CoderGram

Security

Datamatiker 4. semester
29. november 2021

Gruppe: CoderGram
Sigurd Arik Gaarde Nielsen – cph-at89@cphbusiness.dk – GitHub: ariktwena
Emil Elkjær Nielsen – cph-en93@cphbusiness.dk – GitHub: eelkjaer

Hjemmeside: https://virkeligsikker.dk/
GitHub: https://github.com/eelkjaer/web_security_exam

Test users (Case-sensitive)

Bruger
E-mail: user@eksamen.dk
Kodeord: userEksamen1!

Admin
E-mail: admin@eksamen.dk
Kodeord: adminEksamen1!

Indholdsfortegnelse

Introduktion
Som en del af datamatiker studiets 4. semester på Cphbusiness, har vi fået til opgave at udvikle et projekt, som sammenfatter det lærte pensum fra faget Security. Vi har derfor valgt at udvikle et lille projekt, som vi føler kan vise vores diversitet og forståelse for sikkerhed i både kodning og systemudvikling. Vi har som gruppe udviklet projektet samme, herunder kode og rapport, og vi tager begge ligeligt ansvar for udførelsen af dette.

Formål
Formålet med rapporten er at dokumentere og beskrive vores tanker bag udviklingen af projektet, ud fra et sikkerhedsmæssigt perspektiv.

Mål
Målet med rapporten er:

- At give studerende på samme faglige niveau en forståelse for projektet.
- At beskrive projektets formål og krav.
- At dokumentere udviklingen af projektet ud fra et sikkerhedsmæssigt perspektiv på en sådan måde, at projektets sikkerhed kan forstås af personer på samme faglige niveau.
- At udvikle en rapport som lever op til de faglige krav der forventes af studerende på datamatikerstudiets 4. semester Security fag på Cphbusiness.
- At skabe et projekt som ikke kan penetreres eller hackes af studerende på samme faglige niveau.

Baggrund
Som et led i vores undervisning i faget Security på datamatikerstudiets 4. semester, lærte vi om de sårbarheder som er forbundet med blandet andet hjemmesider. Vi lærte at sårbarhederne ofte kan skyldes manglende viden, manglende kryptering/hashing af data, eller som oftest dårlig arkitektur som følge af at sikkerheden ikke bliver tænkt ind i hele løsningen fra starten.

Vi besluttede derfor at lave et projekt med fokus på logindelen, som overholdt de gældende retningslinjer som vi havde lært. Ønsket var at skabe et login system, som ikke kan penetreres eller hackes af studerende på samme faglige niveau.

Problemet
I vores undervisning lærte vi om OWASP[1], som er den førende kilde når det kommer til online sikkerhedsforanstaltninger. Hos OWASP kan man finde en top-10 over de emner eller tiltag, som OWASP mener skal have størst opmærksomhed og som kan gavne flest mulige, når det kommer til online sikkerhed. Denne top-10 repræsenterer derfor de største sikkerhedsrisikoer i forhold til OWASP's vurdering, og kan derfor bruges som en guide til at sikre hjemmesider mod bl.a. penetrering og hacking.

Som et led i vores undervisning, gennemgik vi disse tiltag, samt forsøgte at hacke og omgå dem vha. hjemmesiden tryhackme.com. I denne forbindelse blev vi meget overrasket over hvor nemt det var at hacke sig ind på en hjemmeside, samt indsamle data herom. Vi forsøgte derfor at

[1] https://owasp.org

gennemføre penetreringstest på vores projekter fra 2. og 3. semester og opdagede, at de tiltag og praktiser som OWASP pointerede, også var gældende for vores tidligere projekter. Vi havde derfor bygget brugbare, men ikke sikre projekter. Af denne grund blev vores fokus at bygge et login modul, som tog højde for OWASP's top-10 retningslinjer og tiltag, og som ville blive sikker i forhold til hacking og penetrering af studerende på samme faglige niveau.

Teknologier

Vi har til projektet benyttet os af følgende teknologier:

Server:
Vores server er en VM der er hosted hos DigitalOcean. Denne køre Ubuntu Server 20.04 med Docker samt NGINX installeret. På vores Docker er der to containere: MySQL og Tomcat 9. NGINX benyttes til at lave en reverse proxy til vores Tomcat container. I vores Docker container (Tomcat) sættes der system variabler til bl.a. ReCaptcha secrets og vores Pepper som benyttes i hashing. Der er ligeledes flere sikkerhedsmekanismer sat op på vores server, som vi kommer ind på senere.

Webapplikation:
Vores webapplikation er opbygget i Java, hvor vi benytter Javas standard HttpServlets, for at have kontrol med processen, end hvis vi f.eks. benyttede Spring som framework. Dette er et Maven projekt, hvorfor vi benytter en given række dependencies til dette projekt, som bl.a. BCrypt og en dependency som hjælper med "One-time Password" OTP. Selve præsentationslaget laves vha. JSP, herunder JSTL.

Andet:
Vi benytter os af CloudFlare som er en foranstaltning mod bl.a. DDoS angreb. Denne service hjælper ligeledes med at besværliggøre brug af tools som NMAP. Til SSL/TLS-certifikater har vi benyttet Let's Encrypt. Slutteligt har vi til test af vores to-faktor godkendelse benyttet os af "Google Authenticator" appen på mobilen. Denne bruges til at generere OTP. Andre to-faktor godkendelses applikationer kan ligeledes benyttes.

Planlægning

I de næste par afsnit vil vi kigge på hvordan vi har tænkt os at planlægge projektet. Vi vil gennemgå de valg vi har truffet og hvilken muligheder og konsekvenser de kan have på vores projekt.

Krav

For at gøre projektet overskueligt, har vi udarbejdet nogle krav som vi arbejdede ud fra. Målet med vores krav har været at skabe et fuldt funktionelt login MVP[2] som vi kunne bruge til vores "Proof of concept" [3]. Vi har i kravene forsøgt at inkorporerer "Usability"[4] i det omfang at det ikke gik på kompromis med sikkerheden.

[2] https://en.wikipedia.org/wiki/Minimum_viable_product
[3] https://en.wikipedia.org/wiki/Proof_of_concept
[4] https://www.interaction-design.org/literature/topics/usability

For at løse projektet, skulle vores login system opfylde følgende krav:

- Systemet man logger på, skal være opdelt i forskellige brugerrettigheder.
- Systemet skal hash et kodeord.
- Systemet skal så vidt mulig afvise automatiseret login.
- Systemet skal sikres mod "Brute Force"[5] og gentagende loginforsøg.
- Systemet skal benytte sig af to-faktor godkendelse når brugeren har administrationsrettigheder.
- Systemet skal kontrollere kompleksiteten af det valgte kodeord jf. NIST[6] standarden.
- Siderne man tilgår, skal være beskyttet mod utilsigtet adgang.
- Brugere med lavere rettighedsniveau, må ikke kunne tilgå eller rette i informationer som er på et højere rettighedsniveau.
- Systemet skal kunne modstå de penetrationstest som vi har lært på datamatikerstudiets 4. semester i valgfaget Security.

Afgrænsning

For at kunne give projektet størst mulig værdi i den tid vi havde til rådighed, valgte vi at afgrænse projektet med følgende punkter:

- Vi har kun valgt at se på loginfunktionaliteten.
- Loginområdet består af en enkel velkomstside.
- Projektet er ikke delt op over flere servere.
- Projektet er kun gennemtestet af os selv, samt vores medstuderende, på baggrund af den faglige viden vi har.
- Vi udfører kun penetrationstest som vi er blevet undervist i.

Risikoanalyse

For at finde frem til de mest væsentlige sikkerhedsproblematikker, som vores problemstilling belyser, har vi valgt at lave en risikomodel, samt en risikoanalyse med en tilhørende risikovurdering. Modellen og en delvis analyse kan findes i vores bilag, men vi vil i det følgende give en kort beskrivelse af hvordan modellen, analysen og vurderingen skal forstås.

Risikomodellen skal give os et nemt overblik over de aktiver som vi gerne vil beskytte. Her ser vi både på digitale og fysiske aktiver. Vi kan også se hvilken kontrolværktøjer vi har til at beskytte vores aktiver mod potentielle trusler. I den sidste del af modellen kan vi se den mulige motivation, som vores modstandere kunne have, samt et overblik over det fjendebillede som vi mener eksisterer. Fjender skal her forstås som personer, maskiner osv. som bevidst eller ubevist ønsker at opnå adgang til vores aktiver.

I analysen gennemgår vi hver aktiv i forhold til fjendebilledet. Vi prøver at forholde os til scenarier hvor det lykkes modstanderen at opnå målet og derved få adgang til aktivet. På baggrund af dette scenarie forsøger vi at vurderer sandsynligheden samt konsekvensen af handlingen, så vi kan give en kalkuleret risiko. Vi kommer også ind på de handlemuligheder vi har, for at beskytte os mod scenariet.

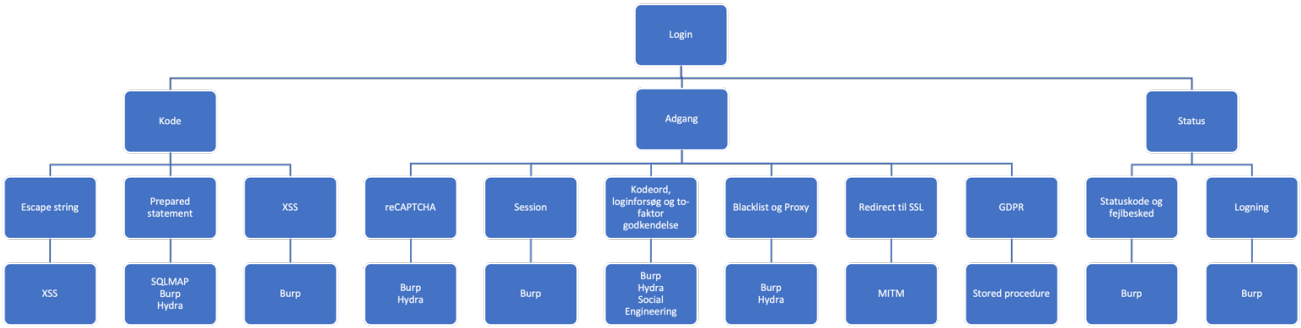
[5] https://da.wikipedia.org/wiki/Brute_force-angreb
[6] https://www.securden.com/blog/top-10-password-policies.html

Da analysen kun fortæller os om risikoen ved et evt. angreb, har vi valgt at inkl. en risikovurdering. Denne vurdering kigger ikke på hvor sandsynligt det er om noget sker, men hvor nemt det er at udføre det pågældende scenarie. På den måde får vi en bedre helhedsforståelse af analysen, da et punkt i risikoanalysen kan være så nem at udføre, at den skal have en højere prioritering end et punkt som vurderes til mellem/høj, men som er sværere og mere urealistisk at udføre, og derved lav i forhold til vurderingen.

Sammenfatter vi vores risikoanalyse med vores krav til systemet, så forsøger vi i dette projekt at løse RB2, RB5, RB6, RB9, RB11, RB12 og RB13, da disse er direkte relaterede til loginprocessen. Målet er at gøre risikovurderingen så lav som muligt, ved at gøre systemet så sikkert som muligt og derved hive vurderinger på de enkelte risikoer op på vurderingen "Svær" mht. udførelsen.

Fremgangsmåde

Da vores projekt består af en masse små problemstillinger, vil vi i dette afsnit prøve at forklare den fremgangsmåde vi har valgt for projektet. Vi har valgt at dele projektet op i tre kategorier. Hver del består af de problemstillinger som vi mener tilhører den pågældende kategori. Under hvert delelement i kategorien har vi skitseret hvilke værktøjer og teknologier som vi vil benytte, for at teste delelementet. Vi vil i de følgende afsnit dykke mere ned i de enkelte delelementer, men først vil vi give et kort overbliv over vores plan. Vi har lavet følgende diagram for at give et visuelt overblik over kategorierne, samt værktøjerne vi vil teste delelementerne med.



Første kategori handler om, at vi vil klargøre vores fysiske kode, så den bliver sikker. I denne kategori handler det om at forbygge fejl i koden, så vi bl.a. kan undgå XSS og injection. For at teste vores udførelse af delelementerne, vil vi primært benytte os af penetrationsværktøjer som kan udføre injection.

Anden kategori handler om selve loginprocessen. Her kigger vi på den handling som en bruger fysisk skal udføre på vores hjemmeside for at logge ind. Vi kigger også på hvordan vi opretholder sikkerheden ved et evt. login, samt hvordan vi sikre brugerdata. I denne kategori kommer vi også ind på automatiserede robotter, som kan udføre loginfunktionaliteten, samt hvordan vi forhindre det. For at teste denne kategori, så benytter vi os ikke kun af automatiserede værktøjer, men også af fysiske som fx MITM[7] angreb.

[7] https://en.wikipedia.org/wiki/Man-in-the-middle_attack

I den sidste kategori vil vi se nærmere på den feedback som brugerne får når loginprocessen fejler. Det er nemlig vigtigt at denne proces ikke giver potentielle hackere brugbar information om vores system, som evt. vil kunne bruges i penetrationsscenarier. Vi vil bruge penetrationsværktøjer til at teste resultaterne, samt arbejde med vores logning, så systemet bliver gennemsigtigt for en systemadministrator.

Proces

I de næste par afsnit vil vi kigge på processen som projektet har været igennem. Vi vil gennemgå de valg vi har truffet og hvilken muligheder og konsekvenser de har haft. Vi vil prøve at give et helhedsbillede af de beslutninger vi har taget, samt hvilke mulige løsninger der kan træffes i fremtiden. Vi vil også komme ind på de udfordringer som projektet har været igennem, samt hvorfor vi har truffet de valg som vi har.

Kode

Escape string:
For at forhindre at en bruger indtaster potentielt skadelig kode, valgte vi at ”escape” det indhold som kommer fra databasen, og som bliver vist på vores frontend. Dette betyder, at vi gemmer alt data i sin rå form. Når indholdet bliver hentet fra databasen, bliver det vist vha. JSP-sidernes ”c:out”[8]. På den måde læser vores JSP-side ikke specielle karaktererne, men viser i stedet det rå indhold som er hentet fra databasen. På den måde ”escaper” vi den hentede data.

Når vi vælger at ”escape” det indhold som bliver vist til brugerne, så gør vi dette på baggrund af vores ”Taint Analyse”[9]. Taint betyder at den data vi modtager, sender eller viser, kan være plette med ondsindet kode. Derfor er det vigtigt at vi forholder os kritisk til den data vi modtager, sender eller viser. Den data vi modtager kommer fra en ”Source”. Når vi fx henter data fra vores database, kan vi rense den vha. en ”Sanatizer”. Når vi viser den, så gør vi det via en ”Sink”. ”Sink” er dog ikke afhængig af vores ”Sanatizer”.

I dette tilfælde, så kan den data vi henter fra vores database være plette, da den består af ”Tainted” brugerdata, som fx brugernavn, e-mail osv. Da vi ikke er sikre på indholdet, så bruger vi ”c:out” som vores ”Sanatizer” for at sikre, at vi ikke eksekverer ondsindet kode på vores frontend når vi skal ”Sink” det indhold som vi har hentet, som fx kunne være <h1>brugernavn</h1>.

Prepared statement:
Når indhold skal gemmes i vores database, så skal man være logget ind med administrationsrettigheder, medmindre man opretter en brugerprofil. Men for at sikre os, at hverken bruger eller administrator gemmer usikkert indhold, så bliver indholdet pakket ind og sendt til databasen vha. ”Prepared Statements”[10] (PS). Dette er også den klare anbefaling fra OWASP, da den sikrer at indholdet fx ikke kan skade eller penetrere ens database.

[8] https://stackoverflow.com/questions/4948532/where-should-i-escape-html-strings-jsp-page-or-servlets
[9] https://www.youtube.com/watch?v=n_l7Dk02eWo
[10] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Præcis som i afsnittet ”Escape string”, så benytter vi os af ”Taint Analyse” når det kommer til PS. Når vi gemmer data i vores database, så kommer det direkte fra brugerne i den rå form. Den data vi modtager består derfor af muligt plettet ”Source”. Derfor pakker vi vores data ind i PS når vi skal ”Sink” den ind i vores database. Vi kunne godt ”Sanatize” den for specieltegn først, men dette kan resultere i at vi dobbelt ”Sanatizer” vores data, og derved nulstiller ”Sanitation” af indholdet. Af denne grund, er den data vi gemme i vores database stadig plettet (Tainted), da vi vælger at gemme den i sin rå form.

Når vi henter vores data igen, er det stadig en ”Source” som kan være plettet. Dette skyldes bl.a., at vi introducerer udefrakommende data ind i vores kodemiljø. Herfra sender vi den videre til forrige afsnit hvor vores hentet data bliver ”Sanatized”, før den bliver ”Sink” og vist til brugeren.

XSS:
For at øge sikkerheden vedr. ”Cross Site Scripting”[11] (XSS), så har vi gjort yderligere foranstaltninger end blot at benytte os af ”Taint Analyse” og gemme indhold vha. af PS. Vi har konfigureret vores NginX[12] til at forhindre XSS. Vi har også gjort os overvejelser om hvilken request data der skal sendes med retur i response’et. På den måde har vi forsøgt at eliminere brugen af XSS på vores hjemmeside. Her tænker vi især på de designvalg som vi har taget mht. rettighederne. Vi har nemlig opbygget hjemmesiden vha. ”Role Based Access Control”[13] (RBAC). Dette betyder, at det er den givne rolle som afgør hvilken sider og funktioner den pågældende bruger har adgang til. Dette er en nem måde at indføre ”Mandatory Access Control”[14] (MAC) på, som har vist sig at være meget sikker, men ikke speciel fleksibel, da data har svært ved at blive delt, læst eller skrevet på tværs af sikkerhedsniveauer. En mere fleksible løsning vil være at benytte sig af ”Discretionary Access Control”[15] (DAC), da man kan sætte forskellige rettigheder til brugere med fx samme rolle. Det er meget tidskrævende og man skal have et godt overblik over alle de filer eller sider som skal indgå i den rettighedsmatrix som man skaber. Til vores hjemmeside var RBAC det mest fornuftige valg, da vi pt. ikke har mange brugergrupper eller medlemmer som skal kunne en masse forskellige ting. Men med tiden, vil en klar anbefaling være at benytte sig af DAC, da man undgår de restriktioner som følger med MAC i fx Bell-LaPadula[16] og Biba[17]. Her kan information strande mellem forskellige rettigheder, da sikkerhedsniveauerne kan begrænse hinandens handlingsmuligheder, plus at integriteten kan blive kompromitteret af bl.a. Bell-LaPadula, da man kan skrive til et højere sikkerhedsniveau, som fx kan medføre, at man inficerer virus i et system vha. en brugerkonto.

Test:
For at teste de overnævnte punkter, har vi benyttet os af XSS-angreb, samt brugt penetreringsværktøjer som SQLMAP, Hydra og Burp. Med den viden vi havde, lykkes det os ikke at gennemføre et XSS-angreb på hjemmesiden. Den ondsindede kode vi sendte til databasen, blev

[11] https://owasp.org/www-community/attacks/xss/
[12] https://www.upguard.com/blog/10-tips-for-securing-your-nginx-deployment
[13] https://en.wikipedia.org/wiki/Role-based_access_control
[14] https://en.wikipedia.org/wiki/Mandatory_access_control
[15] https://en.wikipedia.org/wiki/Discretionary_access_control
[16] https://en.wikipedia.org/wiki/Bell–LaPadula_model
[17] https://en.wikipedia.org/wiki/Biba_Model

vist i sin rå form på vores frontend. Her gjaldt det brugernavnet, da e-mailadressen bliver valideret i HTML formen.

Selve penetrationstesten som skulle injicere databasekode, lykkes heller ikke. Det eneste vi opnåede ved de forskellige angreb, var et response med en statuskode 200. Men vi fik ikke adgang til selve brugerprofilerne.

Adgang

reCAPTCHA:
Som udgangspunkt forventer alle at det er fysiske mennesker som tilgår deres hjemmeside. Men det bliver mere og mere normalt, at hacker bruger "Bots"[18] til at tilgår hjemmesider. Disse Bots er automatiserede processer som vi fx selv bruger når vi skal teste sikkerheden på vores hjemmeside. For at eliminere disse Bots i at sende HTML formler, har bl.a. Google udviklet reCAPTCHA[19], hvor en bruger af hjemmesiden bliver bedt om at verificere et spørgsmål i billeder, for at sikre at brugeren er en fysisk person. Selv om der findes scripts som kan forbipassere denne sikkerhed, følte vi at den var med til at skabe et øget sikkerhedslag. Det gode ved reCAPTCHA er også, at det er kendt blandt brugerne på mange forskellige hjemmesider. Derfor vil denne sikkerhedsmetode ikke virke som en irritation for slutbrugerne.

Session:
For at sikre en god brugervenlighed for brugerne, så er det vigtigt at en bruger ikke konstant skal logge ind, hver gang brugeren tilgår en ny side med samme sikkerhedsniveau. For at imødekomme dette, så benytter vi os af Session, som bliver sat når en bruger logger ind på hjemmesiden. En Session bliver gemt som en cookie på computeren og får et unikt fingeraftryk fra vores Tomcat server. Da hjemmesiden består af flere forskellige rettighedsniveauer, så har vi valgt at give en laveste rettighed en 30 min timeout og den højeste 5 min timeout. På den måde sikre vi, at brugerne automatisk logges ud hvis de er inaktive. Vi har også sikret vores cookie mod at blive brugt i et XSS-angreb, ved at inkl. den i vores response header[20]. Grunden til vi har gjort så meget ud af Session's er, at "Broken Access Control"[21] bliver betragtet som det vigtigste punkt jf. OWASP, når det kommer til online sikkerhed. Ved at stjæle eller ændre i en cookie, kan man potentielt set få adgang til et højere rettighedsniveau end det man har adgang til.

Kodeord, loginforsøg og to-faktor godkendelse:
For at sikre hjemmesiden, forsøgte vi at implementere flest mulige NIST standarder. Denne standard fortæller os som udviklere hvordan vi bl.a. skal håndtere kodeord når de skal oprettes eller gemmes i databasen. Ifølge NIST, så skal et kodeord være på mindst 8 karakter hvis den ikke bliver autogenereret. Dette mener vi dog er meget lavt, da vi både har brugerprofiler og administrationsprofiler. Derfor har vi valgt at holde os til de minimum 8 når det gælder brugerne, da de ikke kan tilgå kompromitterende data, men øget den til 14 når der kommer til en administrator.

[18] https://da.wikipedia.org/wiki/Bot
[19] https://www.google.com/recaptcha/about/
[20] https://owasp.org/www-community/HttpOnly#
[21] https://owasp.org/Top10/

Når vi gemmer adgangskoden i vores database, så hashes kodeordet vha. BCrypt, som kan ses jf. vores bilag. For at gøre kodeordet ekstra sikkert, så hasher vi kodeordet med både Salt & Pepper. Dette giver en hashet værdi, som er meget svær at finde frem til. Skulle det ske at en hacker får fat i det hashet kodeord, så kan kodeordet udelukkende gættes, såfremt hackeren benytter nøjagtigt samme salt + pepper som vi gør. For at øge sikkerheden heri, benytter vi os også af et ”Pepper”, som vi sætter som en miljøvariable på vores server. Den er derfor kun kendt af os, og er ukendt for alle andre som ikke har adgang til serveren. Denne ”Pepper” bruger vi i kombination med vores ”Salt”. Deraf “Salt & Pepper”.

Skulle det ske at en hacker få adgang til en brugers brugernavn og adgangskode, så har vi på baggrund af RBAC givet de højeste rettighedsniveauer to-faktor godkendelse vha. Google Authenticator, som også er en klar anbefaling ifølge NIST. På den måde sikre vi os, at der dannes en sikkerhedslag mere, som skal hjælpe os med at sikre, at det kun er de autoriserede personer som får adgang. Grunden til at vi kun bruger det på de højeste rettighedsniveauer er, at de laveste ikke har adgang til kompromitterende data. Så det er et designvalg vi har taget, som skal sikre en højre brugervenlighed og som er tilpasset trusselsscenariet på hjemmesiden. Der skal også nævnes, at grunden til vi ikke benytter os af to-faktor godkendelse vha. e-mail eller mobil er, at folk tit bruger samme koder til mange applikationer, samt at en stjålen tlf. kan logge ind på applikationerne, hvor to-faktor koden bliver modtaget.

For at øge sikkerheden yderligere, så har vi implementeret et sikkerhedstjek når administratorer skal rette indhold i databasen. Vi har indført to-faktor godkendelse ved sletning/ændring af data, for at sikre os, at sikkerhedsniveauet bliver forhøjet under disse scenarier. På den måde vil en tilfældig person, som evt. opdager en ulåst computer, ikke have mulighed for at slette/ændre tilfældigt data.

Ifølge OWASP, så er Injection også blandt de vigtigste sikkerhedsbrud man skal prioritere. Som i det forrige punkt, så kan en vellykket penetration give en ondsindet bruger adgang til sikker information. Da mange hacker benytter sig af Bots eller scripts, så er et statisk antal loginforsøg en god måde at bekæmpe hackernes kontinuerlige loginforsøg på. Vi kender det fra vores telefoner eller dankort, hvor x antal forsøg låser for yderligere forsøg. Det samme gælder vores server. Her har vi gennemført en masse foranstaltninger, for at sikre vores backend. For det første, så giver vi brugerne 3 forsøg til at logge ind. Sker dette ikke, bliver IP-adressen låst i 24 timer. På den måde gør vi det svært at benytte ”Brute Force” som penetrationsmetode. Men vi har også indført to-faktor godkendelse på vores server, deaktiveret fjernadgang som ”ROOT”, samt fjernet ”password prompt” for at øge sikkerheden på vores backend server.

Blacklist og Proxy:
Et af de sværeste punkter for os har været at blackliste IP-adresser fra at logge ind på frontend. Dette skyldes, at det har været svært for os at teste, samt fremskaffe viden herom. Problemet opstår hvis en bruger taster sin kode forkert flere gange, så vil vi nemt kunne se brugerens IP-adresse. Men hvis der sidder andre på denne IP-adresse, så bliver de også blacklistet. Af denne grund har valgt at logge brugernavn, IP-adressen og tid. Man kan selvfølgelig diskutere om flere ville sidde på samme IP-adresse, og hvis de gør, om det så ikke er en kollektion af hackers. Men vi valgte denne fremgangsmåde. Derfor valgte vi også at køre alt vores trafik gennem en Proxy som

hedder Cloudflare[22]. Det gode ved denne løsning er, at Cloudflare kender til en masse ondsindede IP-adresser og har deres egen blacklist som vi får glæde af. Den sikrer også at trafikken kører via SSL og derved er sikker. Minusset er, at vi overgiver noget suverænitet til en 3. part, som vi skal stole på når det kommer til sikkerhed.

Redirect til SSL:
Som en forlængelse af forrige afsnit, så sikre vi vores SSL-forbindelse vha. Cloudflare. Men hele vores backend er også sat op på en sådan måde, at NginX lytter efter den krypterede forbindelse på port 80 og sender den videre til port 443. Ved at benytte SSL eller TSL[23], så sikre vi os at indholdet som bliver sendt er krypteret og ikke kan læses i tekst. Havde vi ikke opsat dette på vores server, så ville MITM angreb kunne læse alle vores POST data som rå tekst. Ved at sikre, at forbindelsen altid bliver sendt som SSL, så vil en hacker ikke kunne opsnappe indholdet i de sendte "packets"[24] vha. fx Wireshark.

GDPR:
For at overholde persondataloven[25], så har vi udviklet systemet på den sådan måde, at vi kun efterspørger den nødvendige persondata der skal til, for at oprette en bruger. På selve hjemmesiden vil en administrator kun få adgang til de nødvendige informationer, som skal bruges for at løse den pågældende opgave. Vi vil derfor fx aldrig efterspørge cpr. nr. når vi vil oprette en bruger, og en administrator vil aldrig få adgang til brugernes adgangskode. På den måde sikre vi os at vi ikke deler persondata eller følsomme persondata.

For at sikre en kontinuerlig opretholdelse af persondataloven, så gemmer vi et timestamp hver gang en bruger logger ind. På den måde kan vi kontrollere om data er mere end 6 måneder gammelt. Da vi dagligt eksekverer et "Stored Procedure"[26] vha. MySQL event handler, så sletter vi forældet brugerdata og deaktivere kontoerne som er berørt. Dette er selvfølgelig kun noget vi behøver at gøre, da vi ikke beder brugerne om samtykke til at gemme deres brugerdata længere end de 6 måneder.

Test:
I denne kategori prøvede vi at teste systemet med forskellige værktøjer. Men vi forsøgte også at køre scripts, samt at teste tænkte scenarier. Hvis vi starter med vores loginforsøg på vores server, så forsøgte vi kun dette én gang. Dette skyldes, at vores IP-adresse bliver blokeret, og vi ville derfor ikke have mulighed for at logge ind mere, havde vi kørt testen hjemmefra. Derfor tog vi ud et offentligt sted, og testen blokerede vores IP-adresse som forventet.

Vores NIST standard testede vi manuelt, samt i en test i vores projekt. Faktisk viste det os, at denne standard var så svær at opfylde, at vi tit havde problemer med at finde på kodeord som vi efterfølgende kunne huske. Selv autogenerede kodeord blev tit afvist af vores system, da de ikke overholdte standarden. Det gav derfor mere og mere mening, at et kodeord skal være langt, uden

[22] https://www.cloudflare.com
[23] https://da.wikipedia.org/wiki/Transport_Layer_Security
[24] https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/
[25] https://www.retsinformation.dk/eli/lta/2000/429
[26] https://searchoracle.techtarget.com/definition/stored-procedure

at opfylde specifikke krav til dets opbygning. Dette var bl.a. det vi blev undervist i, og derfor lavede vi også en ”light” version af vores NIST standard.

Da det kom til at teste systemet vha. ”Brute Force”, deaktiverede vi vores Proxy hos Cloudeflare, da vi ikke ønskede at blive blacklistet fra deres servers. Derefter kørte vi både Burp og Hydra. Begge tests blev gennemført med statuskode 200 på alle tests. Overraskende kunne vi ikke se en effekt af vores reCAPTCHA da vi forsøgte at gennemføre vores penetrationstests. Dette kunne vi ikke helt afgøre om var positivt eller negativt, da vi både kunne se fordele i, at en hacker ikke får besked om at penetreringer fejler. Men på den anden side, så fik vi heller ikke bekræftet at det virkede, medmindre vi forsøgte at logge ind fysisk.

Vi prøvede også flere forsøg på at kopierer vores session ID fra forskellige brugere og benytte den i forskellige browsers eller på brugere med forskellige rettigheder. Alle forsøg viste sig at være mislykket, som i dette tilfælde var en god ting. Det samme gjaldt vores MITM angreb som også viste sig ikke at give resultater, da hjemmesiden gennemtvang en SSL-forbindelse til vores server.

For at teste vores ” Stored Procedure”, oprettede vi nogle brugerprofiler med gamle oprettelsesdatoer. Vores test gik ud på, om den forældede data blev slette, samt om man kunne logge ind på profilerne igen. Her brugte vi også Burp, for at se om vi kunne identificere forskellige fejl. Det viste sig at alle forældede data blev slette og vi kunne ikke se forskel på når vi kørte Burp mod disse brugerprofiler. Forsøgte vi at logge ind manuelt, så havde vi heller ikke adgang.

Da det kom til vores Social Engineering[27], så blev vores test lidt teoretisk. Dette skyldes, at selve angrebet går ud på at snyde sig adgang via sociale relationer. Dette var jo ikke muligt for os at teste. Derfor valgte vi at kigge på den case, at en hacker havde fået brugerdata fra en medarbejder via snyd, og nu ville forsøge at logge ind som en administrator. Her kunne vi så konstaterer, at den to-faktor godkendelse som vi havde sat op, virkede som det skulle og gav ikke brugeren adgang uden den ekstra godkendelse.

Status

Statuskode og fejlbesked:
En af de ting vi oftest kigger efter når vi skal udføre forskellige penetrationsværktøjer, er den statuskode som systemet returnerer, når vi fx forsøger at ”Brute Force” os vej ind i et system. Hvis statuskoden var alt andet end 200, så vidste vi, at brugernavnet og kodeordet ikke var blevet godkendt og der er derfor ikke adgang til systemet. Af denne grund har vi ændret vores håndtering af statuskoden, så den altid returnerer statuskode 200. På den måde vil en hacker have svært ved at se hvilke brugernavne og kodeord som giver adgang til systemet.

Mht. fejlbeskederne så valgte vi at opsnappe alle de beskeder som gav brugerne information om systemet. Fx fik brugerne besked om duplikator, når samme brugernavn blev oprettet. Men vi valgte ikke at fjerne advarselsbeskederne i vores loginproces. Dette valgte vi, da det både giver en god brugeroplevelse, selv om det mindsker sikkerheden. Men også fordi at et login trigger en redirect, som penetrationsværktøjer vil kunne opsnappe. Vi kiggede på at lave billeder i stedet for

[27] https://en.wikipedia.org/wiki/Social_engineering_(security)

fejlbeskeder, men da billederne kun bliver vist ved fejl login, så vil dette også være noget en hacker vil kunne fokusere på. Så vi endte med at tager den brugervenlige vej, da vi følte at vi havde sikret systemet på alle andre parametre.

Logning:
For at sikre os at vi som udviklere eller fremtidige administratorer har fuldt overblik over loginprocessen, valgte vi at logge alle loginforsøg med så meget information som muligt jf. vores bilag. På den måde kan vi følge med i hvem der logger ind på hjemmesiden, samt hvilken rettigheder de har. Skulle der ske en penetrering, så kan vi via tidspunktet finde frem til den konto som er blevet kompromitteret og ændre eller slette den.

Test:
For at teste denne kategori, benyttede vi os primært af Burp for at kunne identificere de statuskoder og fejlbeskeder som hjemmesiden sender og viser. Vores test viste, at hjemmesiden fungerede efter hensigten og ikke gav forkerte statuskoder, da vi på intet tidspunkt fik andre statuskoder end 200. Mht. fejlbeskederne, så loggede vi alle fejlforsøg i vores log. Vi gennemgik derefter loggen på vores server og kunne se den loggede information. Via vores logs kunne vi få et hurtigt overblik over de fejlforsøg der var logget. Vi prøvede både med kendte og ukendte brugernavne. Vi forsøgte også at give Burp de rette kodeord til systemet for at teste, at godkendte forsøg også blev logget på vores server.

Videreudvikling

Da vi nåede alle vores mål, satte vi os ned for at finde ud af, hvad et evt. næste skridt ville være i en udviklingsproces. Vi kom frem til følgende:

- Implementering af et bedre kodeordssystem.
- Bedre håndtering af fejlbeskeder, så de bliver mere usynlige for penetrationsangreb.
- Videreudvikling af vores server blacklist, så den ikke blacklister IP-adressen, men evt. gør det ud fra parametre, samt at vi kan whiteliste IP-adresserne igen efter behov.
- Udvikle "Discretionary Access Control"[28] (DAC) rettigheder til de forskellige brugere, så fx administratorerne kan have forskellige rettighedsniveauer.
- Opdeling af back- og frontend på forskellige servere, for at øge sikkerheden.

Konklusion

Da vi startede vores projekt, var vi meget ambitiøse mht. hvad vi gerne ville opnå indenfor den givne tid og ramme. Derfor valgte vi fra starten af at presse rammerne så meget som muligt, så vi kunne teste vores loginproces op imod så meget af den teori og praksis som vi havde lært gennem dette semester. Vi fandt hurtigt ud af, at en sikker loginproces er et meget opfattende projekt. Vi endte derfor med at bruge det meste af det lærte pensum.

Vores mål var at skabe en sikker loginproces, som sammenholdt med det lærte pensum, ikke kunne hackes eller penetreres. På baggrund af dette mål lavede vi en delvis risikoanalyse som fremgår af vores bilag. I denne analyse var RB2, RB5, RB6, RB9, RB11, RB12 og RB13 de risikoer,

[28] https://en.wikipedia.org/wiki/Discretionary_access_control

som omfattede loginprocessen. Ud fra vores risikovurdering kan vi se, at vi bl.a. vurderer RB12 til at være ekstrem, da den er nem at udføre, men også har en meget høj risiko for vores hjemmeside. Målet med projektet har derfor været at presse risikovurderingen ned i de grønne felter via de tiltag vi har implementeret på serveren og hjemmesiden.

Under projektet har vi beskrevet hvordan vi har øget sikkerheden på vores server vha. blacklisting af IP-adresser, brug af SSL og logning af events. På selve hjemmesiden og på vores frontend har vi implementeret input- og kodeordskontrol, brug af reCAPTCHA og to-faktor godkendelse, samt håndtering af statuskoder og fejlbeskeder. Systemets brugere er opbygget vha. RBAC, og vi kontrollerer løbende brugerprofilerne pga. GDPR vha. CRON-job. Vi har forsøgt at skabe et så sikkert og penetreringsdygtigt system som muligt, som kan modstå de penetreringsangreb som vi er blevet undervist i, og som får os til at konkludere, at vores nuværende risikovurdering ser ud som følgende.

Risikovurdering				
Risiko: Brugerdata (RB)				
Sværhedsgrad	**Nem**	RB10	RB1	
	Mellem			RB3, RB4, RB7
	Svær	RB2, RB5, RB6, RB8, RB11, RB14	RB9, RB12, RB13	
		Lav	**Mellem**	**Høj/Ekstrem**
		Risiko		

Som det fremgår af den nye risikovurdering, så har vi fået ændre mange af vurderingerne, så vi ikke længere har nogle som er ekstreme mht. loginprocessen. Havde dette været en rigtig hjemmeside, så skulle vi selvfølgelig arbejde på at lave en fyldestgørende risikoanalyse og reducere alle risikovurderingerne. Men ser vi på resultatet sammenholdt med de krav som vi havde for projektet, så kan vi konkludere, at vi har formået at skabe et projekt, der ud fra vores faglige kompetencer indenfor sikkerhed, kan vurderes til at være så sikkert som muligt, selv om intet er 100% sikkert inden for IT-sikkerhed. Vi håber dog, at dette projekt har givet den fornødne indsigt i vores planlægning og proces, samt hvilken tanker og overvejelser vi havde undervejs.

Bilag

Risikomodel					
Aktiver		**Kontrol**	**Motivation**	**Identificerbare modstander/angriber**	**Uidentificerbare modstander/angriber**
Digital	**Fysisk**				
Brugerdata	Server	Firewall	Informationsindsamling	Uvidende medarbejdere	Medarbejdere: spionage & leaks
Database	Computere	HTTPS, SSL/TLS	Ransomware	Vulnerability scanners	Script kiddies
Virksomhedsdata	Medarbejdere	SSH	Virksomhedsspionage	Web crawlers	Hackere og Cyberkriminalitet
OS	Kunder	Kryptering af bl.a. kodeord	Finansielle informationer	Backdoors	Eks-partnere
Applikation	Netværk	To-faktor godkendelse	IT-infrastruktur	Virus	Konkurrerende virksomheder
Kode Biblioteker	Endpoints	NIST	Social Engineering	Spyware	Terrorister
GitHub	Serverrum	Adgangskontrol		Phishing	Venlige og fjendske nationer
HR-data	Andet hardware	reCAPTCHA		MITM	Aktivister
Koden		Data rettigheder		Malware	Naturen
Salgsdata		Opdatering af hardware og software		DDoS angreb	
Klientdata		Antivirus		SQL injection	
Marketingdata		Netværksmonitorering og logning		XSS	
		Proxy		Brute Force	
		Router redirect		Broken Access Control	
		Black/whitelist		Bugs	
		GDPR			

Risikoanalyse						
Risiko: Brugerdata (RB)		Sandsynlighed	Konsekvens	Samlet risiko	Risikoniveau	Beskyttelse
		(1, 3 el. 5)	(1, 3 el. 5)	(1-25)		
RB1	Medarbejder kommer ved en fejl til at dele brugerdata fra hjemmesiden	3	3	9	Mellem	Access control management File management
RB2	Medarbejder kommer ved en fejl til at dele brugernavn og adgangskode	1	3	3	Lav	2 faktor godkendelse
RB3	Web crawlers scanner hjemmesiden og opsnapper offentlig eller ikke beskyttet brugerdata	5	3	15	Høj	Access control management File management
RB4	Spyware opsnapper beskyttet brugerdata	3	5	15	Høj	Opdatering af software og hardware Antivirus og scan
RB5	Medarbejder videresender brugerdata pga. en phishing-mail	1	3	3	Lav	2 faktor godkendelse Access control management File management
RB6	Brugerdata bliver opsnappet via MITM angreb	1	5	5	Lav	2 faktor godkendelse HTTPS, LLS/TLS (Wireshark)
RB7	Brugerdata bliver slettet eller kompromitteret pga. Malware	3	5	15	Høj	Opdatering af software og hardware Antivirus og scan (Wireshark)
RB8	Brugerdata bliver blokeret og kan ikke indhentes pga. DDoS angreb	1	3	3	Lav	Network monitoring Router redirect Black/whitelist

RB9	Brugerdata bliver indhentet via SQL, XSS eller Brute Force angreb	5	5	25	Ekstrem	HTTPS, SSL/TLS Password encryption 2 factor authentication Password length and complexity reCAPTCHA Network monitoring and logging
RB10	Medarbejder udleverer brugerdata med vilje	1	5	5	Lav	Access control management
RB11	Personer som er nybegyndere mht. hacking, får adgang til brugerdata	1	5	5	Lav	HTTPS, SSL/TLS Password encryption 2 factor authentication Password length and complexity reCAPTCHA Network monitoring and logging
RB12	Personer som er rutineret i at hacke, får adgang til brugerdata og kan bruge informationen til bl.a. kriminalitet, afpresning osv.	3	5	15	Høj	HTTPS, SSL/TLS Password encryption 2 factor authentication Password length and complexity reCAPTCHA Network monitoring and logging

RB13	Gamle medarbejdere eller partnere får adgang til systemet og brugerdata	3	3	9	Mellem	Password encryption 2 factor authentication Network monitoring and logging GDPR (CRON Job)
RB14	Konkurrerende virksomheder får adgang til brugerdata	1	5	5	Lav	HTTPS, SSL/TLS Password encryption 2 factor authentication Password length and complexity reCAPTCHA Network monitoring and logging

Risikovurdering: Samlet risiko = Sandsynlighed * Konsekvens, Lav: 1-5 Mellem: 6-11 Høj: 12-19 Meget Høj: 20-25

Risikovurdering				
Risiko: Brugerdata (RB)				
Sværhedsgrad	**Nem**	RB2, RB5, RB6, RB10	RB1, RB13	RB12
	Mellem			RB3, RB4, RB7, RB9
	Svær	RB8, RB11, RB14		
		Lav	**Mellem**	**Høj/Ekstrem**
		Risiko		

User tabel

Oversigt over hashed kodeord. Brugere slettes aldrig men sættes til (active = true || false), og seneste login timestamp gemmes.

id	name	email	role	password	totp	active	last_login
12	Emil admin	admin@emil.dk	ADMIN	$2a$10$09clka…	KI5DMEWILL4D…	1	2021-11-16 22:49:06
13	emil user	user@emil.dk	USER	$2a$10$09clka…	NULL	1	2021-11-16 10:38:44
14	arik admin	admin@arik.dk	ADMIN	$2a$10$L/QWr…	OG4EQRL23WO…	1	2021-11-17 07:54:52
15	Arik user	user@arik.dk	USER	$2a$10$WlkBB…	NULL	1	2021-11-17 07:53:21
16	Removed by GDPR	removed@email.com	USER			0	2021-04-04 23:37:45
17	Eksamen user	user@eksamen.dk	USER	$2a$10$h78fR…	NULL	1	2021-11-15 20:33:27
18	Eksamen admin	admin@eksamen.dk	ADMIN	$2a$10$0Zi6Q…	NULL	1	2021-11-16 14:05:31
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Loginforsøg tabel

Oversigt over de forskellige loginforsøg med tid, succes, bruger id og IP-adresse.

id	user_id	success	ip	timestamp
75	12	1	5.179....	2021-11-16 15:10:20
76	12	0	5.179....	2021-11-16 15:16:09
77	13	0	5.179....	2021-11-16 15:16:23
78	12	1	5.179....	2021-11-16 15:16:33
79	12	1	80.210…	2021-11-16 18:41:02
80	12	1	80.210…	2021-11-16 18:47:04
81	12	1	80.210…	2021-11-16 19:48:55
82	12	1	80.210…	2021-11-16 19:54:31
83	12	1	80.210…	2021-11-16 20:00:32
84	12	1	80.210…	2021-11-16 20:07:09
85	12	1	80.210…	2021-11-16 20:11:18
86	12	1	80.210…	2021-11-16 20:13:15
87	12	1	80.210…	2021-11-16 20:17:50
88	12	1	80.210…	2021-11-16 20:27:45
89	12	1	80.210…	2021-11-16 20:34:25
90	12	1	80.210…	2021-11-16 22:49:06
91	15	1	185.15…	2021-11-17 07:53:21
92	14	1	185.15…	2021-11-17 07:54:52
93	15	0	185.15…	2021-11-17 07:56:28
94	-1	0	5.179....	2021-11-19 10:48:36
95	-1	0	2.106....	2021-11-22 12:34:06