

Datasikkerhet i utvikling og drift

Gruppe 21 - Dokumentasjon

- Frida Unneberg
- Sander Gunvaldjord
- Lars Martin Randem
- Fredrik Gjemmestad
- Alexander Vesterhus Lyon

INTRODUKSJON	2
Prosjektutviklingen	2
SIKKERHETSKRAV	3
Liste over sikkerhetskrav	3
ABUSE CASES	4
Abuse cases	4
RISIKOANALYSE AV ARKITEKTUREN	5
De ulike delene av systemet	6
Sannsynlighet for ulike angrep	8
RISK MANAGEMENT FRAMEWORK	9
Tabeller	9
RISK BASED SECURITY TEST	11
Unit tests	15
CODE REVIEW	15
Code Factor	15
Codacy	15
Manuelt	16
Sammendrag av Code Reviews	17
UTVIKLINGSPROSESS	17
Steg 1 - Utvikling av prosjektet	17
Steg 2a - Angrepsperioden	17
Steg 3 - Utbedringer av systemet	17
ARBEIDSFORDELING	21
Før 15. mars:	21
Etter 15. mars:	21

INTRODUKSJON

Dette er prosjektdokumentasjonen som er en del av oppgaven. Den inneholder et par avsnitt som beskriver prosjektutviklingen, den inneholder sikkerhetskrav, abusecases, risikoanalyse av arkitekturen på nettsiden vi har lagd, et risk management framework, en risk based security test som baserer seg på sikkerhetskravene og abusecases som ble lagd, utviklingsprosessen og hvilke sikkerhetstiltak vi implementerte, code review og tilslutt arbeidsfordelingen i gruppa samt en oppsummering av prosjektet.

Prosjektoppgaven går ut på å lage en nettside for delvis anonyme tilbakemeldinger til en foreleser i emner ved høyskolen. Systemet skal kunne registrere studenter som kan sende melding angående ønsket emne, men forbli anonym. Forelesere skal kunne registrere seg og angi emner foreleseren underviser i, i tillegg kunne lese og svare på meldinger fra studentene. Det skal være en eller flere administratorer som skal godkjenne forelesere som registrerer seg. En gjestebruker som skal kunne se meldinger for et valgt emne, men kun de emnene man kjenner en firesifret kode til, og kunne rapportere og kommentere på meldinger til studenter. I tillegg til nettsiden skal det være en app som kan lastes ned på siden, som er mobilversjonen av nettsiden. Nettsiden skal ligge på en server som har en database med de registrerte brukerne i systemet.

Prosjektutviklingen

Under utviklingsfasen av prosjektet tok vi i bruk WAMPserver lokalt på maskinen for å teste sammenkoblingen mellom database, API og PHP/HTML-koden. For testing opprettet vi en database med MySQL Workbench og hadde den på en WAMPserver. Når vi skulle benytte serveren som vi fikk tildelt installerte vi MySQL og PHP på serveren og opprettet databasen på selve serveren. For å koble sammen nettsiden og Android applikasjonen med serveren brukte vi PHP-script for å sende og hente ut informasjon til databasen. Disse filene ligger på serveren.

Android applikasjonen ble lagd med Java i Android Studio og den kobler seg til serveren med URL-kall til PHP-script. Disse PHP-scriptene sender tilbake JSON formaterte tekststrenger som blir brukt for å hente ut data for å kunne presentere det.

Gruppa har brukt kunnskaper fra tidligere fag bla. objekt orientert programmering og databaser. En på gruppe hadde forkunnskaper om Linux fra før og en annen hadde kunnskaper om PHP. Vi har også benyttet oss av Google for å finne løsninger.

SIKKERHETSKRAV

Liste over sikkerhetskrav

1. Software skal holdes oppdatert.

- Software som apache og annet software som kjører på serveren, skal holdes oppdatert.
- Påse at nyere versjoner fungerer med resten av systemene. F.Eks Graylog og Elasticsearch, Graylog støtter ikke nyeste versjon av Elasticsearch. Se på hva slags nye sikkerhetshull.

2. Tiltak mot DoS/DDoS:

- Dersom bruker sender for mange spørringer i løpet av et bestemt tidsvindu skal brukeren blokkeres for en viss tid.

3. Tiltak mot SQL injection:

- Alle spørringer mot databasen skal være utformet slik at SQL injection ikke skal være mulig.
- Bruke prepared statements i spørringene i koden.
- I tillegg ha inputvalidering som ikke tillater tegn som kan brukes til å skrive inn spørringer.

4. Database sikkerhet:

- Alle brukere på databasen skal ha et eget passord.
- Brukere med forskjellige roller skal ha forskjellige brukerrettigheter til databasen basert brukeren.

5. Tiltak mot XSS:

- Det skal ikke være mulig for en bruker å skrive kode i noen form for inputfelt (kommentarfelt, innlogging osv).
- En bruker skal ikke kunne endre programmet kjøremåte ved å redigere url parametere.
- Apache skal ikke tillate TRACE requests

6. Error message:

- Eventuelle error messages som vises for brukeren skal ikke inneholde for mye informasjon.

7. Passord:

- Det skal være passordkrav til brukere, f.eks passordlengde, store og små bokstaver, og minst et spesialtegn.
- Passord skal hashes når de blir langt til eller oppdatert i databasen.

8. Validering skal foregå både på server side og client side:

- Skal være noen form for inputvalidering i HTML kode samt i PHP kode.
- Dersom bruker skulle klare å redigere HTML koden skal PHP validering plukke det opp isteden.

9. Filopplastninger:

- Det skal være en max grense på hvor stor filen som skal lastes opp kan være.
- Det skal være begrensninger på hvilken filtype som bruker kan laste opp på nettstedet, f.eks kun .png, .jpg.
- Skal ikke være mulig å laste opp kjørbare filer, eller filer som inneholder kode.
- Ideelt skal bare autoriserte brukere ha mulighet til å laste opp filer på nettstedet/app.

10. Nettstedet skal bruke HTTPS.

- Nettstedet skal ha kryptert nettrafikk over HTTPS

11. Logging:

- Det skal være logging på all brukeraktivitet på nettstedet.
- Skal være enkelt å oppdage eventuelle angrep forsøk.

12. Session autentisering:

- Nettstedet og appen skal bruke rollebasert tilgangskontroll.
- Når bruker logger inn på nettstedet blir en session id tildelt basert på om bruker er student, foreleser eller admin.
- Dette vil begrense tilgang på nettsidene og funksjonalitet basert på hvilken rolle man er innlogget som.

ABUSE CASES

Abuse case er en modell for sikkerhetskrav der vi har valgt å omtale actor som bruker, da en bruker kan "abuse"/misbruke systemet uten å ha vonde intensjoner. Abuse Cases beskriver interaksjonen mellom en eller flere personer og systemet der utfalle av handlingene fører til skade på systemet, de som drifter eller brukere av systemet.

Abuse cases

1. Filopplastning

Scenario: Angriperen laster opp en fil på nettstedet

- Angriper laster opp veldig stor fil, fyller lagringsplass til server.
- Angriper laster opp fil som inneholder skadevare, eller ondsinnet kode.
- Angriperen laster opp ZIP bomber eller XML bomber.
- Angriperen skal overskrive allerede eksisterende filer på nettstedet/app.

2. XSS

Scenario: Angriperen klarer å skrive/legge inn skadelig kode på nettstedet/appen.

- i. Angriperen legger XSS code på sin egen brukerprofil. Denne koden vil da kunne be nettleser til andre brukere å utføre script i javascript.
- ii. Angriperen skriver JS kode i et kommentarfeltet, koden blir da lagret på nettsiden og kjørt hver gang noen laster inn siden.
- ii. Angriper sender inn <script> tag som redirecter siden, i feks en kommentar. Alle som ser denne kommentaren vil da bli redirected vekk fra siden.

3. SSH tilgang

Scenario: Angriperen klarer å få tilgang til SSH.

- i. Angriper bruker bruteforce metoder til å koble seg opp med SSH til serveren. Oppnår full tilgang til operativsystemet.

4. Fysisk tilgang til server

Scenario: Angriperen klarer å få fysisk tilgang til noen datamaskin eller serveren.

- i. Angriperen får tilgang til servermaskinen, de kan da gjøre skade eller få full tilgang til maskinen.

5. Brute Force:

Scenario: Angriperen anvender et brute force angrep mot nettstedet/app.

- i. Angriper finner passordet til en eller flere brukere, kan derfor logge inn som den brukeren og benytte seg av funksjonalitet tildelt brukerens rolle.
- ii. Angriper finner PIN kodene til emner, kan da få opp informasjon om emnet og legge inn henvendelser uten å måtte være innlogget.

6. SQL Injection

Scenario: Angriper klarer å utføre en SQL-injection.

- i. Angriperen kan hente ut sensitiv informasjon om andre brukere fra databasen, som passord, adresser, e-postadresser osv.
- ii. Angriperen kan slette, endre eller sette inn informasjon i databasen.
- iii. Angriperen kan endre eller fjerne brukerrettigheter til databasen.
- iv. *Eksempel: Angriper endrer passord til adminbruker, kan så logge inn som admin og få tilgang til alle admin sider og admin funksjonaliteter.*

RISIKOANALYSE AV ARKITEKTUREN

I risikoanalysen vår, begynte vi å sjekke systemet opp mot abuse casene våres og gikk igjennom alle funksjonene på nettsiden. I hver funksjon testet vi dem og diskuterte potensielle farer og angrep som kunne utnyttes underveis i prosessen. Vi har delt funksjonene i systemet opp og nevnt hvilke angrep hver av disse delene kunne være utsatt for, eksempler på hvordan disse angrepene kan bli utført og løsninger vi har gjort eller løsninger som kan/bør gjøres.

De ulike delene av systemet

1. Innlogging:

I innloggingsfunksjon er det flere angrep vi vil hindre blant annet Brute Force angrep, SQL-injection og XSS (Cross Site Scripting). Brute Force angrep kan bli utnyttet når det ikke er gjort noen sikkerhetstiltak mot hvor mange ganger det er mulig å logge inn. Dette lar hackere kjøre script som kan iterere gjennom passord-kombinasjoner. SQL-injection og XSS kan utnytte inputfelt dersom det ikke er gjort noen sikkerhetstiltak mot disse typer angrep. Dersom det ikke er gjort noen tiltak kan de sende inn SQL-spørringer eller Javascript inn til serveren. Noe som man også vil unngå er session hijacking. Dette kan f.eks skje med social engineering. Svindleren lurer en bruker til å oppgi sin session-ID til svindleren og dermed får svindleren tilgang til brukeren.

- Brute Force angrep kan hindres ved å begrense antall innloggingsforsøk innen en viss tidsramme. Et annet tiltak som hindrer brute force angrep er to-faktor autentisering. Dette innebærer at selv om man skriver riktig brukernavn og passord må brukeren identifiseres med en ekstra kode eller lignende.
- SQL-injection har vi forhindret med å ta i bruk inputvalidering i PHP ved bruk av regex og i tillegg brukt prepared statements.
- XSS kan også forhindres ved bruk av inputvalidering noe som vi tenkte på da vi implementerte inputvalidering for SQL-injections.
- For å unngå session hijacking kan man lagre brukerens session-ID i en database og deretter sjekke i koden om det er flere session-ID'er er i bruk på samme tid.

2. Registrering:

I prosedyren for registrering hadde vi tidligere ingen sjekk for hvor sterke passord brukerne lagde, men det er noe vi har implementert.

I registrerings funksjonen er det i hovedsak to typer angrep vi vil unngå: DDoS/DoS-angrep og XSS.

DDoS/DoS-angrep kan utnytte registreringsfunksjonen ved å registrere veldig mange brukere på kort tid, resultatet av dette er en database med mange falske brukere. XSS kan brukes til å sende med skadelig script i registreringsinformasjonen som blir sendt til databasen. Disse scriptene kan være skadevare med flere ulike funksjoner. I registreringen av forelesere er det også et krav til bildeopplasting. Dette kan utnyttes ved at en bruker laster opp en fil som inneholder skadelig programkode eller en fil som er for stor.

Det er også viktig å sikre sensitiv informasjon om brukeren som blir opprettet, slik som passord, for å forhindre det fra å komme på avveie dersom det skulle oppstå en datalekkasje.

- For å hindre DoS-angrep kan man begrense antall ganger man kan registrere seg innen en viss tidsperiode. Dette vil gjøre at scriptene som blir kjørt av DDoS/DoS angrepet blir lite effektive.
- XSS har vi stoppet ved å ta i bruk input validering av "skumle" tegn som kan brukes for å opprette skadelig script, som f.eks. script-tagger.
- For bildeopplastingen har vi sjekket at filen er et bilde, og sjekker størrelsen på filen. Hvis filen er for stor, kan det bety at filen inneholder skadelig kode, eller at den vil ta opp for mye plass. Vi endrer også navnet på alle bildefilene som blir lastet opp.
- Når en person oppretter en bruker med navn og passord, vil passordet hashes før det sendes til databasen og blir lagret der med hashkoden, så hvis noen skulle fått tilgang til informasjonen i databasen, vil ikke de se passordet i klartekst men som en passordhash.

3. Søke etter emner og legg til emner:

I emner ligger det en søkefunksjon dersom det skulle være nødvendig og søke etter et emne. Dette innebærer at den er utsatt for SQL-injections og XSS. Dette gjelder også funksjonen "legg til emner". I "emner" kan Hackeren legge inn en SQL-spørring som inneholder en "insert-setning" som kan legge til et skadelig script forkledd som et ekstra emne i databasen. I siden "legg til emner" er det mulig å opprette emner, det er derfor viktig å sørge for at ikke hvem som helst har tilgang til denne siden.

- For å sikre at ikke hvem som helst kan legge til emner så sjekker vi at brukeren som går til siden er en foreleser.
- For å forhindre dette har vi tatt i bruk inputvalidering for å stoppe injections.
- Vi har også tatt i bruk databaserettigheter, der vi har en database bruker som bare har rettigheter til å lese fra databasen og ikke mulighet til å endre eller slette noe.

4. Kommentarfelt:

I kommentarfeltene til de ulike emnene, så skal det ikke være mulig å kunne spamme med kommentarer, ikke sende inn farlig skadevare eller SQL-spørringer. Disse angrepstypene kan bli gjort med DoS-angrep, SQL-injection og XSS.

- Vi har hindret SQL-injection og XSS med inputvalidering og prepared statements.

- Denial of service angrep kan stoppes ved å begrense antall kommentarer som kan bli sendt inn av de ulike brukerne av systemet innenfor en viss tidsramme.

5. Bytte passord:

Ved bytting av passord er det viktig å sjekke at brukeren kun har muligheten til å endre sitt eget passord.

- Inputvalidering, for å forhindre SQL-injection som legger inn eller fjerner data fra databasen.
- Bruker må skrive inn det gamle passordet sitt, slik at man ikke har muligheten endre passord til andre brukere.

6. Godkjenne forelesere:

Det er kun administrator som skal ha tilgang til å autorisere forelesere. Derfor har vi gjort det slik at det bare er administrator som har tilgang til siden. Med inputvalidering i inputfelt, forhindrer vi at brukere utgir seg for å være administrator.

7. Server:

Vi vil vite hva som blir gjort på serveren for å kunne detektere potensielle angrep. Vi vil også hindre DoS-angrep på serveren.

- For å detektere potensielle farer, kan man loggføre det som blir gjort på nettsiden og serveren.
- Vi har brukt Graylog for å logge handlinger som brukere gjør på nettsiden. Dette hindrer ingen angrep, men kan advare mot potensielle angrepsfarer.
- Vi har brukt mod_evasive, som detekterer DoS-angrep og blokkerer IP-adressen til angriperen.

8. Database:

Vi vil ikke at hvem som helst skal kunne gjøre hva som helst i databasen. For å unngå det, har vi laget flere database brukere med forskjellige rettigheter. Vi har en bruker som kun har rettigheter til å hente data ut fra databasen, og en databasebruker som i tillegg har mulighet for å skrive til databasen. Disse brukerne brukes til forskjellige deler av systemet.

Sannsynlighet for ulike angrep

Etter implementasjonen av de nevnte sikkerhetstiltakene i systemet og de tiltakene som bør gjøres, vil dette gjøre serveren og nettsiden mye tryggere. Vi har gjort de sikkerhetstiltakene vi mener er viktigst og som har størst sannsynlighet for å bli utført i systemet vårt.

Disse sikkerhetstiltakene er Inputvalidering, sperre IP-adresser som utfører DoS-angrep eller bruteforce, hashing av passord i databasen for å sikre det fra datalekkasje, og en databasebruker som kun har leserettigheter som forhindrer modifisering i databasen.

RISK MANAGEMENT FRAMEWORK

Risk Management Framework er en prosess som blir brukt til å identifisere potensielle farer i en bedrift eller organisasjon. Det går ut på å finne ut hva som kan true businessen og hvor stor konsekvens, sannsynlighet og alvorlighetsgrad disse farene har. De blir omtalt som business risks, project risks eller product risks. Når man har funnet ut hvilke farer som finst, må man ned på det tekniske nivået, der man ser hvordan disse farene kan oppstå. Tilslutt ser man på sikkerhetstiltak og finner ut hvordan man kan stoppe de tekniske angrepene som kan lede til de overordnede farene bedriften er redd for skal skje. Vi har satt opp dette i tabeller med alvorlighetsgrad, så man ser hvilke farer som er viktigst å forhindre med tanke på hvilke konsekvenser de har og hvor sannsynlig det er at det skjer.

Tabeller

Tabell 1:

Tabellen viser tre forskjellige overordnede farer som kan oppstå i systemet vårt, med en beskrivelse av hva de er, hvordan de oppdages, sannsynlighet, konsekvensen og alvorlighetsgraden av faren.

ID	Navn	Beskrivelse	Risk Indicator	Sannsynlighet	Konsekvens	Konsekvens	Alvorlighetsgrad
BU-1	Datalekkasje	lekkasje av brukernavn og passord lagret i klartekst.	Brukere klager for de ikke får logget inn.	Høy	Sensitiv informasjon kommer på avveie.	Høy	Høy
BU-2	Nedetid	Feil som gjør systemet utilgjengelig.	Feilmeldinger og klager av brukere av systemet.	Lav	Brukerne får ikke tilgang til systemet og kan bli misfornøyde med siden.	Medium	Lav
BU-3	Innsending av farlig data	Utnytte inputfelt som kan brukes til å sende inn farlig data.	- Databaser, nettsider osv blir slettet.	Høy	Data kan bli fjernet, hente ut eller slettet.	Høy	Høy

Tabell 2:

Viser de tekniske farene som kan skje i systemet med, beskrivelse av hva de er, hvordan det oppdages, sannsynligheten for at det skjer og konsekvensen av det.

ID	Navn	Beskrivelse	Sannsynlighet	Konsekvens	Risk Indicator
TEK-1	Input-Injection	Innsending av data som er skadelig for nettsiden og/eller databasen. F.eks. innsending av SQL-kommandoer som ødelegger databasen, eller henter som logger bruker inn uten å skrive inn passord (SQL-injection), eller sende inn script som ødelegger nettsiden.	Høy	Høy	- Databasen er ødelagt. - Nettsider fungerer ikke pga. script som ødelegger.
TEK-2	Brute Force-angrep	Kjøre script eller prøve å logge inn så mange ganger de vil.	Høy	Medium	- Brukere får ikke logget inn på konto. - Misbruk av brukerkontoer - Unormal aktivitet
TEK-3	DoS-angrep	Oversvømmer nettstedet med trafikk.	Medium	Høy	Systemet gjøres utilgjengelig for bruk.
TEK-4	Session Hijacking	Tar i bruk noen andres sessionID ofte ved bruk av Social engineering	Lav	Lav	- to eller flere brukere med samme ID
TEK-5	Man In The Middle -Angrep (MITM)	At noen lytter til ukryptert data som blir sendt mellom serveren og en bruker.	Lav	Medium	Hvis angriperen endrer på data, ellers vanskelig

Tabell 3:

Viser de overordnede farene som kan forekomme i systemet, der hver enkelt fare har en eller flere tekniske angrep som er potensielle måter de overordnede farene vi vil unngå kan oppstå.

Business risk		Technical risk		Påvirkelse
BU-1	Datalekkasje	TEK-1	Input-injection	H
		TEK-5	MITM- angrep	M
BU-2	Nedetid	TEK-3	DoS-angrep	H
BU-3	Innsending av data	TEK-1	Input-injection	H
		TEK-5	MITM-angrep	M

Tabell 4:

Viser de tekniske angrepene som kan lede til farene vi vil unngå, med sikkerhetstiltak som kan hindre mange av de tekniske angrepene. De rangeres med "N/A, L, M og H" for hvor hjelpsomme tiltakene er. N/A vil si at de ikke har noen effekt.

Technical risk	Inputvalidering	Begrenset innloggingsforsøk	To-faktor autentisering	Begrense requests til server innenfor en tidsperiode	Logge ut brukere med lik session-ID	HTTPS
TEK-1	H	M	N/A	N/A	N/A	N/A
TEK-2	N/A	H	H	H	N/A	N/A
TEK-3	N/A	N/A	N/A	H	N/A	N/A
TEK-4	N/A	N/A	N/A	N/A	H	N/A
TEK-5	N/A	N/A	N/A	N/A	N/A	H

RISK BASED SECURITY TEST

Vi har valgt å teste sikkerheten manuelt.

Vi tester sikkerheten til nettstedet og appen basert på resultatene fra Risk Management Framework .

Vi prioriterer scenarioene som har størst sannsynlighet og størst konsekvens.

BU-1: Datalekkasje:

Vi har her gjort en test for å teste hashing av passord.

Student

Fyll inn informasjon

Navn

E-post

Studieretning

Årskull

Passord

[Registrer](#)

Er du allerede registrert? [Logg inn her!](#)

Registrerer en brukeren med navn 'Jens Jensen'.

Kan se hvordan brukeren er lagret under, utsnitt av databaseraden som hører til Jens Jensen.

```
| Jens Jensen | jens@jensemail.com | Informatikk | 2018 | $2y$10$BVHFukoluQByFHgv50fg701cfi08ituCuh1low/fGBrxkC0Pu.sXG |
```

I bildet over kan man se at den nye brukerens passord blir registrert i databasen i hashet format med Bcrypt.

Sikkerhetstest passert: Passordet ble ikke lagret i klartekst i databasen.

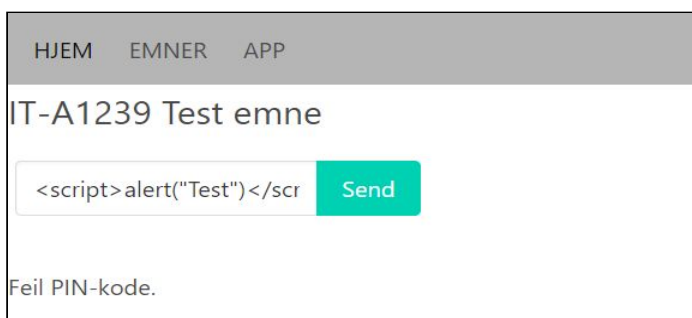
BU-1/BU-3 TEK-1: Input-injeksjon(XSS)

For å teste systemet mot XSS angrep skriver vi JavaScript kode i inputfelt på nettsted. På tre forskjellige nettsider kjører vi testkoden: `<script> alert("Test");</script>`.

(Input validering)TEST 1:

Søkefelt for PIN-kode.

Bildet under viser nettsiden etter at testkoden ble sendt.



HJEM EMNER APP

IT-A1239 Test emne

Send

Feil PIN-kode.

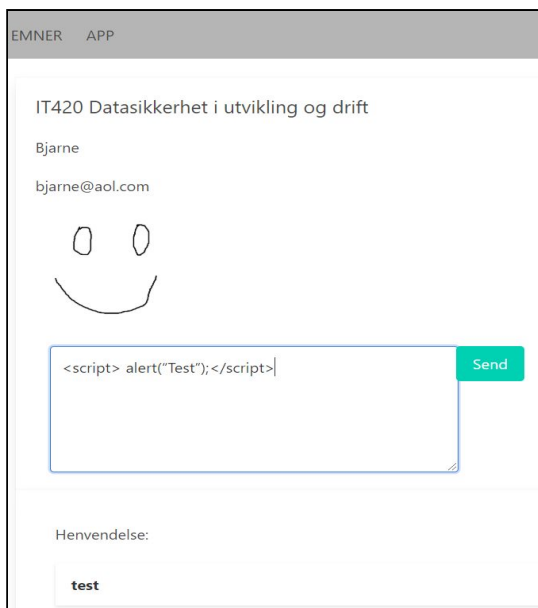
Resultat:

Nettsiden ga tilbakemeldingen "Feil PIN-kode", ingenting annet ble endret.

Testkoden hadde ingen effekt.

(Input validering)TEST 2:

Tekstfelt for posting av henvendelser knyttet til et emne, og kommentarer.



EMNER APP

IT420 Datasikkerhet i utvikling og drift

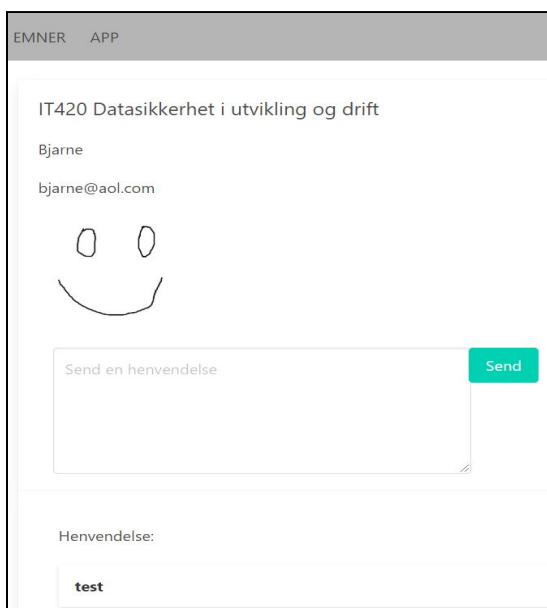
Bjarne

bjarne@aol.com

Send

Henvendelse:

test



EMNER APP

IT420 Datasikkerhet i utvikling og drift

Bjarne

bjarne@aol.com

Send

Henvendelse:

test

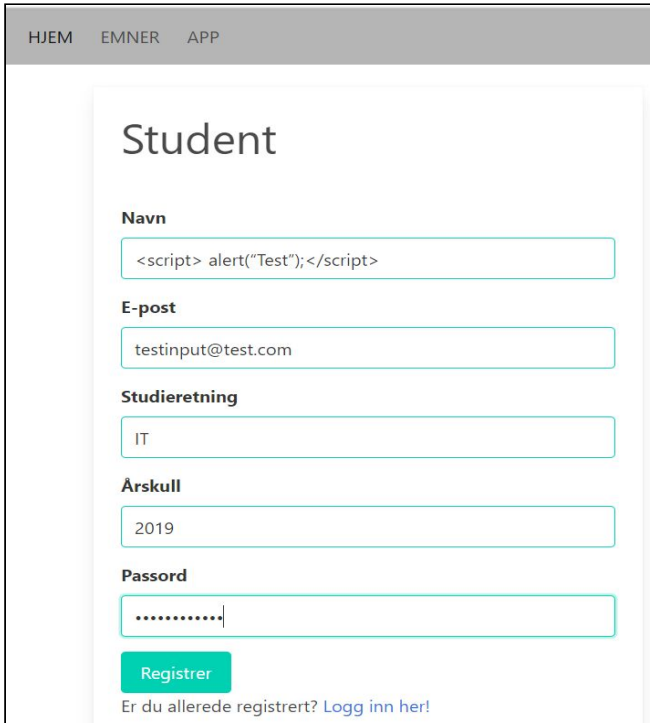
Bildet over til venstre viser testkoden i tekstfeltet for å poste en henvendelse.

Bildet over til høyre viser nettsiden etter at koden ble sendt.

Resultat:

Hverken henvendelsen eller kommentaren ble postet på nettsiden, testkoden hadde ingen effekt.

(Input validering)TEST 3:



Inputfelt for registrering av ny bruker.

Sørget for at de andre inputfeltene som ikke inneholder testkoden oppfyller kravene til inputvalideringen.

Resultat:

Brukeren ble ikke registrert og testkoden hadde ingen effekt.

Test passert: Testkoden hadde ingen effekt på noen av nettsidene.

BU-2 TEK-3: DoS/DDoS:

```
kali@kali:~$ nping --tcp-connect -rate=90000 -c 900000 -q 158.39.188.221

Starting Nping 0.7.80 ( https://nmap.org/nping ) at 2020-05-04 17:50 EDT
^CMax rtt: 2.008ms | Min rtt: 0.027ms | Avg rtt: 0.188ms
TCP connection attempts: 29721 | Successful connections: 15 | Failed: 29706 (99.95%)
Nping done: 1 IP address pinged in 9.54 seconds
```

Forklaring av bilde: Program som kjører i kali: nping for å pinge serveren. Serveren kjørte i dette tilfelle med Mod Evasive med 10 sekunders utkastelse etter 15 connections per sekund. Kjørte testen i 9.54 sek og kun 15 connections gikk igjennom.

Vi ble stoppet i å sende siden flere connections.
(99.95% blokkert, kun de 15 første kom igjennom)

Etter test: Setter opp utkastelses tid til 3600 sek(1 time)

Case: Tester om side kaster ut bruker som gjør for mange forespørsler til server.

Test passert, da testenheten ble nektet tilgang etter 15 forsøk.

BU-1 TEK-2: Brute Force:

Case: Teste systemet for brute force, tester om vi klarer å brute force passordet til SSH brukeren vår. Krav for passert test: Vi får ikke tak i SSH passordet.

Prøvd i lengre tid med brute force på SSH passordet med kjent brukernavn. Hadde ingen effekt. Passord anses som trygt (Tillegg: Kunne vært bedre med endring av port til SSH, slik at det er vanskeligere med brute force pga ukjent port.)

Test passert: Vi fikk ikke tak i passordet.

TEK-2/TEK-3: Logging:

Case:

Handlinger som blir foretatt på nettsiden, skal logges.

Vi prøver å logge inn med en admin bruker. Dette kommer opp i Graylog:

timestamp
2020-05-04 22:39:28 +00:00
Admin ble logget inn.

Utfyllende:

2020-05-04 22:39:28 +00:00 Admin ble logget inn.	
✉ 1e66eef0-8e58-11ea-a5b9-005056a9872c	
Timestamp 2020-05-04 22:39:28.124	ctxt Brukernavn admin@admin.no
Received by LogTilNettside on P a9ad6717 / datasikkerhet	facility sikkerhet
Stored in index graylog_0	level 6
Routed into streams • All messages	message Admin ble logget inn.
	source datasikkerhet
	timestamp 2020-05-04 22:39:28 +00:00
	user Gjest-5eb0743609ca8

Case passert, Innloggingen vises i loggesystemet.

Noen flere logg eksempler, her hvor innlogging og pin ikke ble gjort riktig: Greit å få ut, om noen tester mange nok pin(Bedriver angrep). Systemet skriver også ut om innlogging ble vellykket.

```
2020-05-04 20:34:44 +00:00
Bruker prøvde å oppgi PIN-kode for et emne. Noe gikk galt.

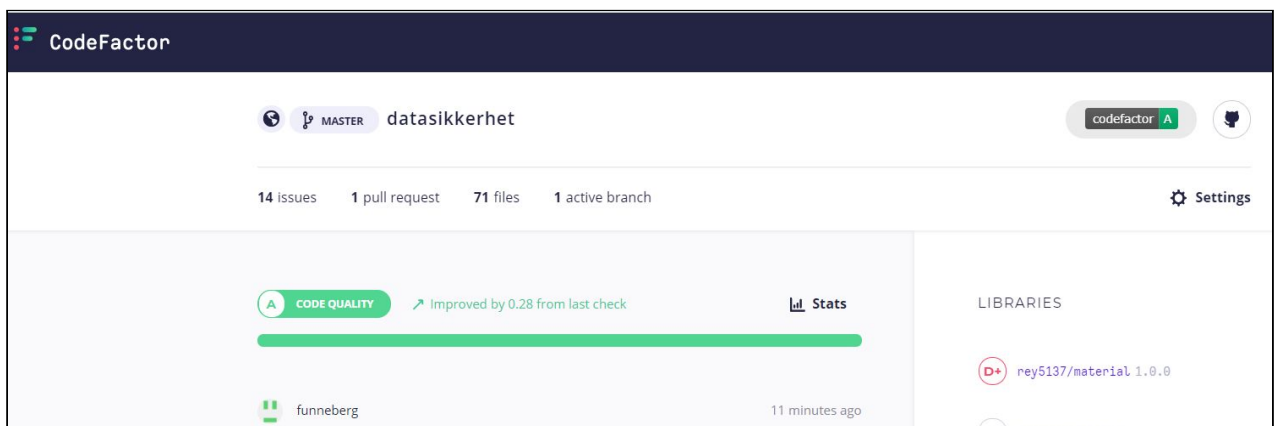
2020-05-04 20:32:28 +00:00
Bruker ble ikke logget inn. Innlogging mislykket.
```

Unit tests

Valgt å ikke implementere unit test i android applikasjonen, grunnet at den kommuniserer med REST-api'et som bruker samme logikk som nettstedet. Gjort manuelle tester.

CODE REVIEW

Code Factor



Kjørte en code review med et verktøy på GitHub som heter Code Factor. Den går gjennom prosjektets repository, og gir en rating for hvor god koden er. Første gangen vi kjørte koden gjennom Code Factor fikk koden en A- rating. Feilene som Code Factor fant var i stor grad i filer fra gamle versjoner av systemet som fortsatt ligger i repository'en. Etter disse ble fjernet og en liten feil ble rettet opp i en CSS fil, fikk koden en A rating.

Codacy

- Codacy pekte ut en del flere problemer enn det Code Factor gjorde. Den tok for seg ting som:

**"include" statement detected. File manipulations are discouraged.
Statement is not a function, no parentheses are required.**


```
include("model/Model.php");
```

- Ser ut som at Codacy gjør en mer grundig retting av syntax og utformingen på koden. Som f.eks at Codacy mener at CSS'en vår får strykkarakter pga feil indentering.

Expected indentation of 2 spaces (indentation)

```
margin-right: 4vh;
```

- Sier også at mye av metodene vi tar i bruk er frarådet. Som her:

“The use of function stripslashes() is discouraged”

```
$email = stripslashes(trim(htmlspecialchars($email)));
```

- I admin siden, har vi f.eks en ubrukt variabel som ble brukt tidligere, men er ikke fjernet:

- Avoid unused local variables such as '\$lecturers'.

```
$lecturers = $response['lecturers'];
```

Nyttig verktøy, men mye av feilene vi fikk opplyst, er kun pga utseende på koden. Typisk nitpick, og ikke nødvendigvis sikkerhetshull eller ting som går på performance.

Ubrukt variabel er noe vi burde ha fjernet.

Manuelt

Code Review ble gjort i to runder, med forskjellige deltagere. Dette lot de to delgruppene se over koden og finne feil den andre gruppen muligens har oversett.

Når vi skulle utføre en manuell code review var det to ting vi i hovedsak så etter:

- Er koden lesbar og lett forståelig?
- Fungerer koden som den skal?

Første runde:

Etter vi endret koden til en MVC-struktur ble koden mer lesbar og mindre “spaghettikode” aktig.

Når vi gikk igjennom med manuell code review, syntes vi koden har et ryddig og profesjonelt preg over seg.

Det er ingen tydelige “code smells”, ikke for mye kommentering, fornuftige navn på funksjoner og variabler, og funksjonene er ikke for store.

Forutenom noen variabler som ikke har noen videre hensikt enn å være variabler er det ingen deler av koden som er ubrukt eller overflødig.

Andre runde:

Koden er ryddig satt opp leselig, gitt at man forstår språket. Mye av koden gjenbruks der det gir mening, MVC strukturen hjelper også til med dette. Kommentarene for hver metode er korte og presise. Det kunne vært kommentar på hva som skjer enkelte metoder som har en “else” setning. Ellers så er det enighet med det som ble beskrevet i første runde.

Sammendrag av Code Reviews

- Koden kjører uten noen kjente bugs.
- Koden er skrevet etter MVC formatet.
- Bruker REST-API for appen, for databehandling, tar i bruk samme logikk for både app og nettsted. Ikke overflødig kode.

UTVIKLINGSPROSESS

Steg 1 - Utvikling av prosjektet

I begynnelsen av prosjektet prioriterte vi å lage nettsiden og appen, samt å oppfylle kravspesifikasjonene i oppgavebeskrivelsen.

- Vi hadde implementert noe session autentisering i PHP-koden som skulle forhindre at brukeren kunne besøke noen av nettsidene uten å være innlogget, men dette var ikke optimalt.
- Hadde noe input validering i HTML koden, f.eks kunne ikke skrive bokstaver i et inputfelt som hvor forventet input var tall.
- Vi hadde bevisst latt være å fjerne sikkerhetshull som f.eks feilmeldinger fra databasesystemet som vi hadde brukt under utviklingen av nettsiden.

Steg 2a - Angrepsperioden

Under angrepsperioden, ble nettsiden vår testet av to andre grupper, mens vi testet nettsidene til to andre grupper. Vi prøvde en del angrep, som SQL-injection, XSS, og Brute Force. Dette ga oss kunnskap om ting vi måtte sikre nettsiden vår mot.

Etter angrepsperioden fikk vi en tilbakemelding fra de andre gruppene om hva de hadde gjort på nettsiden vår. I rapportene vi fikk, sto det:

- At de kunne navigere seg fritt på siden vår ved å endre på URL-en, uten at de trengte å være pålogget.
- At vi hadde lite inputvalidering.
- At vi ikke hadde noen måte å forhindre spam, slik at de fylle databasen med masse falske brukere.
- At de hadde funnet en klasse i appen med navnet "Links.java", som inneholdt URL-ene som appen brukte.
- At de klarte å finne ut hva tabellene i databasen vår het, på grunn av feilmeldinger på nettsiden vår.

Steg 3 - Utbedringer av systemet

Begynte med å fokusere på svakheter i sikkerheten som de andre gruppene hadde funnet, samt sikkerhetshull som vi hadde latt være etter steg 1.

Input validering:

- I begynnelsen av steg 3 brukte vi PHP funksjonen `htmlspecialchars()` som skal konvertere typiske tegn som kan brukes i `script`(f.eks `"`, `'`, `<` og `>`) til HTML entites, og dermed uskadeliggjort. I ettertid derimot fant vi ut at dette ikke var beste løsningen, fordi dersom en bruker skriver inn et "ugyldig" tegn blir dette lagret i databasen som HTML entites og dermed nesten uleselig.
- Likevel valgte vi å la det stå i noen av (PHP)funksjonene der inputen ikke blir lagret i databasen(f.eks i `logg inn` funksjonen), som en ekstra beskyttelse mot XSS og SQL-Injection.
- La til input validering på alle inputfelt på nettstedet og appen i PHP-koden. Bruker `if`-tester som sammenligner inputverdien opp mot et `RegEx` mønster som var basert på hvilken input som var forventet.
F.eks i felt for e-postadresser bruker vi en innebygd PHP funksjon `filter_var()` som sjekker om verdien skrevet inn stemte overens med en typisk e-postadresse (inneholder `@` og `.` men ingen ugyldige karakterer).
- I tillegg til dette la vi inn mer inputvalidering i HTML-koden, dette gjorde vi for å ha validering både på client side og server side.
- Den ekstra inputvalideringen i HTML-koden kan ses på som overflødig men vi tenkte det er bedre å ha en ekstra "hindring" for å forhindre XSS og SQL-injection.
- Config fil i Apache er satt opp til å ikke tillate TRACE requests

Feilmeldinger:

- Fjernet alle feilmeldinger som ble vist på nettsiden, da dette ga for mye informasjon til brukeren.

Passord:

- I funksjonene for registrering av ny bruker la vi til en innebygd PHP funksjon `password_hash()` som hasher passord med Bcrypt når de blir lagret i databasen.
- For å sikre at brukerne våre har tilstrekkelig sterke passord har vi implementert `if`-tester i metodene som registrerer brukerne.
I `if`-testene bruker vi en PHP-funksjon `preg_match` som sammenligner inputverdien mot et `RegEx` mønster som krever minst åtte tegn, store og små bokstaver, minst ett nummer og minst ett spesialtegn.

PHP/nettsidens struktur:

- Vi omstrukturerte PHP-koden vår til å følge en Model-View-Controller-struktur. Dette gjorde vi for å få bedre oversikt over koden ved å løsne opp i spagettikoden fra versjon 1, hvor mye av HTML- og PHP-koden var blandet sammen. I den ferdige versjonen av nettsiden, har vi laget modellklasser som jobber med all dataen i systemet (altså, henter fra, og skriver til databasen). Controller-klassene tar i mot

input fra brukeren, og gjør endringer på modellene. View-klassene tar seg av det brukeren skal se. De henter de forespurte dataene fra modellklassene, og viser det til brukeren (i HTML koden).

SQL:

- Vi byttet ut alle SQL-spørringene i PHP-koden med MySQLi prepared statements med parametere, som forhindrer SQL-injection.
- Det er forskjellige brukere med ulike rettigheter som brukes der det er nødvendig.

Session autentisering:

- Når en bruker logger inn på nettstedet sin startside så blir e-postadressen og passordet sjekket opp mot databasen sine tabeller for forelesere, admin og studenter.
- Dersom brukeren er registrert i systemet blir en session ID tildelt basert hvilken tabell brukeren er registrert i.
- Her bruker vi en SQL spørring som tester brukernavnet og passordet opp mot tabellen og dersom en rad returneres eksisterer brukeren i den tabellen.
- Dersom brukeren ikke finnes i systemet skjer det ingenting.

DoS/DDoS Protection:

- Har brukt mod_Evasive for DoS protection.
- conf fil: over
- Om mer enn 15 connections i sekundet mot en side, tolkes dette som et angrep. For hele nettstedet så må det kobles 200 ganger på 5 sekunder for å telles som Dos angrep.
- Vil da bli kastet ut i 1 time. Konfig fil:
 - DOSPageCount 15
 - DOSSiteCount 200
 - DOSPageInterval 1
 - DOSSiteInterval 5
 - DOSBlockingPeriod 3600

Logging: Graylog

I prosjektet vårt har vi tatt i bruk Graylog for å logge handlinger i systemet. Graylog er et system for logging som vi installerte på serveren. Graylog er basert på Elasticsearch og MongoDB. Dette var en del av prosjektoppgaven å logge relevante hendelser i systemet. Vi tok i bruk Graylog fordi det var en gratis løsning som vi ble foreslått at vi kunne bruke.

Fordelen med et loggesystem er at man kan detektere "unormale" handlingsmønstre eller "skumle" handlinger som forekommer i systemet. Eksempel på en "skummel" handling er gjentatte ganger at en bruker av systemet skriver inn typiske tegn som blir brukt i script eller SQL-spørringer. Dette kan være et tegn på et angrepsforsøk. Det er viktig å ikke varsle om for mye i et system for å unngå at kritiske handlinger i systemet ikke blir oversett. Under ser du en liste over ting som vi logger i systemet vårt.

Kritiske handlinger som er korrekte / autentisering:

- Hvis bruker logger inn.
- Hvis bruker prøver å logge inn men skriver feil passord eller brukernavn.
- Hvis bruker bytter passord.

Feil ved input:

- Logger når hendelser, innloggingsforsøk og andre inputfelt inneholder tegn som ikke er lov.

Tilgangsfeil/nektinger (autorisasjonsfeil):

- Hvis en bruker som ikke har tilgang til en side kommer til en side som de ikke har tilgang til.
- Det ble ikke implementert en blokkering av tilgang til Graylog. Det ble ikke prioritert fordi man allerede trenger passord for å komme inn.

Feil i sesjonshåndtering:

- For å unngå sesjonskaping kan man i koden sjekke om det er to sessions-ID'er i bruk samtidig, hvis de er det så blir de logget ut.

Applikasjonsfeil og systemfeil:

- Hvis en henvendelse ikke kommer fram til databasen så skal det logges.

Generelle feil:

- Hvis bruker prøver å gå til en side som ikke eksisterer.
- Hvis bruker skriver inn en emnekode i url-en som ikke eksisterer.

Verdier som endres:

- Logger når en bruker endrer passordet sitt.

Input/output:

- Logge henvendelser som kan bli brukt som bevismateriale etc.

Uventede handlinger og bruksmønstre:

- Logger hvis en admin fjerner brukernavnet/endrer det til en bruker som ikke eksisterer, før han godkjenner læreren.

Filopplastninger:

- Når en bruker registrerer seg som en foreleser blir de bedt om å laste opp et bilde, de blir da lagret i server filene som igjen peker på disse gjennom databasen.

- (PHP) På siden for registrering er det en metode "isLegalFile" som kjører tester på valgt fil.
- (PHP) Funksjonen tester om filen er over den tillatte størrelsen og om det ikke er en av de tillatte filtypene(.jpg, .jpeg, .png, .gif).
- Dersom filen ikke slår ut på disse testene returnerer metoden true.

ARBEIDSFORDELING/REFLEKSJONSNOTAT

Vi har brukt Trello for å ha en oversikt over gruppens gjøremål, hva som er under arbeid og hva som er ferdig. Her kan man se noe av arbeidsfordelingen gjennom prosjektet.

<https://trello.com/b/EMV2KCW3/secgruppe5>

Samarbeid i gruppen

I begynnelsen av prosjektet fordelte vi arbeidsoppgaver i mellom medlemmene i gruppen. Har vært mye samarbeid mellom de forskjellige punktene, jobbet bra som en gruppe, dette er ikke lett å representere dette i en tabell, der noen punkter er større enn andre. Vi hadde også regelmessige gruppemøter på skolen, og etter 15.mars fortsatte vi med dette med men over internett i stedet.

Før 15. mars:

Alexander:

- Oppsett av webserver med Apache. (MySQL, PhP og annet software Apache trenger.)
- Feilsøking.

Frida:

- Skrev PHP-kode og HTML-kode for første versjon av nettstedet.
- Session autentisering for studenter og forelesere.
- Opprettet første versjon av database.
- Feilsøking.

Fredrik:

- Etterskriving av PHP-kode og HTML-kode.
- Oppsett av server via command line
- Satt opp tillatelser på serveren for brukerkontoen.
- Feilsøking.
- Laget første versjon av API mellom app og nettside.

Lars Martin:

- Jobbet med appen.
- Jobbet med API mellom app og nettside.
- Lagde siden for innsending av henvendelser i PHP.
- Konfigurering av databasen.

Sander:

- Var med på å planlegge og sette opp databasen.
- Konfigurering av databasen.
- Jobbet med appen.

Etter 15. mars:

Alexander:

- Installering av Graylog.
- DoS/DDoS protection med bruk av ModEvasive.
- HTTPS SSL selv-sertifisering.
- Dokumentasjon.
 - Abuse cases.
 - Sikkerhetskrav.
 - Risikobasert Sikkerhetstester.
 - Code review med verktøy.

Frida:

- Inputvalidering med RegEx i PHP og HTML.
- XSS protection.
- SQL-Injection protection.
- La til og fikset brukergrensesnittet på den endelige versjon av nettstedet.
- Dokumentasjon.
 - Abuse cases.
 - Sikkerhetskrav.
 - Risikobasert Sikkerhetstester.
 - Code review med verktøy og manuelt.

Fredrik:

- Oppsett av config filer i Apache og PHP
- XSS protection
- Konfigurering av brukerrettigheter i databasen
- Implementering av forskjellige database-brukere på nettsiden
- Code Review
- Dokumentasjon.

Lars Martin:

- Endret PHP-koden for nettsiden til en Model-View-Controller-struktur.
- Endret alle koblingene til databasen i PHP-koden til MySQLi prepared statements.
- La til hashing av passord i PHP-koden.
- La til endring av filnavn på bildefiler som lastes opp til serveren i PHP-koden.
- La til logging av handlinger på nettsiden i PHP.
- Endret appen for å få den til koble seg til den nye nettsiden med HTTPS.
- Hjalp til med oppsett av Graylog.
- Jobbet med Risk Management Framework.
- Jobbet med risikoanalyse av arkitekturen.

- Dokumentasjon.

Sander:

- Installerte Graylog på serveren.
- La til logging av handlinger på nettsiden i PHP med Monolog.
- Dokumentasjon.
- Jobbet med Risk Management Framework.
- Jobbet med risikoanalyse av arkitekturen.