyn (eng: view) definierar hur applikationen skall se ut och vilken data som skall synas. Modellen (eng: model) definierar vad för data som skall lagras, och som signalerar åt vyn ifall nånting har ändrats i datan, så att vyn kan uppdateras. Kontrollern (eng: controller)

tar emot input av användaren, och uppdaterar modellen och vyn enligt vad användaren gör.[21]

4.1 Django

Django är ett flexibelt Python ramverk som ger utvecklaren möjlighet att på kort tid enkelt skapa webbapplikationer som är både säkra och enkla att underhålla. Ramverket sköter om alla grundläggande funktioner som behövs i en webbapplikation, och låter utvecklaren fokusera på det som är viktigt. Ramverket har en väldigt stor kodbas som inkluderar nästan allt en utvecklare kan tänka sig behöva för att utveckla webbapplikationer. Detta medför att allting fungerar väldigt smärtfritt och funktionaliteten över alla delar är konsistenta.

Dokumentationen i Django är väldigt omfattande och välgjord, vilket förstås är till stor hjälp för en utvecklare. Med Django kan man i princip bygga vad som helst, allt från diskussionsforum och innehållshanteringssystem till nyhetssidor och sociala nätverk. Ramverket är också öppen källkod och har en stor community. Det är ett mycket populärt ramverk med en väldigt stor användarbas. Många högtrafikerade webbsidor och webbapplikationer är byggda med detta ramverk, som till exempel Disqus och Instagram. [8]

4.1.1 Cross site scripting

I Django finns det inbyggt stöd för att förhindra attacker av denna typ. För att Django skall kunna förhindra dessa attacker kräver det att utvecklaren använder den inbyggda mallmotorn när HTML-sidor skapas för att visa innehåll och input-formulär.[9]

En mall i Django är ett HTML-dokument som stöder extra syntax för att dynamiskt generera ett färdigt HTML-dokument som sen skickas till användarens webbläsare för rendering. En av funktionerna i motorn är att skriva ut variabler i dokumentet. Detta sker genom att använda specialtaggen *{{ variabel_namn }}*. Som exempel har

vi en variabel med namnet *userName* som innehåller användarens användarnamn. För att skriva ut användarens användarnamn på en HTML-sida så kan som exempel följande kod läggas i mallen:

*** {{ userName }} ***

När användaren sen besöker sidan kommer mallmotorn att generera ett färdigt HTML-dokument som sen skickas till webbläsaren. Om användaren hade som exempel registrerat sig med namnet *Roger* kommer följande HTML-dokument att genereras och skickas till webbläsaren:

*** Roger ***

När dokumentet genereras så ersätts vissa specialtecken i varje variabel som skrivs ut i dokumentet till motsvarande tecken som är säkra att inkludera i ett HTML-dokument. Se tabell 2 för vilka tecken som ersätts, och vad ersättande tecken är.

Ursprungligt tecken	Ersättande tecken
<	&lt;
>	&gt;
’	&#39;
”	&quot;
&	&amp;

Tabell 2: Tabell över tecken som ersätts vid variabelsanering.[10]

Detta betyder att en användare kan inte inkludera egna HTML-taggar i data som den skickar till servern för lagring, i hopp om att kunna modifiera webbapplikationens beteende. Som exempel om en användare försöker att skicka följande text:

<script>alert(’hej’)</script>

Då kommer mallmotorn att generera ett HTML-dokument med följande innehåll:

<script>alert('hej')</script>

I det första fallet skulle webbläsaren köra JavaScript-koden och öppna en ruta med texten *hej* om koden inkluderades på en webbsida. I det andra fallet skulle texten endast skrivas ut eftersom att HTML-taggarna har ersatts med motsvarande tecken

som är säkra för rendering. Även om detta ger ett visst skydd mot attacker så är det inte helt säkert.[10]

Det finns vissa specialfall där det fortfarande går att kringgå skyddet. Att manuellt med egen kod sanera all inkommande data är fortfarande det säkraste sättet att helt undgå denna typ av attack. [9]

4.1.2 Cross site request forgery

I Django finns det inbyggt stöd för att hantera CSRF-attacker. Modulen som hanterar dessa attacker är aktiverad som standard. Men det löns ändå för utvecklaren att verifiera att så är fallet eftersom att den kan enkelt stängas av i konfigurationen för projektet.

CSRF-attacker i Django fungerar genom att webbserver genererar en hemlig nyckel som den sedan kräver att alla HTTP POST-anrop inkluderar i huvudet på anropet. Om nyckeln inte finns inkluderad, kommer webbserver att neka anropet. Detta gör det svårt för utomstående personer att konstruera anrop som användaren inte är medveten om, eftersom det är praktiskt omöjligt att veta vad den hemliga nyckeln är. Funktionen för att inkludera denna nyckel i POST anrop är dock inte automatisk. När programmeraren skapar webbformulär i HTML-mallen så måste en speciell tagg läggas in. Syntaxen för denna tagg är:

{% csrf_token %}

Taggen skall inkluderas inuti HTML *<form>* taggar. Som exempel följande HTML-formulär:

<form action="" method="POST">
</form>

För att inkludera taggen i formuläret så att mallgeneratorn lägger in nyckeln på ett korrekt sätt, görs det enkelt som i följande exempel:

<form action="" method="POST">
{% csrf_token %}
</form>

Efter det kommer mallgeneratorn automatiskt inkludera den hemliga nyckeln i HTTP POST anropet, och webbservern kan verifiera att anropet kommer från den riktiga användaren, och inte från en tredje part som kan ha en dålig avsikt med anropet.

För att skicka med CSRF-nyckeln med ett Ajax POST anrop så bör det göras på ett sätt som avviker från standardsättet med formulär och mallar. CSRF-nyckeln finns lagrad i en specialkaka som har getts namnet *csrftoken.* Den kan man enkelt hämta från *document.cookie* objektet i JavaScript. Efter att nyckeln har hämtats, så måste den inkluderas i huvudet på *XMLHttpRequest*. Attributet i anropshuvudet som den skall sättas in i heter *X-CSRFToken*. Med detta attribut korrekt satt så kommer Django att acceptera anrop som sker med Ajax. Det finns dock en fallgrop. Om mall-taggen *{% csrf_token %}* inte har använts i nån mall, kan det hända att Django inte genererar denna nyckel alls och anropen misslyckas. Om så är fallet kan genereringen och lagringen av *csrftoken* som kaka explicit sättas genom att anropa *ensure_csrf_cookie()* funktionen i webbapplikationens källkod.

Mallgeneratorn i Django så ger ett ganska bra skydd mot den här typen av attacker. Men det finns ändå en del begränsingar. Subdomäner på samma webbserver som kontrolleras av en opålitlig användare kan sätta kakor för hela domänen, och genom att skicka med sin hemliga nyckel kringgå CSRF-skyddet.[9][11]

4.1.3 SQL-injektioner

Django har inbyggt stöd för att motverka SQL-injektioner i modulen som har hand om databasanrop. Som exempel har vi följande databasanrop, som hämtar alla personer med åldern 20:

SELECT * FROM persons WHERE age=20;

Normalt så skulle detta kommando direkt skickas till SQL-servern. Men i Django så används en ORM (se sektion 2.6) istället som ser till att alla anrop som skickas är säkra och inte kommer orsaka problem med SQL-injektioner. I Django krävs det att alla tabeller och fält i databasen konfigureras genom applikationens modell. I detta fall antar vi att en modell med namnet *Persons* har konfigurerats. För att göra anropet ovan skulle det i Django göras som följande[12]:

from app.models import Persons

Persons.objects.filter(age=20)

Det finns också stöd i Django för att göra SQL-förfrågningar som är egenkonstruerade med modulen RawSQL.[9] Men detta kräver förstås att utvecklaren tar i beaktande säkerhetsproblemen som detta medför.

4.2 Flask

Flask är ett lättviktigt ramverk för webbapplikationer skrivet i programmeringsspråket Python. Ramverket består främst av en mindre kärna med alla grundläggande funktioner som behövs. Även om detta ramverk är väldigt litet så kan det utvidgas genom att installera externa moduler som ger ramverket fler funktioner. Detta ger utvecklaren full flexibilitet i vad som skall köras på webbservern, utan att ramverket tvingar på utvecklaren onödiga funktioner som ej kommer att användas, och därför slösar på resurser.[7] För att generera HTML-dokument i Flask använder det sig av mallgeneratorn Jinja2 som är ett fristående projekt. Det finns också en inbyggd utvecklingsserver för att enkelt testa och felsöka kod i en webbapplikation under utveckling, utan att behöva installera komplicerade servermiljöer. [18]

4.2.1 Cross site scripting

Ramverket Flask använder som standard Jinja2 mallgeneratorn för att generera HTML-dokument.[18] I denna mallgenerator finns det inbyggt stöd för att automatiskt ta bort HTML-taggar i text som skrivs ut från databasen, för att förhindra att olovlig JavaScript-kod skickas till webbläsaren. För att skriva ut innehållet av en variabel med hjälp av mallgeneratorn används syntaxen *{{ variabel_namn }}*. Som exempel för att skriva ut variablerna som innehåller rubriken och texten på en nyhetspost som finns sparad i databasen, om variablerna hette *title* och *body*, så skulle följande kod användas:

*** {{ title }} ***
*** {{ body }} ***

Det skulle resultera i att mallgeneratorn genererar och skickar HTML-dokumentet innehållande följande kod till webbläsaren, om variabeln *title* innehöll ”subjekt” och variabeln *body* innehöll meingen ”test text”:

*** subjekt ***
*** test text ***

Om webbapplikationen hade en funktion för att skicka in dessa nyhetsposter genom ett formulär, och någon med dålig avsikt som har tillgång hade försökt skicka med JavaScript-kod i antingen *title* eller *body* variablerna:

<script>alert('hello')</script>

Då kommer mallgeneratorn att ersätta HTML-taggarna med tecken som är säkra i HTML, och JavaScript-koden kommer endast att skrivas ut, och inte att köras i webbläsaren.

Mallgeneratorn ger inte ett hundraprocentigt skydd, utan det finns vissa specialfall där det går att komma förbi dess funktion.[13]

4.2.2 Cross site request forgery

I detta ramverk finns det inte något inbyggt stöd för att hantera CSRF-attacker. Eftersom flask är ett lättviktigt ramverk har projektet valt att exkludera funktioner för att hantera det, och hänvisar till formulärvaliderings ramverk. [13] Detta leder till att en utvecklare som inte är medveten om denna attack, eller av annan anledning inte väljer att adressera problemet kommer att få säkerhetsproblem i sina webbapplikationer. Det finns dock en del tredjepartsmoduler för att hantera CSRF-attacker. En modul som kan installeras i en Flask-applikation är WTForms. Förutom CSRF-skydd ger också denna modul extra funktionaliteter som till exempel reCAPTCHA, internationalisering och filuppladdningar.[14] Efter installation av modulen så initialiseras den enkelt i koden genom att lägga in följande kod i projektet:

from flask_wtf.csrf import CSRFProtect
csrf = CSRFProtect()

För att skydda ett formulär mot CSRF-attacker skall funktionen *csrf_token()* anropas i mallen inuti en <*input*> taggs *value*-fält som sedan läggs inuti <*form*> taggen för formuläret som skall skyddas mot attacker. <*input*>-taggen måste också ha namnet *csrf_token* för att det skall fungera korrekt. Som exempel följande HTML-formulär:

<form action="" method="POST">
</form>

För att skydda detta formulär mot CSRF-attacker utökas mallkoden till:

<form action="" method="POST">
<input type="hidden" name="csrf_token" value="{{ csrf_token() }}"
</form>

Formuläret kommer nu att skicka en hemlig nyckel till webbserver när HTML POST-anropet görs, och webbserver kan identifiera att det är ett legitimt anrop som inte kommer från en tredje part med dåliga avsikter. Om ett POST-anrop görs utan en giltig CSRF-nyckel kommer anropet att nekas, och ingen skada kan ske.

För att få Ajax-anrop att fungera med Flasks CSRF-skydd kräver det att utvecklaren manuellt specifierar det i *XMLHttpRequest* konfigureringen. CSRF-nyckeln finns inte automatiskt satt i en variabel, utan en variabel måste själv sättas genom att använda mallgeneratorn. Som exempel, för att sätta en global variabel med namn *CsrfToken* innehållande CSRF-nyckeln kan följande JavaScript-kod inkluderas i sidans mall:

<script>
    var CsrfToken = "{{ csrf_token() }}";
</script>

Variabeln *CsrfToken* kan nu kommas åt i JavaScript. CSRF-nyckeln skall sättas i XMLHttpRequest-objektets X-CSRFToken attribut i anropshuvudet.[13][15]

4.2.3 SQL-injektioner

Flask ramverket har som standard endast stöd för SQLite3 som är en lättviktigt databas som inte kräver en separat server. Flasks databasgränssnitt är väldigt enkelt. Gränssnittet har inbyggt stöd för både säkra och sanerade SQL anrop, och möjlighet att göra direkta SQL anrop till databasen. För att göra anrop till databasen används metoden *execute* på databasens instans, som härefter kommer att refereras till som ***database***. Som exempel följande SQL anrop, som hämtar alla personer med *age* kolumnen satt till 30 ur en databastabell med namnet *persons*:

SELECT * FROM persons WHERE age=30

För att göra detta anrop i Flasks databasmodul finns det två olika sätt. I följande två exempel antas att det finns en variabel med namn *alder* som innehåller en siffra som specifierar vilken ålder på personer i databasen som skall hämtas. För att göra ett enkelt anrop används följande kod:

result = database.execute(”SELECT * FROM persons WHERE age=”+str(alder))

Variabeln result kommer efter det innehålla datan för alla personer med åldern *alder.* Att göra anrop på detta vis är det enklaste, men det kommer att öppna upp för problem med SQL-injektioner om utvecklaren inte har sanerat inputdatan korrekt. För att göra anropet säkert från SQL-injektioner kan koden skrivas om på följande sätt:

result = database.execute(”SELECT * FROM persons WHERE age=?”,[alder])

I denna syntax ersätts variabler i anropssträngen med frågetecken, som instruerar Flasks *execute* funktion att sanera och lägga in en variabel från räckan i argument 2 med index som motsvarar position på frågetecken i anropssträngen. Att använda denna typ av anrop är det korrekta sättet i dom flesta fall och en utvecklare bör undvika att själv direkt lägga in variabler i strängen om möjligt.

Denna databasmodul stöder ju dock endast SQLite3, som inte är så optimal att använda i stora applikationer. Därför används ofta diverse tredjepartsmoduler för att få stöd för databaser som till exempel MySQL. Hur säkerheten i olika tredjepartsmoduler hanteras kan variera, och därför måste utvecklaren vara noga med att välja en modul som har en bra implementation med tanke på säkerhet.[16]

5. Resultat

Ramverken Django och Flask har en del likheter. Men på vissa punkter skiljer dom sig också åt i hänsyn till struktur och säkerhet. Båda ramverken har en mallgenerator för att dynamiskt generera HTML-dokument från data som antingen direkt skickas till servern eller som finns lagrad i databasen. Att använda mallgeneratorn istället för att skicka ut egengenererade HTML-dokument är alltid en bra vana eftersom i generatorerna finns det inbyggda funktioner för att göra konsistenta och säkrare dokument.

Cross Site Scripting-attacker kan till ganska stor grad undvikas i både Django och Flask utan att utvecklaren behöver sätta alltför stor tid på att granska sin kod om mallgeneratorerna används på ett korrekt vis. Det finns dock en del specialfall som fortfarande måste tas i beaktande av utvecklaren av webbapplikationen när mallar designas för att helt säkerställa att det inte finns några sårbarheter som öppnar för Cross Site Scripting-attacker. Med tanke på denna säkerhetsrisk har både Django och Flask ungefär lika bra säkerhet.

I ramverket Django finns det inbyggt stöd för att motverka att Cross Site Request Forgery attacker skall kunna orsaka problem i webbapplikationen. Detta stöd finns direkt inbyggt i mallgeneratorn och är väldigt enkelt att använda. Det finns specialfall där detta skydd inte fungerar helt, men i dom flesta webbapplikationerna kommer det antagligen inte att orsaka problem om utvecklaren är medveten om vad begränsningarna i mallgeneratorns skydd för denna typ av attack är.

Ramverket Flask har dessvärre inget inbyggt stöd för att motverka denna typ av attack. Ansvaret är hos utvecklaren att vara medveten om vilka risker detta medför och att på ett korrekt vis motverka att webbapplikationen kan utsättas för denna attack. Det enklaste sättet som togs upp tidigare avhandlingens kapitel 4 är att använda sig utav en tredjepartsmodul som till exempel WTForms, vilken lägger till funktioner i ramverket för att enklare kunna motverka attacker. Möjligheten att själv programmera funktioner för detta finns också, men att använda en modul med stor användarbas ger större garanti att det inte finns några felaktigheter i koden som hanterar säkerheten.

I detta fall så har Django klart en större fördel mot Flask. Eftersom Django kommer färdigt med detta stöd är det bra framförallt för oerfarna utvecklare som kanske inte tänker på att ett sånt här fel finns, men även erfarna utvecklare vinner på detta. Fördelen med Flask är att eftersom det är modulärt, kan utvecklaren själv skräddarsy vilken modul som används, och därefter få ett mer effektivt system som är anpassat till webbapplikationens behov.

Dom två ramverken har båda stöd för att hantera attacker av typen SQL injektion. Denna attack är antagligen den farligaste av attackerna som en webbapplikation kan utsättas för, eftersom det är ganska enkelt att utföra om utvecklaren har gjort ett misstag, och därefter få tillgång till mycket data som möjligen kan vara konfidentiellt.

I Django finns det inbyggt en Object-Relational Mapper för att hantera databasåtkomst. Detta gränssnitt förhindrar dom flesta typerna av SQL-injektioner och låter utvecklaren programmera på ett mer enhetligt och välstrukturerat sätt. Det finns dock fortfarande stöd för att köra egenkomponerade SQL-förfrågningar, men det är inte det primära sättet att hantera databaser i Django, vilket uppmanar dom flesta programmerarna att inte utsätta webbapplikationen för onödiga säkerhetsbrister.

I Flask finns det endast inbyggt enkelt stöd för att ansluta till SQLite3 databaser, som är en lättviktig typ av databas som inte kräver en extern server. Databasgränssnittet som medföljer är tyvärr enkelt att använda på fel sätt, speciellt om en oerfaren utvecklare skall programmera en webbapplikation utan att ordentligt ha läst igenom dokumentationen. För att använda det på korrekt sätt krävs att SQL-strängar konstrueras med frågetecken som platshållare, och att variablerna skickas in som ett eget argument till databasgränssnittet.

För att använda mer avancerade databaser som till exempel MySQL kräver det att utvecklaren installerar tredjepartsmoduler som lägger till ett egetutvecklat databasgränssnitt. Detta medför dock att det är upp till utvecklaren att hitta en modul som är designad och implementerat på ett korrekt sätt. Detta kan vara en nackdel om utvecklaren inte vet vad som är en bra modul och vad som är dåligt, eftersom det kan finnas ett stort antal alternativ att välja mellan. Till fördel är att utvecklaren inte är

tvungen att använda ramverkets egna gränssnitt, utan kan välja en modul som kanske är bättre anpassad till vad som behövs för webbapplikationen.

6. Avslutning

Denna avhandling har tagit upp vanliga säkerhetsproblem som ofta förekommer i webbapplikationer. När en webbapplikation utvecklas är det viktigt att ta i beaktande säkerheten. Genom att vara medveten om vilken typ av attacker som väldigt ofta används när intrångsförsök görs, kan utvecklaren förhindra dessa problem och vara självsäker på att webbapplikationen åtminstone håller en tillfredsställande nivå av säkerhet. För att illustrera dessa metoder så har i denna avhandling analyserats ramverken Django och Flask, två ramverk som bygger på programmeringsspråket Python.

Det finns en del vanliga metoder som används för att attackera webbapplikationer. I denna avhandling togs tre av dessa upp. Metoderna är *Cross Site Reference Forgery*, *Cross Site Scripting* och *SQL-injektioner*. I analysen kom det fram att dom båda ramverken hanterade dessa sårbarheter i olika grad. Sammanfattningsvis kan det konstateras att Django har ett mer enhetligt skydd mot dessa typer av attacker.

Som standard i Flask finns det en del grundläggande funktioner för att skydda webbapplikationer mot en del av dom här sårbarheterna. Men nivån av skydd är dock inte det samma som i Django.

Det man vinner på med Flask är att valbarheten att konfigurera sin utvecklingsmiljö blir större, vilket leder till att en utvecklare har större möjlighet att skräddarsy sin webbapplikation, och att eventuellt öka till exempel smidighet och prestanda i webbapplikationen. På grund av detta är Django ett mer komplett ramverk och lämpar sig bättre att använda för till exempel utvecklare med mindre kunskap. Django är också ett bra ramverk för utvecklare som värderar en robust grund som är vältestad och som låter utvecklaren fokusera på att programmera sin webbapplikation istället för att spendera mycket tid på att konfigurera sin utvecklingsmiljö.

Referenser

[1] L. K. Shar and H. B. K. Tan, "Defending against Cross-Site Scripting Attacks," in Computer, vol. 45, no. 3, pp. 55-62, March 2012.

[2] N. Jovanovic, E. Kirda and C. Kruegel, "Preventing Cross Site Request Forgery Attacks," 2006 Securecomm and Workshops, Baltimore, MD, 2006, pp. 1-10.

[3] Writing Secure Code, 2nd edition

[4] "General Python FAQ — Python 2.7.13 documentation", Docs.python.org, 2017. [Online]. Available: https://docs.python.org/2.7/faq/general.html. [Accessed: 28- Feb- 2017].

[5] "A Short History of JavaScript - Web Education Community Group", W3.org, 2017. [Online]. Available: https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript. [Accessed: 28- Feb- 2017].

[6] "About JavaScript", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed: 28- Feb- 2017].

[7] M. Grinberg, Flask Web Development, 1st ed. Sebastopol, CA: O'Reilly & Associates, 2014.

[8] "Django introduction", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. [Accessed: 28- Feb- 2017].

[9] "Security in Django | Django documentation | Django", Docs.djangoproject.com, 2017. [Online]. Available: https://docs.djangoproject.com/en/1.10/topics/security/. [Accessed: 30- Mar- 2017].

[10] "The Django template language | Django documentation | Django", Docs.djangoproject.com, 2017. [Online]. Available: https://docs.djangoproject.com/en/1.10/ref/templates/language/#automatic-html-escaping. [Accessed: 30- Mar- 2017].

[11] "Cross Site Request Forgery protection | Django documentation | Django", Docs.djangoproject.com, 2017. [Online]. Available: https://docs.djangoproject.com/en/1.10/ref/csrf/. [Accessed: 30- Mar- 2017].

[12] "Making queries | Django documentation | Django", Docs.djangoproject.com, 2017. [Online]. Available: https://docs.djangoproject.com/en/1.10/topics/db/queries/. [Accessed: 30- Mar- 2017].
[13] "Security Considerations — Flask Documentation (0.12)", Flask.pocoo.org, 2017. [Online]. Available: http://flask.pocoo.org/docs/0.12/security/. [Accessed: 30- Mar- 2017].
[14] "Flask-WTF — Flask-WTF 0.14", Flask-wtf.readthedocs.io, 2017. [Online]. Available: http://flask-wtf.readthedocs.io/en/stable/index.html. [Accessed: 30- Mar- 2017].
[15] "CSRF Protection — Flask-WTF 0.14", Flask-wtf.readthedocs.io, 2017. [Online]. Available: http://flask-wtf.readthedocs.io/en/stable/csrf.html#csrf. [Accessed: 30- Mar- 2017].
[16] "Using SQLite 3 with Flask — Flask Documentation (0.12)", Flask.pocoo.org, 2017. [Online]. Available: http://flask.pocoo.org/docs/0.12/patterns/sqlite3/. [Accessed: 30- Mar- 2017].
[17] L. D. Paulson, "Building rich web applications with Ajax," in Computer, vol. 38, no. 10, pp. 14-17, Oct. 2005.
[18] "Welcome | Flask (A Python Microframework)", Flask.pocoo.org, 2017. [Online]. Available: http://flask.pocoo.org/. [Accessed: 30- Mar- 2017].
[19] M. Makai, "Object-relational Mappers (ORMs) - Full Stack Python", Fullstackpython.com, 2017. [Online]. Available: https://www.fullstackpython.com/object-relational-mappers-orms.html. [Accessed: 30- Mar- 2017].
[20] A. Silberschatz, H. Korth and S. Sudarshan, Database system concepts, 6th ed. New York: McGraw-Hill, 2011.
[21] "MVC architecture", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture. [Accessed: 03- Apr- 2017].
[22] "Server-side web frameworks", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. [Accessed: 03- Apr- 2017].

[23] "An overview of HTTP", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. [Accessed: 03- Apr- 2017].
[24] "HTML", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Accessed: 03- Apr- 2017].
[25] "Architecture of the World Wide Web, Volume One", W3.org, 2017. [Online]. Available: https://www.w3.org/TR/webarch/. [Accessed: 17- Apr- 2017].
[26] "HTTP cookies", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Accessed: 17- Apr- 2017].