Rapport Steg 2

Gruppe 1:

- Jarle Syvertsen, jarle.syvertsen@hiof.no
- Anel Hadzic, anel.hadzic@hiof.no
- Håkon Halland Hovland Pedersen, haakonp@hiof.no
- Alexander Ombudstvedt Zarkoob, alexander.o.zarkoob@hiof.no
- Ole Marcus Løve Hansen, omhanse@hiof.no
- Andreas Sebastian Salomonsen Thorbjørnsen, andreas.s.thorbjornsen@hiof.no
- Daniel Nilsen Johansen, daniel.n.johansen@hiof.no

Sikkerhetsforbedringer og kjente problemer

SQL-Injection

I steg 1 tok vi kun i bruk prosedyrer sin interne «casting» for sikkerhet (der dette var gratis i forhold til utvikler tid, så følte dette ikke gikk utover «ikke tenk sikkerhet».) Likevel har dette alene noen svakheter, vi klarte f.eks. å utføre sleep commandoer via prosedyrene, og det er deriblant dokumentert at string-concatination av disse verdiene kan bryte sikkerheten (vi benyttet derimot ikke dette.) Selv om vi (og gruppe 2) ikke klarte å fremkalle noen seriøse feil, er ikke dette noen grunn for å ikke utbedre:

Prepared statements

Alle queries som ikke var prepared statements var konvertert til prepared statements. Der inputet til databasen er int blir disse kastet også før prepared statements. Dette betyr at disse har beskyttelsen av php-casting → prepared-statements → procedures.

SQL-Accounts

I oppgave 1 benyttet kun 1 brukerkonto med tillgang til alle prosedyrene, samt en DEFINER konto med tilgang til all data i den gitte databasen. Tilfeldige SQL queries kunne ikke utføres, men hadde en lavere priviligert bruker fått tilgang til å kalle prosedyren direkte, kunne høyere privilegerte prosedyrer bli kalt.

Databasen har nå fått brukerene:

API (Beyttet også til hard-limiting av kall)

Register/Gjest (Gjest blir behandlet som en ulogget inn bruker med unntak av emne området, der de trenger noen utvidelser av rettigheter.)

Vanlige brukere (Gjest blir eksalert til dette kun i emne siden, kunne i praksis gjøre at de kunne kalt kjente emner/fag, men de har ingen gyldig brukerid for å matche disse forespørslene, så dette ble akseptert for å ikke ha for mye duplisering av rettigheter.)

Administratorer (For administreringspanel.)

De har også fått to nye DEFINERS

DEFINER_HIGH

DEFINER_LOW

Disse lar oss definere tabeller og underliggende data i to forskjellige klasser. Dette benyttes for å definere rettighetene på prosedyrene. Fordelen med dette er at rettighetene til de underliggende dataene ikke trengs å gis til «brukerne» av prosedyrene (såkalte INVOKERS), noe som ville involvert at

disse kontoene kunne ha kalt rå SQL mot disse tabellene. DEFINERENE har videre ingen passord+ingen tillatt pålogging uten passord, der disse aldri skal brukes som ekte brukere.

XSS/JS-Injection

Sandboxing

Sandbozing av felter som kan inneholde brukerinnhold har blitt introdusert. Dette er spesielt relevant i samtaleboksen. Disse sandboxene har ingen tillatelser (med unntak av at meldingersboksen har tilgang til form). Dette innvolverte endring av hvordan data sendes tilbake fra emne (for å støtte reskaping av sandboxen uten å reloade siden, og endring av alle knapper til å fungere med en CSS backend for å blokkere all JS.)

Vi kunne gått for den lettere og sikrere delen av å isolere hver melding i en egen sandbox, men etter testing var det med kun 500 meldinger en loadtime på ish 15 sekunder uten lazy loading (ikke støttet på alle nettlesere), og 600 mb ekstra minnebruk. En test med 2000 meldinger gjorde at loadingen enda ikke var ferdig 1 minutt inn, og trengte 1.5GB med ekstra minnebruk. Dette ledet til valget om at en felles sandbox var best, men endringer til å støtte at denne sandboxen kunne være så strengt låst som mulig.

Videre endring i hvordan koden benytter anførselstegn ble gjort for å støtte en ekstra html_escape av disse før output, for hardening mot escaping av denne sandbozen.

Security-headers

Security headers ble lagt til for videre hardening. Content security policy blir brukt til å stramme javaskript (benyttes ikke på siden i det hele tatt, så kan disables), og intern css (med unntak av emne, der disse benyttes for å formatere kommenterer etter dybde, (attr støttes enda ikke for ints, eller så kunne dette håndteres på en god måte).

Javascript brukes på documentation index websiden, av den grunn kan det e

Refereres er også strammet inn via disse headerne.

Registrer – Feil-tilbakemelding

Et konkret problem ble funnet av gruppe 2 der registeringssidene benyttet en GET spørring for å bestemme hvilken feilmelding som skulle vises. Disse ble ikke riktig escapet, og security policyen var ikke aggresivt satt nok til å blokkere all ekstern javascript enda. Dette har blitt utbedret.

Problemet rørte aldri noe permanent lagring, og siden nettsiden vil i de aller fleste tilfeller (dette skulle vært alle engentlig), logge ut brukerene ved å gå inn på registrering, er ikke omfanget av svakheten så stor som nevnt, men likevel har den ingen grunn til å være der.

Dette understreket poenge med at alle inputs er farlige, og at ingen del av systemet kan gå i glemmeboken, selv om de ikke rører databasen og den generelle brukeren direkte.

Fileupload/Brukere kan se mappestruktur

I steg 1 hadde brukere mulighet til å se og snakke med filene direkte som ble hostet. Det var ikke mange sikkerhetssjekker (per oppgave krav), men ingen filer som hadde utvidelser annet en de vanlige bildene ble ikke godtatt, og selve filnavnet ble ødelagt for å unngå feilbehandling av nullchars og lignende. Dette gjorde at vi avhengte av at Apache aldri serverte en .jpg som noe annet en enn .jpg, som for oppgaven så ut til å duge. Likevel er denne delen av systemet veldig farlig, og hele module har blitt omskrevet.

File_handler

Bilder har blitt flyttet ut av apache sin domene fullstendig, som gjør det vanskelig å adressere filen direkte, og gjør det vanskeligere å få innsyn i hva som fungere. Dette skal også gjøre at .php filer, skulle de på noen måtte kunne bli lest som dette, ikke vil eksekvere på grunn av Apache sin standard i et public domene.

Dette gjør at filene i stedet hentes via scripter. Hva filene ble kalt av bruker, hva de faktisk er på disk, og en UUID som kan benyttes av HTML til å hente disse filene blir lagret og styrt av databasen.

Disse filene lastes heller ikke direkte, men basert på hva brukeren har som filutvidelse, vil filen gjenskapes til en GD_IMAGE format for php, og gjennomgå destruktive bilde handlinger før lagring. Bilde går også gjennom denne rekonstruksjonsprosessen ved henting, men uten destruktive handlinger.

Dette åpner for at parsingen av bilde kan abuses (dette er en umulighet å unngå for så å si all parsing), men modulene har ingen kjente feil, og dette ville krevet et veldig spesialisert angrep for å utnytte, så fordelen av at dette «vakser» alle filer etter verifisering av type (Check av header plus MIME type, i seg selv begge ikke gode nok beskyttelse) er ansett til å være verdt det.

Logging

Greylog er satt opp og benyttes for å aggregere og gi en lettere oversikt over innhold fra monolog og rsyslog. En del av modulene har fått satt opp monologging etter spesiell viktighet, og når disse modulene ble jobbet med direkte. Det ligger triggere for logging i selve applikasjonen på de hendelsene som er interessante ved hjelp av Monolog, definert i logger.php fil som blir brukt etter behov, og sender data via GELF handler til Graylog.

SSL

SSL/HTTPS er satt opp med eget sertifikat, og er aktiv på alle sider via Apache. Appen er oppdatert til å kunne gjenkjenne og godkjenne dette sertifikatet innenfor domenet.

DoS

mod_evasive

Mod evasive er satt opp som hoved beskyttelsen mot DoS, der den kan over alle steder i domenet blokkere mange forespørsler fra en IP i et kort tidsrom. Denne er satt noe generøst, men nok til at rask bruk av API, f.eks., låste ut brukeren.

API_user

Siden APIet var av spesiell interesse for å vise DoS; og kan isoleres fra hele systemet, er dette også satt harde limits på via SQL. Dette er derimot en hard limit i timen, og er i praksis en måte å DoSe hele APIet. Derimot vil dette tillate at resten av systemet fungerer, så vi aksepterte dette (i det minste for øyeblikket) til å være en akseptabel risiko.

Dataexfiltrering/Gjetting av struktur

Melding-IDer

Meldinger i systemet benytter forms for å registrere hvilket svar skal legges til hvilken melding, og til dette trenger den en ID på meldingen. I steg 1 kunne strukturen lett hentes og gjettes på ved å sjekke html-en i emne.php. I backend har vi migrert fra auto_inkrementerende ID-er til UUIDer på meldinger. Videre krypteres disse idene med SHA_2 algorithmen, der outputtet videre konverteres til base64 (ingen sikkerhetshensikt, men gjør at den fungerer til å puttes rett inn i htmlen.)

Krypteringen/Dekrypteringen + de nødvendige konverteringene gjøres via funksjoner opprettet i databasen, slik at disse funksjonene kan gjenbrukes etter ønske i andre moduler.

Apache/Webserver/Webfirewall/UFW Firewall

Mod evasive med nyeste OWAST regel sett satt opp.

Script på å banne IP i sammenheng med mod evasive satt opp.

To virtual host satt opp, en for port 80 og en for 433.

Alt av headers/CSP i henhold til forelesnings slides satt opp.

HTTPS med self-signed sertificate satt opp.

Satt opp HSTS

Redirect 80->433

Høyere krypterings cipher satt opp i henhold til råd ifra chat gpt 3.5

Javascript modul disabled

Satt opp bash script for å enable javascript/myphpmyadmin kun via ssh tunnel med ON/OFF script for å aktivere og disable access til myphpadmin.

Satt opp mer logging i apache2 conf

Firwall satt opp i henold til apps, laget nye apps.

Programvare som graylog etc etc etc satt opp i firewall via ssh tunnel.

Endret default charset.

Rate limiting satt opp.

Disabled følgende moduler:

Javascript-common.conf

Localized-error-pages.conf

Serve-cgi-bin.conf

Disabled 000-default.conf site

Fjernet apache-status websiden modulen

Samt noen moduler ifra mods-available/enabled.

Satt opp webmin server admin

Satt opp rate limiting i UFW firewall.

Satt opp reverse proxy via ferdig løsning cloudflare zero-trust-tunnel, dette er nå disabled igjen, men for å teste ut.