Prosjekt - gruppe 3

Rapport steg 2:

Til steg 2 valgte vi å sette oss ned og begynne å dele ut de forskjellige oppgavene. I likhet med steg 1 fant vi ut at det var lettere at folk jobbet med det de allerede hadde jobbet med, slik at det allerede var en forståelse bak det å lettere og implementere sikkerheten. Helt i starten valgte vi og fordele abuse cases og sikkerhetskrav der alle på gruppen skrev ned noen egne tanker rundt det de hadde jobbet med. Risk Management Framework (RMF) tok vi i plenum der alle på gruppen fikk delt sine tanker rundt de forskjellige problemene.

Risk Management Framework og risiko analyse av arkitekturen

ID	Navn	Beskrivelse	Risk Indicator	Sannsynlighet	Konsekvens(tekst)	Konsekvens	Antatt kostnad/tap	Alvorlighetsgrad
BU-1	Config for apache	Config-fil for apache ligger ute tilgjengelig for alle	Bruker finner default index-side	Høy	Indikerer svak sikkerhet, åpner for angrep for standard verktøy	Høy	Antatt høy kostnad	Høy
BU-2	Tilgjengelighet for bilder	Bildene ligger lett tilgjengelig i mappestruktur	Inspiserer bildeelement	Middels	Kan scrapes	Høy	Antatt høy, brudd på personvern	Middels Høy

BU-3	Legge inn JS, etternavn	Man kan legge inn JS kode i etternavn input-boksen (foreleser)	Bruker legger inn kode i etternavn	Høy	XXS-scripting	Høy	Antatt høy	Høy
BU-4	Passord	Passord med 1 tegn er mulig	bruker ønsker et kort passord	Middels	Brute force angrep blir lettere, social engineering	Høy	Antatt middels	Middels høy
BU-5	Falske brukere	Det er ingen begrensning på antall falske brukere	Bruk av lagringsplass, oppdager mange brukere på kort tid	Middels	DoS-angrep grunnet systemressurser blir oppbrukt	Høy	Antatt høy	Middels høy
BU-6	Spam, meldinger	Kan legge inn så mange meldinger man vil	Bruk av lagringsplass, finner ikke meldinger man leter etter	Middels	Dårlig brukeropplevelse, DoS-angrep grunnet systemressurser blir oppbrukt	Høy	Antatt høy	Middels høy
BU-7	Full database	Tjenesten kan stoppe hvis databasen blir full	Kan ikke lage nye brukere, eller sende meldinger	Høy	DoS-angrep grunnet systemressurser blir brukt opp, dårlig	Høy	Antatt høy	Høy

					brukeroppl evelse			
BU-8	Filstruktur vises	Filstrukturen ligger ute og tilgjengelig for alle	Bruker skriver url som fører til bildemapp en	Høy	Kan få tilgang til skjulte filer som php filer, kan finne server-sensitiv informasjo n	Høy	Antatt høy	Høy
BU-9	Manglende Input validering	Input validere meldinger, innlogging, registrering	Siden blir angrep, siden er nede, brukerinfor masjon ser ikke riktig ut	Høy	Generelt hacking, XXS, DoS-angre p, SQL injection	høy	Antatt høy	Høy

Code review

Ved gjennomgang av koden for registrering av forelesere, fant gruppen at etternavn aldri ble input validert. Dette førte til at feltet kunne ta imot javascript kode. Videre er det ikke input validering av innloggingssiden. Det er ingen begrensninger på handlinger brukere kan gjøre, som eks tids delay på innlogging, eller hvor mange meldinger en student kan sende innen et gitt tidsintervall. Videre baserer underliggende abuse case og sikkerhetskrav seg på funn ved code review.

Abuse cases

Student og foreleser innloggingsside

1. Botting(bruteforce login)
2. SQL injections
3. Man kan D-DOSE ved registrering
4. Man kan registrere så mange forelesere man vil for å kunne fylle opp databasen med bilder

Foreleser bruker side

1. Hva som vil oppnås: Se meldinger fra studenter som ikke tilhører emne som foreleser har.
   Hvordan de vil oppnå det: Manipulere parameterne i hidden feltet til emnet.
2. Hva som vil oppnås: Få informasjon om studentene fra databasen.
   Hvordan de vil oppnå det: Ved å gjøre SQL injection mot databasen via kommentaren de kan sende til en student.

Student tilbakemelding siden:

1. Hva som vil oppnås: Få tjenesten til å stoppe opp/DoS-angrep
   Hvordan de vil oppnå det: Brukeren kan spamme databasen med meldinger

Database:

1. Nye brukere i databasen har standardiserte brukernavn og passord, disse vil kunne bli benyttet av en angriper.
2. Angriper som fyller opp hele databasen og tømmer den for minne som vil sette en stopper for hele serveren

Password reset:

1. SQL injection
2. MITM, sniffe trafikken med tanke på at det er HTTP.
3. DOS, belaste tjenesten ved å sende mange request eller ha requests som tar plassen der requesten ikke er tidsbegrenset
4. Kryptere passord slik at det ikke blir lagret i klartekst hos databasen. Dette vil gjøre de hashet passordene ubrukelig hvis angriperne lykkes med å få tak i data fra databasen.

Gjestebruker

1. Tilgang til fag som man ikke skal ha tilgang på ved brute force av 4-sifret pin
2. Få tilgang til database og få ut passord og sensitiv informasjon om brukerne og finne ut hvordan systemet fungerer

Sikkerhetskrav

Innloggingsside, student og foreleser:

1. Timeout ved for mange forsøk
2. Captcha
3. Least privilege så hvis kode blir sendt i det hele tatt vil man ikke ha rettigheter til å execute.
4. Brute-force attacks, automatisering av innlogging med de kjente brukernavn og passord
5. to-trinns faktorisering
6. minimum 8 tegn
7. inputvalidering av brukernavn og passord på innloggingsside

Foreleser bruker side:

1. Hvis foreleser forsøker å se emner de ikke har via parameter manipulasjon så skal det logges.
2. Kommentaren skal sendes via et prepared statement slik at det skal beskyttes mot SQL injection.
3. Hvis kommentar ser ut som en SQL injection så skal vi logge det.

Student tilbakemelding siden:

1. Systemet skal ha en begrensning for hvor mange meldinger som kan sendes i et gitt tidsintervall
2. Systemet skal hente emner og forelesere gjennom en koblingstabell fra databasen
3. Meldinger skal valideres for spesialtegn osv

Database:

1. Sørge for å minske antall privilegier brukere har, og benytte seg av sterke passord på disse. Unngå å lage brukere som har tilgang til alle “tables”
2. Sette en grense for hvor mye data som kjøres gjennom databasen slik at den ikke krasjer og vi risikerer data på avveie, og minimere nedetid.

Passord reset:

1. Https, sikrere kommunikasjon for å forebygge avlytting
2. Prepared statements, forebygge sql injections
3. Regulære uttrykk, svarteliste tegn som ikke er nødvendig å ha i input-formen.
4. Prosedyrer
5. Views, begrense tilgangen til databasen
6. Legge til unik Salt for hver bruker for å gjøre krypteringen sikrere
7. Redusere tid for hver request
8. Legge til måler for passordstyrke
9. Begrenset antall forsøk for passord reset i en tidsperiode
10. Hashing av passord

Endringer gjort i applikasjonen

Endringer som vi har gjort til steg 2 i config-filene har blitt gjort med inspirasjon fra forelesningsnotatene.

Satt opp en ny Virtual Host

Endret DocumentRoot fra /var/www/html til /var/www/steg2
Skjult informasjon om server ved bruk av /ServerTokens prod og /ServerSignature off
Fjernet HTTP 1.0
Fjernet ETags (FileETag None)
Fjernet Cross Site Tracing (TraceEnable off)
Benyttet <LimitExcept>. Gjør det mulig kun å benytte GET, POST, HEAD til serveren
“Hardkodet” headere i webserveren. (HTTPOnly, clickjacking og XXS-protection)

PHP.ini konfigurasjon

allow_url_fopen = Off
allow_url_include = Off

max_input_time= 30
max_execution_time= 30

memory_limit= 8M
post_max_size = 20M
max_execution_time = 30
max_input_vars= 100

register_globals= Off
expose_php = Off
cgi.force_redirect= 1

date.timezone = Europe/Oslo

Endringene vi kunne gjøre uten å måtte endre på hvordan vi håndterer at brukere skal logge inn.

session.use_strict_mode = 1
session.use_cookies = 1
session.cookie_samesite = Strict
session.cookie_lifetime = 14400
session.name = IKKEHVADUTENKTE

- Det var andre session innstillinger vi ikke kunne implementere da dette førte til at innloggingen feilet. Dette igjen fører til lavere sikkerhet og er ønskelig at skulle vært implementert gitt mer tid.

max_file_uploads = 2

Firewall

Lastet ned mod_security
Brukt config forslag
sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf

Clonet Github eksempel

https://github.com/SpiderLabs/owasp-modsecurity-crs.git
sudo cp /usr/share/modsecurity-crs/crs-setup.conf.example
/usr/share/modsecurity-crs/crs-setup.conf

Inkludert

IncludeOptional /usr/share/modsecurity-crs/*.conf
IncludeOptional /usr/share/modsecurity-crs/rules/*.conf
i /etc/apache2/mods-enabled/security2.conf
Kommenterte ut IncludeOptional /usr/share/modsecurity-crs/owasp-crs.load
For å ikke inkludere samme fil to ganger

Lasted ned mod_evasive.

Autentisering:

to-trinns verifisering er lagt til for innlogging for foreleser og student.

Login Student:

Lagt til begrensninger i antall forsøk som er lov til å logge inn før man får timeout i 5 minutter

Logging:

- Lagt inn logging ved sql spørringer i student tilbakemelding siden. Samt regex utløser logging på melding input fra tekstbok som inneholder < eller />
- lagt til logging av feil i registrering (student/foreleser)
- Logger hvis bruker forsøker å endre på parametere til emne i Foreleser HjemmeSide og for kommentarer.

Database:

Gått igjennom SQL-script å sjekket etter mangler.
Opprettet 3 nye brukere (Foreleser, Student og gjest) og gitt disse de rettmessige rettighetene.
Endret små feil fra Steg 1

Passord lengde

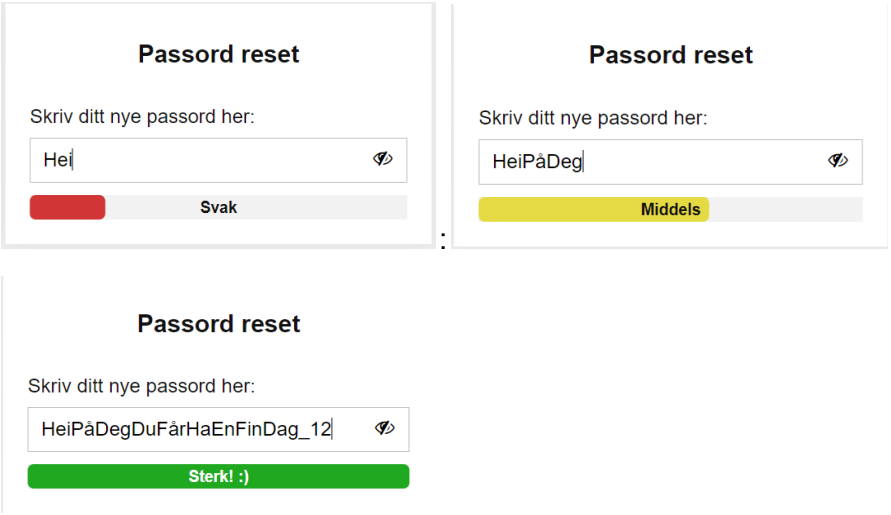
Økt den minimale passord-lengden til 7 tegn.

Passord reset

Målet med passord reset var å styrke hashing funksjonen og lage en passordstyrke måler. Forbedringen gjøres ved å legge til en salt for hver bruker. Dette gjøres på registreringssiden

når man lager en bruker, hver gang en bruker lages så skal det legges til en unik salt for brukeren. Deretter legges saltet bak passordet og hashes med en krypteringsteknologi. Gjerne med en tregere kryptering som bcrypt, for en tregere kryptering gjør det vanskeligere å bruteforce. Denne løsningen er veldig fin, men ble ikke fullt gjennomført fordi det er komplisert i praksis på grunn av både at den er bundet mot innloggingssiden, registreringssiden og passord reset, altså alle steder der hvor det er autentisering.

Laget også en passordstyrke måler som jeg testet lokalt. Visuelt fungerte dette veldig bra, men fikk den ikke til å fungere på live serveren. Resultatet ble bra. Html'en og CCS'en så fin ut, men ble skeptisk til JavaScript koden som brukes for å teste passord styrken. Erfaring med JavaScript har vi ikke og dermed kan passordstyrke måleren bli en ulempe enn en fordel. Positive med måleren er at den har en «psykologisk effekt» på brukeren ved at det lyser opp forskjellige farger som kan få brukeren til å få lyst til å lage et bedre passord. Ulempen er at JavaScript koden kan ha sikkerhetshull med tanke på at dette er hentet fra eksterne kilder. Best av alt ville det vært å skrive koden fra bunnen av, hvis det ikke går bør det gjøres en ordentlig secure code review for å se om den oppfyller sikkerhetskrav slik at det ikke blir en inngangsport til tjenesten. Noen bilder for å vise konseptet:



Her er selve algoritmen. For å få et sterkt passord må man oppfylle alle kravene. Styrken økes hver gang en if test bestås i funksjonen. Legg merke til at her brukes det også regulære uttrykk.

```
function getPasswordStrength(password){
  let s = 0;
  if(password.length > 6){
    s++;
  }
  if(password.length > 10){
    s++;
  }
  if(/[A-Z]/.test(password)){
    s++;
  }
  if(/[0-9]/.test(password)){
    s++;
  }
  if(/[^A-Za-z0-9]/.test(password)){
    s++;
  }
  return s;
}
```

Endringer gruppen ønsket å gjøre, men ikke fikk tid til

- Ved oppsett av serveren ble det snakket om å lage forskjellige brukere, men dette ble ikke prioritert. Nå med økt kunnskap ser gruppen at det hadde vært hensiktsmessig å ha implementert dette sammen med eks passordløs påkobling gjennom sertifikater eller to-faktor autentisering. Med sertifikater unngår man at passord kan komme på avveie, men VIKTIG at alle maskiner/brukere har sitt eget sertifikat og man må huske at man mister tilgangen hvis man mister/krasjer maskinen. Hvis man bruker to-faktor autentisering økes passordsikkerhet og man får muligheten til å logge på fra forskjellige maskiner, men et krav burde være at man må logge på via VPN.
- Endre innlogging slik at ytterligere session config innstillinger kan settes.
- HTTPS - Selv om dette ikke var et krav var det noe vi på gruppen ville prøve, men fikk ikke sertifikatene til å fungere. Vi tok deretter å utelukket dette og fokuserte på andre ting.
- Legge på honeypot felt ved registrering, slik at man minimerer bot angrep.