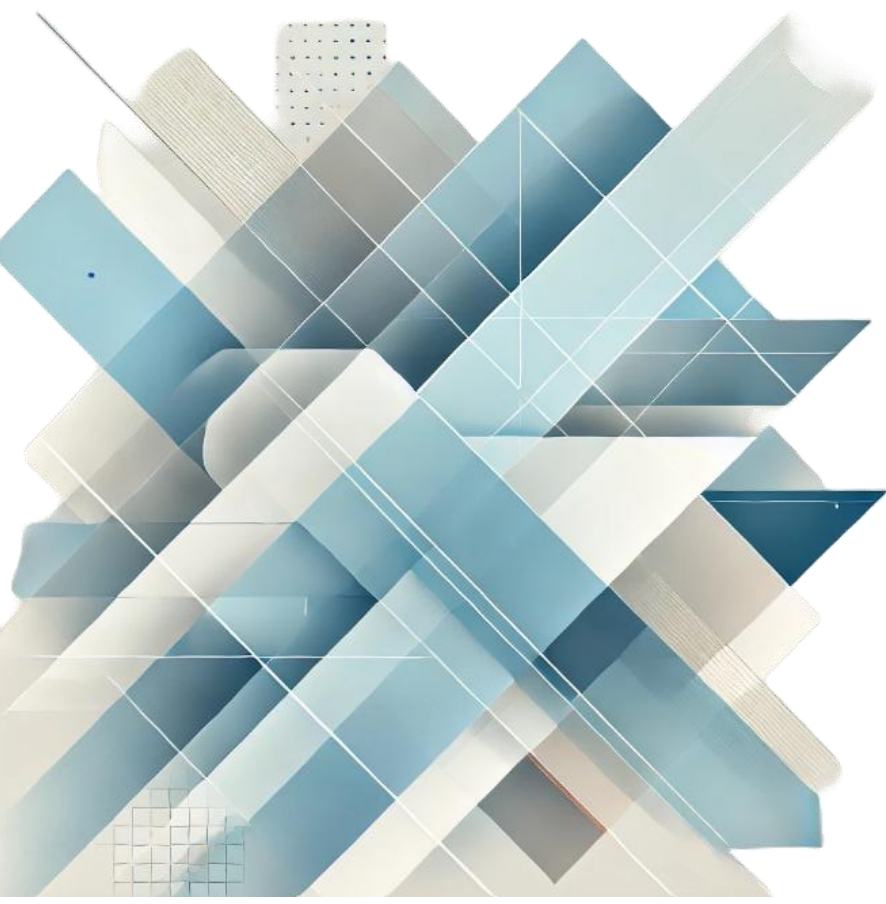


Teknisk Sårbarhetsrevisjon





Innholdsfortegnelse

Executive summary.....	3
Oppsummering av sårbarheter.....	3
Alle identifiserte sårbarhetsfunn.....	4
Risikonivå klassifisert etter farger.....	5
Testutførelse.....	6
Kontraktsregler	6
Deltagere	6
Detaljert beskrivelse av sårbarhetsfunn.....	7
Sårbarhet 1 (Ubeskyttet dataoverføring over HTTP)	7
Sårbarhet 2 (Klientsidemanipulasjon via inspeksjonsverktøy)	10
Sårbarhet 3 (Hardkoding av sensitiv informasjon)	14
Sårbarhet 4 (Utdatert og svak passordhashing med MD5)	16
Sårbarhet 5 (SQL Injection - SQLite)	18
Sårbarhet 6 (Uautorisert tilgang til /admin/admin.php)	19
Sårbarhet 7 (Utdatert Apache-versjon)	22
Sårbarhet 8 (Utdatert og synlig operativsystemversjon)	24
Sårbarhet 9 (SQL Injection – Oracle - Time-Based)	26
Sårbarhet 10 (Cross-Site Scripting)	28
Sårbarhet 11 (Content Security Policy Header Not Set)	30
Sårbarhet 12 (Missing Anti-clickjacking Header)	31
Sårbarhet 13 (Absence of Anti-CSRF Tokens)	33
Sårbarhet 14 (Missing X-Content-Type-Options Header)	34
Sårbarhet 15 (Cookie No HttpOnly Flag)	35
Sårbarhet 16 (Cookie without SameSite Attribute)	36
Informasjon 1 (Session Management Response Identified)	37
Informasjon 2 (Authentication Request Identified)	38



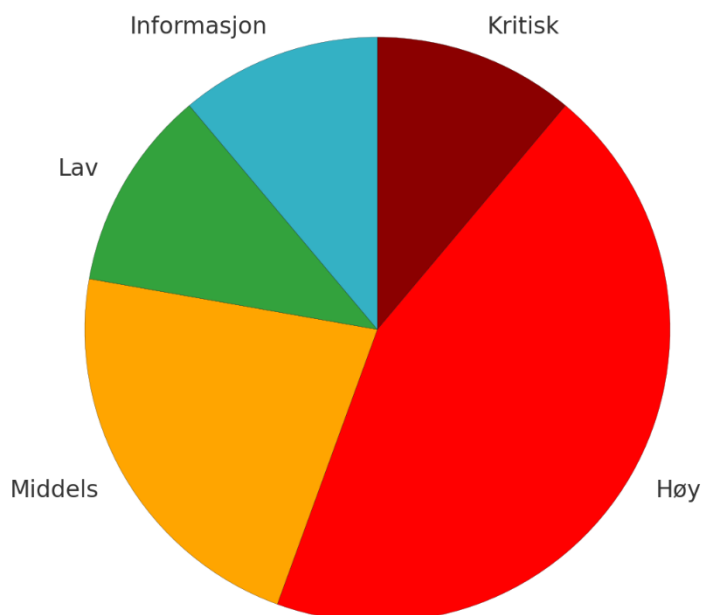
Executive summary

Nettsiden står overfor flere alvorlige sårbarheter som krever umiddelbar oppmerksomhet for å sikre dets konfidensialitet, integritet og tilgjengelighet. Blant de mest kritiske sårbarhetene er ubeskyttet dataoverføring over HTTP og klientsidemanipulasjon, som begge utgjør en betydelig risiko for sikkerheten. Under testingen ble det også identifisert andre sårbarheter, inkludert Cross-Site Scripting, som tillater angripere å injisere skadelige skript i brukernes nettlesere, og SQL Injection - Oracle Time-Based sårbarheter, som kan utnyttes til å manipulere databaseforespørsler for å få tilgang til sensitiv informasjon.

En annen betydelig sårbarhet er manglende anti-clickjacking header. Selv om denne sårbarheten er en middels risiko, bør den ikke undervurderes, da det kan føre til uautoriserte handlinger på vegne av brukerne.

Samlet sett er nettsiden utsatt for flere typer angrep som kan kompromittere både brukersikkerhet og systemets integritet. Det er viktig å løse disse problemene umiddelbart for å bevare brukernes tillit og unngå potensielle sikkerhetsbrudd.

Oppsummering av sårbarheter





Alle identifiserte sårbarhetsfunn

Sårbarhet 1	Ubeskyttet dataoverføring over HTTP	Kritisk
Sårbarhet 2	Klientsidemanipulasjon	Kritisk
Sårbarhet 3	Hardkoding av sensitiv informasjon	Høy
Sårbarhet 4	Utdatert og svak passordhashing med MD5	Høy
Sårbarhet 5	SQL Injection - SQLite	Høy
Sårbarhet 6	Uautorisert tilgang til /admin/admin.php	Høy
Sårbarhet 7	Utdatert Apache-versjon	Høy
Sårbarhet 8	Utdatert og synlig operativsystemversjon	Høy
Sårbarhet 9	SQL Injection – Oracle - Time-Based	Høy
Sårbarhet 10	Cross-Site Scripting	Høy
Sårbarhet 11	Content Security Policy Header Not Set	Middels
Sårbarhet 12	Missing Anti-clickjacking Header	Middels
Sårbarhet 13	Absence of Anti-CSRF Tokens	Middels
Sårbarhet 14	Missing X-Content-Type-Options Header	Middels
Sårbarhet 15	Cookie No HttpOnly Flag	Lav
Sårbarhet 16	Cookie without SameSite Attribute	Lav
Informasjon 1	Session Management Response Identified	Informasjon
Informasjon 2	Authentication Request Identified	Informasjon



Risikonivå klassifisert etter farger

Sårbarhetsfarge	Rød
Kritisk trusselnivå	Når en sårbarhet blir klassifisert som kritisk, må den håndteres umiddelbart. Disse sårbarhetene kan utgjøre en stor trussel mot systemet og nettverket, og kan føre til katastrofale skader hvis de blir utnyttet. Det er avgjørende å handle raskt.
Sårbarhetsfarge	Rød
Høyt trusselnivå	Når en sårbarhet blir klassifisert som høy, må den håndteres umiddelbart for å unngå alvorlige sikkerhetsbrudd.
Sårbarhetsfarge	Oransje
Middels trusselnivå	Når en sårbarhet blir klassifisert som middels, bør den håndteres innen en rimelig tidsramme for å unngå at den utvikler seg til et større sikkerhetsbrudd.
Sårbarhetsfarge	Grønn
Lavt trusselnivå	Når en sårbarhet klassifiseres som lav, kan den håndteres etter de mer kritiske problemene. Likevel bør den ikke ignoreres, da den over tid kan utvikle seg til en potensiell sikkerhetsrisiko.
Sårbarhetsfarge	Blå
Informasjon	Når en sårbarhet klassifiseres som blå, betyr det at den ikke utgjør en umiddelbar trussel. Det er hovedsakelig informasjon som er nyttig å forbedre over tid.



Testutførelse

Testen retter seg mot følgende inngangspunkt:

- Domenenavn: [REDACTED]
- IP: [REDACTED]

Tester har fått tilgang til følgende brukere/kontoer:

Web Portal User/PW:

- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]

Kontraktsregler

MSS Cyber Security og kunden har inngått en avtale, SECURITY ASSESSMENT AGREEMENT, som er signert av alle parter på forhånd. Avtalen som dekker dette oppdraget, er datert 01.09.2024 og gjelder for perioden 12.09.2024 til 29.09.2024.

Deltagere

Testene ble gjennomført av Mahamed-Maki Saine, Head of Cyber Operations hos sikkerhetsselskapet MSS Cyber Security.



Detaljert beskrivelse av sårbarhetsfunn

Sårbarhet 1

Sårbarhet 1	Ubeskyttet dataoverføring over HTTP
Kritisk	Ubeskyttet dataoverføring via HTTP er en veldig kritisk sårbarhet. En angriper kan fange opp sensitive data, som er en trussel mot systemets integritet og sikkerhet. Dette kan føre til katastrofale skader og må derfor håndteres umiddelbart.

Identifisering:

Sårbarheten ble identifisert ved bruk av Wireshark og manuell testing.

Området som ble påvirket:

<http://192.168.1.100/login.php>

Beskrivelse:

Ubeskyttet dataoverføring via HTTP, der brukernavn og passord sendes over nettverket i klartekst, er en kritisk sårbarhet som må håndteres umiddelbart. Dette betyr at sensitive data kan fanges opp av en angriper som overvåker nettverkstrafikken med verktøy som Wireshark. Hvis autentiseringsinformasjon ikke er kryptert, kan angripere enkelt få tilgang til brukerkontoer og kompromittere hele systemet. Videre kan dette føre til man-in-the-middle-angrep, der en angriper har muligheten til å manipulere eller avskjære data som sendes mellom brukeren og serveren.

Forebyggende tiltak:

For å forebygge ubeskyttet dataoverføring over HTTP, bør Hypertext Transfer Protocol Secure (HTTPS) implementeres som standard på alle nettsider. Det kreves at et SSL/TLS-sertifikat installeres på serveren. Dette sikrer at all kommunikasjon mellom brukeren og serveren er kryptert. For å sikre at nettsiden alltid kobler seg til HTTPS, er bruk av HTTP Strict Transport Security (HSTS) avgjørende. Regelmessig oppdatering av sertifikater og overvåking av sikkerhetsvarsler fra kilder som OWASP og NSM bidrar til å opprettholde en høy sikkerhetsstandard på serveren.

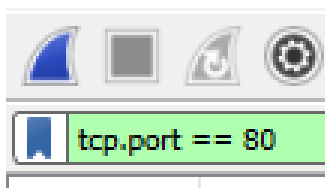


Utførelse:

Først startet jeg Wireshark for å overvåke nettverkstrafikken som går gjennom systemet. Jeg ville sjekke om sensitive opplysninger, som meldinger eller innloggingsinformasjon, ble sendt ukryptert. Derfor logget jeg inn på nettsiden med administratorbrukeren [REDACTED] og postet en melding. Målet var å se om Wireshark kunne fange opp denne meldingen.

Last messages	
Initiated By / Date	Message
[REDACTED] (Paris) Administrator 2024-09-27 20:57:42	Tester Wireshark.

Etter å ha postet meldingen på nettsiden, ble pakkefangsten i Wireshark stoppet. Deretter brukte jeg følgende kommando:



Målet til denne kommandoen er å filtrere trafikk som går over port 80, som er kjent for å være ukryptert. Valget av port 80 er basert på tidligere funn som viser at denne porten er ukryptert og åpen. Mer informasjon om dette vil bli gjennomgått senere i rapporten.

Deretter ble de fangede pakkene gjennomgått for HTTP-metoder som POST, GET, PUT og DELETE, som viser ulike forespørsler sendt til serveren. Blant disse ble en pakke identifisert som inneholdt brukernavn og passordet som ble brukt for å poste meldingen «Tester Wireshark»:

```
HTML Form URL Encoded: application/x-www-form-urlencoded
> Form item: "username" = [REDACTED]
> Form item: "password" = "[REDACTED]"
> Form item: "csrf_token" = "[REDACTED]"
> Form item: "login" = "login"
```




Dette bekrefter at brukernavn og passord ble sendt i klartekst, noe som gjør det enkelt for en angriper å fange opp og misbruke. CSRF-tokenet ble også fanget opp, noe en angriper kan bruke for å utføre uautoriserte handlinger på vegne av en bruker. Dette vil bli grundigere forklart i sårbarhet 13.

Videre ble de innsamlede pakkene undersøkt nærmere, og en pakke som inneholdt meldingen postet som administrator, ble identifisert:

```
-----
HTML Form URL Encoded: application/x-www-form-urlencoded
> Form item: "message" = "Tester Wireshark. "
> Form item: "csrf_token" = "[REDACTED]"
> Form item: "post_message" = "post_message"
```

Dette viser at meldinger som skulle vært private og sikre, kan leses i klartekst av angripere og andre uvedkommende.

Denne sårbarheten viser at all informasjon som sendes via HTTP, som brukernavn, passord og meldinger, er sårbar og kan lett fanges opp og utnyttes av en angriper med grunnleggende ferdigheter. Denne sårbarheten er derfor en kritisk prioritet, og må håndteres umiddelbart, for å beskytte integriteten og konfidensialiteten til nettsiden.

Finn ut mer:

- <https://nsm.no/getfile.php/133705-1592914512/NSM/Filer/Dokumenter/https.pdf>
- <https://nettrafikk.no/seo-tips/hva-er-https>
- https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/TLS



Sårbarhet 2

Sårbarhet 2	Klientsidemanipulasjon
Kritisk	Klientsidemanipulasjon ved hjelp av inspeksjonsverktøy har avdekket en alvorlig sårbarhet på nettsiden. Ved å opprette en uautorisert bruker kan en angriper omgå eksisterende tilgangskontroller. Når denne sårbarheten kombineres med andre eksisterende sårbarheter, som blir nevnt senere i rapporten, øker risikoen betydelig, og den blir svært kritisk og må håndteres umiddelbart.

Identifisering:

Sårbarheten ble identifisert gjennom manuell undersøkelse.

Området som ble påvirket:

<http://192.168.1.100/signup.php>

Beskrivelse:

Klientsidemanipulasjon er en kritisk sårbarhet. Den uautoriserte brukeropprettelsen er en alvorlig sårbarhet på nettsiden, som er kun ment for intern bruk. En brukers opprettelse uten autorisasjon, truer integriteten til tilgangskontrollen. Denne sårbarheten kan eskalere videre ved hjelp av sosial manipulering, hvor man kan lure administratorer til å aktivere den uautoriserte kontoen, noe som gir tilgang til sensitive data.

Ved å utnytte denne sårbarheten, kombinert med andre kritiske sårbarheter i systemet, som den utdaterte MD5-hash algoritmen brukt for passord, øker risikoen betraktelig. MD5 kan enkelt knekkes ved hjelp av verktøy som John the Ripper. Hvis for eksempel en admin-konto blir kompromittert, kan angriperen aktivere den uautoriserte brukeren og potensielt få større privilegier og tilgang til enda mer sensitive data.

Videre, ved å opprette en bruker med et troverdig brukernavn og e-postadressen til noe som ligner på en intern adresse, som [REDACTED], kan en angriper potensielt sende phishing e-poster



til andre ansatte i organisasjonen. Dette kan føre til at ansatte og administratorer gir fra seg sensitive opplysninger eller utfører handlinger som setter hele systemets sikkerhet i fare.

Dette gjør at sårbarheten ikke bare er høy, men kritisk, siden den åpner opp for en rekke angrep som kan kompromittere hele systemet. Kombinasjonen av disse sårbarhetene gjør det nødvendig med umiddelbare tiltak for å sikre nettsidens integritet.

Utførelsen av denne sårbarheten, vises lenger nede i rapporten.

Forebyggende tiltak:

For å forebygge uautorisert brukeropprettelse og andre sårbarheter knyttet til dette, bør flere tiltak implementeres. Regelmessig gjennomgang av brukerkontoer er avgjørende for å sikre at ingen uautoriserte kontoer eksisterer, og at kun nødvendige brukere har tilgang. Å ha strenge tilgangskontroller for å begrense tilgang til sensitive ressurser er også et viktig tiltak. Videre bør all aktivitet i systemet registreres og overvåkes sentralt for å oppdage mistenkelig oppførsel tidlig. Til slutt bør ansatte få jevnlig opplæring i sikkerhetsrutiner og være bevisste på phishing forsøk for å redusere risikoen for sosial manipulering.

Utførelse:

Jeg begynte med å klikke på lenken «Don't have an Account?» som tok meg til registreringssiden. Der fylte jeg ut all nødvendig informasjon, som brukernavn, passord, e-postadresse og andre påkrevde felt. Så prøvde jeg å lage en bruker, men «Sign up!» knappen var deaktivert.

Create an account

Username :	<input type="text" value="mss"/>	
Password :	<input type="password" value="....."/>	
Confirm Password :	<input type="password" value="....."/>	
Site :	<input type="text" value="Paris"/>	
Email address :	<input type="text" value="olanordmann@gmail.com"/>	
Firstname :	<input type="text" value="Ola"/>	

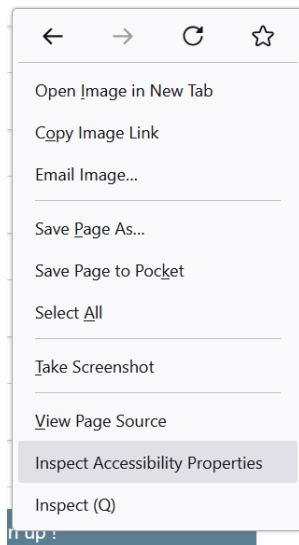
Firstname :

Signup!

Sign up !



For å aktivere knappen, høyreklikket jeg på «Sign up!» knappen og valgte deretter «Inspect».



I inspeksjonsverktøyet identifiserte jeg disabled attributtet for knappen. Jeg endret dette attributtet fra disabled til enabled.





Etter denne endringen ble knappen aktiv og var mulig å trykke på.

Sign up !

Etter å ha trykket på «Sign up!» knappen, fikk jeg en bekreftelse på at kontoen min var opprettet.

Your account was successfully created !

Men da jeg prøvde å logge inn med den nye kontoen, fikk jeg en melding som sa: «Your account has been locked or is inactive. Please contact the administrator team.»

Deretter navigerte jeg til admin/admin/php siden for å sjekke, og bekrefte at brukeren min var opprettet, men markert som inaktiv.

mss	Ola	Nordmann	olanordmann@gmail.com	Collaborator		Inactive
				Collaborator		Active
						Active

Finn ut mer:

- <https://www.csoononline.com/article/569533/john-the-ripper-explained-an-essential-password-cracker-for-your-hacker-toolkit.html>
- <https://nsm.no/aktuelt/ny-versjon-av-nsms-grunnprinsipper-for-ikt-sikkerhet-klar>
- <https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/informasjonsikkerhet-internkontroll/etablere-internkontroll/iverksette-styringssystem-for-informasjonsikkerhet/>
- <https://nordvpn.com/no/blog/sosial-manipulering/>



Sårbarhet 3

Sårbarhet 3	Hardkoding av sensitiv informasjon
Høy	Databasen har hardkodet sensitiv informasjon i kildekoden, som passord. Det er tilgjengelig for alle som har tilgang til koden, inkludert potensielle angripere.

Identifisering:

Sårbarheten ble identifisert gjennom manuell undersøkelse av kildekoden.

Området som ble påvirket:

Ingen områder ble påvirket.

Beskrivelse:

Hardkoding av sensitiv informasjon i kildekoden er en betydelig sikkerhetsrisiko. Å inkludere konfidensielle data som passord, API-nøkler eller databaseforbindelsesstrenger direkte i koden kan føre til alvorlige konsekvenser. Når slik informasjon er hardkodet, er den tilgjengelig for alle som har tilgang til koden, inkludert potensielle angripere hvis koden blir kompromittert. I tillegg er det vanskelig å oppdatere hardkodede koder uten å endre selve koden, noe som betyr at utdatert eller kompromittert informasjon vil bli fortsatt i bruk.

Forebyggende tiltak:

Flere forebyggende tiltak bør iverksettes for å unngå de alvorlige sikkerhetsrisikoene som oppstår ved å hardkode sensitiv informasjon som passord i kildekode. Når passordet «password» er hardkodet i koden, er dette en sårbarhet som angripere kan utnytte. Å bruke miljøvariabler i stedet for å hardkode sensitiv informasjon er en effektiv måte å håndtere denne sårbarheten på. Dette gjør det mulig å holde passord og andre private data utenfor koden, slik at de ikke blir eksponert. I tillegg er det lurt å vurdere å bruke sikre nøkkelhvelv for å beskytte sensitiv data og sørge for at disse tiltakene overholdes ved å gjennomføre regelmessige gjennomganger av kildekoden.



Utførelse:

Jeg navigerte gjennom de ulike mappene på serveren og gikk inn i konfigurasjonsmappen ved å bruke følgende kommando:

```
osboxes@osboxes:~$ cd /var/www/html/config
```

Deretter brukte jeg kommandoen:

```
osboxes@osboxes:/var/www/html/config$ ls -la
total 24
drwxr-xr-x 2 root root 4096 Jan 10 12:14 .
drwxr-xr-x 2 root root 4096 Jan 10 12:14 ..
-rw-r--r-- 1 root root 1211 Jan 10 12:14 config.inc.php
-rw-r--r-- 1 root root 1211 Jan 10 12:14 setup.php
```

For å liste opp filene som var i mappen.

Det var da jeg oppdaget filen config.inc.php, som inneholder kritiske innstillinger for databasen. For å åpne filen, brukte jeg kommandoen:

```
osboxes@osboxes:/var/www/html/config$ cat config.inc.php
```

Da oppdaget jeg at brukernavn, passord, og databasenavn var skrevet rett inn i koden.

```
osboxes@osboxes:/var/www/html/config$ cat config.inc.php
<?php
// Database Configuration
$_bdd = array();
$_bdd['server'] = 'localhost';
$_bdd['port'] = "3306";
$_bdd['user'] = "root";
$_bdd['password'] = "root";
$_bdd['database'] = "osboxes";
```

Finn ut mer:

- https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password
- <https://blog.codacy.com/hard-coded-secrets>
- <https://docs.guardrails.io/docs/vulnerability-classes/hard-coded-secrets>
- <https://workbook.securityboat.net/Pentesting/Thick%20Client/windows-application-pentesting/hardcoded-sensitive-information/>



Sårbarhet 4

Sårbarhet 4	Utdatert og svak passordhashing med MD5
Høy	Systemet benytter seg av MD5 for passordlagring, en algoritme med kjente svakheter som er utdatert og sårbar.

Identifisering:

Sårbarheten ble identifisert gjennom manuell undersøkelse av kildekoden.

Området som ble påvirket:

Ingen områder ble påvirket.

Beskrivelse:

MD5 har kjente svakheter som gjør det sårbart for brute-force angrep. Dette betyr at en angriper kan dekryptere hashede passord og få tilgang til sensitiv informasjon. Denne sårbarheten utgjør en betydelig risiko, da kompromitterte passord kan føre til uautorisert tilgang til brukerkontoer og potensielt videre utnyttelse av systemet.

Forebyggende tiltak:

MD5 er sårbar for brute-force, noe som gir angripere muligheten til å knekke hashede passord og få tilgang til sensitiv informasjon. For å beskytte passordlagring bør sterkere hash-algoritmer som bcrypt eller Argon2 brukes. Disse algoritmene er utformet for å være mer robuste mot angrep. Det er også viktig å holde hash-algoritmene oppdatert og ved bruk av flerfaktoraутентisering, så blir det et ekstra lag med sikkerhet.

Utførelse:

Jeg navigerte gjennom de ulike mappene på serveren og gikk inn i konfigurasjonsmappen ved å bruke følgende kommando:

```
osboxes@osboxes:~$ cd /var/www/html/config
```




Deretter brukte jeg kommandoen:

```
osboxes@osboxes: /var/www/html/config$ ls -la
total 24
[REDACTED] root root [REDACTED] .
[REDACTED] root root [REDACTED] ..
[REDACTED] root root [REDACTED] config.inc.php
[REDACTED] root root [REDACTED] setup.php
```

For å liste opp filene som var i mappen.

Det var da jeg oppdaget filen config.inc.php, som inneholder kritiske innstillinger for databasen. For å åpne filen, brukte jeg kommandoen:

```
osboxes@osboxes: /var/www/html/config$ cat config.inc.php
```

Under gjennomgangen av koden oppdaget jeg at passordene til brukerkontoene er «hashet» med MD5 algoritmen.

```
if(!($stmt = $GLOBALS['__mysqli_ston']->prepare("INSERT INTO user VALUES
(1, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 1, NOW(), 1),
(2, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 2, NOW(), 1),
(3, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 1, NOW(), 1),
(4, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 1),
(5, '[REDACTED]', '' . md5('2[REDACTED]') . '', 'Manager', [REDACTED](), 1),
(6, '[REDACTED]', '' . md5('1[REDACTED]') . '', [REDACTED], 3, NOW(), 1),
(7, '[REDACTED]', '' . md5('1[REDACTED]') . '', [REDACTED], 3, NOW(), 1),
(8, '[REDACTED]', '' . md5('1[REDACTED]') . '', [REDACTED], 1),
(9, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 4, NOW(), 1),
(10, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 4, NOW(), 0),
(11, '[REDACTED]', '' . md5('[REDACTED]') . '', [REDACTED], 5, NOW(), 1),
(12, '[REDACTED]', '' . md5('[REDACTED]') . '', 'Collaborateur', [REDACTED], 5, NOW(), 1),
(13, '[REDACTED]', '' . md5('[REDACTED]') . '', 'Administrator', [REDACTED], 1, NOW(), 1)
))) {
    $SESSION['technical_error'] = "Sorry, a technical error has occurred : " . $GLOBALS['__mysqli_ston']->error;
    $stmt->close();
    return;
}
```

Finn ut mer:

- <https://debugpointer.com/security/md5-vs-bcrypt>
- <https://stytch.com/blog/argon2-vs-bcrypt-vs-scrypt/>
- <https://specopssoft.com/blog/hashing-algorithm-cracking-bcrypt-passwords/>



Sårbarhet 5

Sårbarhet 5	SQL Injection - SQLite
Høy	Websiden kan være sårbar for SQL-injeksjon, noe som kan føre til uautorisert tilgang.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.100/login.php>

Beskrivelse:

SQL-injeksjon er en sårbarhet som lar en angriper endre en databasespørring ved å injisere ondsinnet SQL-kode i inputfelter, for eksempel på parameteren «username». Dette kan tillate en angriper å kontrollere hvordan spørringen utføres, noe som kan tillate dem å hente ut sensitiv informasjon eller utføre uautoriserte handlinger i databasen. Selv om sårbarheten ikke er verifisert, er det viktig å beskytte applikasjonen ved å bruke sikre metoder for databaseforespørsler og begrense databasetilgangen.

Forebyggende tiltak:

For å beskytte mot SQL-injeksjonsangrep er det viktig å validere og kontrollere all input på serversiden. Bruk parameteriserte spørringer som PreparedStatement i stedet for å lage dynamiske SQL-spørringer. Unngå å bruke strenger for å sette sammen SQL-spørringer. I tillegg er det avgjørende at databasebrukerne har de minste og nødvendige privilegiene, og at dataene som mottas fra klienten blir rensset for å forhindre injeksjoner. Det bør også være en liste over tillatte eller forbudte tegn som brukes av brukerne.

Finn ut mer:

- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- <https://www.visma.no/blogg/eksempler-sql-injeksjoner/>
- <https://no.kaspersky.com/resource-center/definitions/sql-injection>



Sårbarhet 6

Sårbarhet 6	Uautorisert tilgang til /admin/admin.php
Høy	Denne sårbarheten tillater uautorisert tilgang til /admin/admin.php, noe som eksponerer sensitiv informasjon uten noen form for autentisering.

Identifisering:

Sårbarheten ble identifisert ved bruk av Nmap og manuell testing.

Området som ble påvirket:

<http://192.168.1.100/admin/admin.php>

Beskrivelse:

Siden /admin/admin.php er tilgjengelig uten autentisering, noe som gjør det mulig for uautoriserte brukere å få tilgang til sensitiv informasjon som brukernavn, e-postadresser, roller, og status (aktiv/inaktiv). Dette medfører eksponering av sensitive data og mulig misbruk av brukerkontoer, siden disse dataene kan brukes til innlogging eller utføre ytterligere angrep på nettsiden.

Forebyggende tiltak:

For å forhindre uautorisert tilgang til /admin/admin.php er det viktig å implementere flere sikkerhetstiltak. Det bør være en sikker autentisering og tofaktoraутentisering på alle administrative sider. Dette vil sikre at kun autoriserte brukere får tilgang til sensitive områder. I tillegg anbefales det å bruke rollebasert tilgangskontroll (RBAC) for å begrense tilgangen ytterligere. Ved å gi brukere spesifikke rettigheter basert på deres roller, kan risikoen for uautorisert tilgang minimeres.

Å bruke HTTPS for å kryptere all kommunikasjon mellom klienten og serveren er også et viktig tiltak. All trafikk som kommer fra port 80 bør flyttes til for eksempel port 443 for å sikre at trafikken alltid er kryptert.

Regelmessig gjennomgang og logging av tilgangsforsøk er også et avgjørende tiltak. Dette vil bidra til en effektiv beskyttelse av sensitiv informasjon ved å oppdage og reagere raskt på mistenkelig aktivitet.



Utførelse:

Jeg startet med å utføre en Nmap-skanning mot 192.168. [REDACTED] for å finne åpne porter og tjenester. Ifølge skanningen var port 80/tcp åpen, noe som betyr at en HTTP-server kjører på denne porten. Jeg oppdaget også en oppføring i robots.txt-filen under skanningen, som tydet på at det var en skjult administrasjonsside på /admin/admin.php.

OS	Host	Port	Protocol	State	Service	Version
	192.168. [REDACTED]	80	tcp	open	http	Apache httpd 2.4.38 ((Debian))

```
| http-robots.txt: 1 disallowed entry  
|_/admin/admin.php
```

Deretter gikk jeg manuelt til [http://192.168. \[REDACTED\]/admin/admin.php](http://192.168. [REDACTED]/admin/admin.php) i nettleseren, hvor jeg oppdaget en administrasjonsside som var tilgjengelig uten autentisering.

[REDACTED] /admin/admin.php

Home

Don't h

Users

Username	Firstname	Lastname	Email address	Role	Last Connection	Status	Action
[REDACTED]				Administrator	[REDACTED]	Active	
[REDACTED]				Collaborateur	[REDACTED]	Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]						Inactive	
[REDACTED]						Active	
[REDACTED]				Financial approver	[REDACTED]	Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]						Active	
[REDACTED]				Manager	[REDACTED]	Active	



Finn ut mer:

- <https://www.invicti.com/blog/web-security/protecting-website-using-anti-csrf-token/>
- <https://www.acunetix.com/websitesecurity/csrf-attacks/>
- <https://learn.snyk.io/lesson/csrf-attack/>
- <https://www.strongdm.com/rbac>



Sårbarhet 7

Sårbarhet 7	Utdatert Apache-versjon
Høy	Nettsiden kjører versjon 2.4.38 av Apache serveren, som inneholder sårbarheter som kan føre til alvorlige angrep. Normalt ville denne sårbarheten bli klassifisert lavere, men på grunn av de spesifikke sårbarhetene i denne versjonen, er risikovurderingen satt til høy.

Identifisering:

Sårbarheten ble identifisert ved bruk av Nikto og manuell undersøkelse.

Området som ble påvirket:

<http://192.168.1.100/>

Beskrivelse:

Apache-versjonen 2.4.38 som kjører på serveren har flere kritiske sårbarheter som kan utnyttes av eksterne angripere. En av de mest alvorlige er Server Side Request Forgery (SSRF), som gjør det mulig for angripere å få serveren til å sende forespørsler til andre systemer, noe som kan føre til lekkasje av sensitiv informasjon fra interne systemer. Det finnes også sårbarheter som kan føre til at angripere kan kjøre ondsinnet kode på serveren, og få full kontroll over systemet. I tillegg kan svakheter i håndteringen av WebSocket over HTTP/2 eller andre konfigurasjoner føre til at serveren krasjer. Disse sårbarhetene kan utnyttes eksternt uten spesifikke forutsetninger og har alvorlige konsekvenser hvis de blir utnyttet. Derfor er risikovurderingen satt til høy.

Forebyggende tiltak:

Oppgrader Apache til den nyeste versjonen. Den nyeste versjonen, Apache HTTP Server 2.4.62 inneholder viktige sikkerhetsoppdateringer. Ved å oppgradere til denne versjonen, reduseres risikoen for angrep og beskytter systemet mot utnyttelse av sårbarheter i eldre versjoner.



Utførelse:

Jeg startet med å åpne terminal i Kali Linux. Deretter brukte jeg kommandoen:

```
(kali㉿kali)-[~]  
$ nikto -h http://192.168.1.100  
- Nikto v2.5.0
```

Jeg kjørte denne kommandoen for å skanne webserveren. Skanningen startet med å identifisere «Target IP», «Target Hostname» og «Target Port».

```
+ Target IP:      192.168.1.100  
+ Target Hostname: 192.168.1.100  
+ Target Port:    80  
+ Start Time:     2024-09-28 14:00:00
```

Nikto viser at webserveren kjører Apache versjon 2.4.38 på en Debian-server, og at denne versjonen er utdatert.

```
+ Server: Apache/2.4.38 (Debian)
```

```
+ Apache/2.4.38 appears to be outdated
```

Finn ut mer:

- https://httpd.apache.org/security/vulnerabilities_24.html
- <https://www.rapid7.com/db/?q=apache+http+server>
- https://owasp.org/www-community/attacks/Server_Side_Request_Forgery
- <https://portswigger.net/web-security/websockets>



Sårbarhet 8

Sårbarhet 8	Utdatert og synlig operativsystemversjon
Høy	Operativsystemet er utdatert og synlig, noe som kan gi en angriper viktig informasjon som kan brukes til å utnytte sårbarheter.

Identifisering:

Sårbarheten ble identifisert ved bruk av Nmap og manuell undersøkelse.

Området som ble påvirket:

<http://192.168.> [REDACTED]

Beskrivelse:

Operativsystemet som brukes på serveren er en eldre Linux-kjerneversjon, 3.10 til 4.11, som er synlig for alle og inneholder flere kjente sårbarheter. Disse består av alvorlige sårbarheter som CVE-2017-18017, som kan utnyttes eksternt for å påvirke kjernens nettfiler, og CVE-2016-5195, som tillater eskalering av rettigheter og gir en angriper muligheten til skriveadgang til filer som ellers kun skal være lesbare. Den er it tillegg sårbar for andre sårbarheter, som CVE-2017-6074 og CVE-2016-8655, som gir mulighet for vilkårlig kodeeksekvering og eskalering av privilegier. Selv om synliggjøring av operativsystemversjonen i seg selv kunne anses som en sårbarhet med middels risiko, økes risikoen betraktelig av de kjente sårbarhetene. På grunn av de alvorlige konsekvensene disse sårbarhetene kan forårsake, vurderes risikoen som høy.

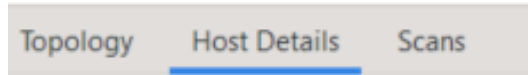
Forebyggende tiltak:

For å forebygge risikoen knyttet til den synlige og sårbare Linux-kjernen, bør Linux-kjernen oppdateres til den nyeste tilgjengelige versjonen. Serveren bør også konfigureres for å skjule eksponeringen av operativsystemets informasjon. Disse tiltakene vil redusere risikoen for at angripere enkelt kan identifisere Linux-versjonen, og redusere sannsynligheten for angrep.



Utførelse:

Jeg brukte Nmap til å skanne IP-adressen til nettstedet 192.168. [REDACTED]. Etter at skanningen var fullført, navigerte jeg til «Host Details»:



Hvor jeg fant informasjon om operativsystemet som kjører på serveren. Som vist i bildet, ble det identifisert at operativsystemet er en versjon av Linux mellom 3.10 og 4.11.

▼ Operating System

Name: Linux 3.10 - 4.11

Finn ut mer:

- <https://ubuntu.com/security/notices/USN-6974-1>
- https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
- <https://arstechnica.com/security/2024/05/federal-agency-warns-critical-linux-vulnerability-being-actively-exploited/>



Sårbarhet 9

Sårbarhet 9	SQL Injection – Oracle - Time-Based
Høy	Nettsiden er sårbar for Oracle Time-Based SQL-injeksjon. Ved å analysere tidsforsinkelser i databasens respons, kan angripere hente ut sensitiv informasjon eller utføre uønskede handlinger.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.100/signup.php>

Beskrivelse:

SQL-injeksjon – Oracle Time-Based er en sårbarhet der angripere utnytter tidsforsinkelser i databasens respons til å utføre injeksjonsangrep. På samme måte som SQL-injeksjon i SQLite, kan angripere manipulere inputfelter som «username» for å injisere skadelig kode. Forskjellen mellom denne sårbarheten og den tidligere nevnte SQL-injeksjonen i SQLite er at Oracle Time-Based bruker tidsforsinkelser i responsen som angripere kan analysere for å hente ut sensitiv informasjon eller utføre uønskede handlinger.

Forebyggende tiltak:

For å forhindre SQL-injeksjon – Oracle Time-Based bør applikasjonen bruke parameteriserte spørringer, i stedet for å sette inn rå brukerinput direkte i SQL-spørringen. Dette forhindrer injisering av skadelig SQL-kode ved å binde variabler. Videre bør alle brukerdata valideres og filtreres for å sikre at kun vennlig data blir akseptert. Det er også lurt å begrense databasetilgangen for applikasjonsbrukere. Til slutt kan bruk av Object-Relational Mapping (ORM) bidra til å fjerne behovet for direkte SQL-spørringer, og dermed redusere risikoen for SQL-injeksjon – Oracle Time-Based angrep.

Se vedlegget under på et SQL-injeksjonsangrep som ble oppdaget på parameteren «username».



Utførelse:

Parameter: username

field: [username], value

Attack:

")

ion SELECT

Finn ut mer:

- <https://hibernate.org/orm/what-is-an-orm/>
- <https://www.sqlalchemy.org/>
- [https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.h
tml](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)
- <https://portswigger.net/web-security/sql-injection>



Sårbarhet 10

Sårbarhet 10	Cross-Site Scripting (XSS)
Høy	XSS er en stor sikkerhetsrisiko. Det gir angripere muligheten til å utnytte sårbarheter på en nettside for å kjøre skadelige skript i brukerens nettleser.

Identifisering:

Sårbarheten ble identifisert ved manuell undersøkelse.

Området som ble påvirket:

<http://192.168.1.100/index.php>

Beskrivelse:

Cross-Site Scripting (XSS) er en sårbarhet som lar angripere injisere skadelige skript. Disse skriptene kan brukes i brukerens nettleser for å stjele sensitiv informasjon, kapre økter og utføre andre skadelige handlinger. XSS oppstår når et nettsted manipuleres til å returnere ondsinnet kode til brukere, noe som kompromitterer deres interaksjon med nettsiden.

Forebyggende tiltak:

Alle brukerinput bør valideres og renses for å forhindre XSS-angrep. Når brukere kan redigere HTML-innhold, er bruken av HTML-sanitisering, som DOMPurify, avgjørende. I tillegg bør en Content Security Policy (CSP) implementeres for å begrense hvilke ressurser som kan lastes inn. Å oppdatere alle rammeverk og komponenter regelmessig er i tillegg utrolig viktig for å beskytte mot kjente sårbarheter.

Utførelse:

Jeg logget inn på nettsiden som admin brukeren [REDACTED]. Deretter navigerte jeg nederst i siden til meldingsfeltet der jeg kunne poste meldinger. For å teste sårbarheten skrev jeg inn følgende skript:

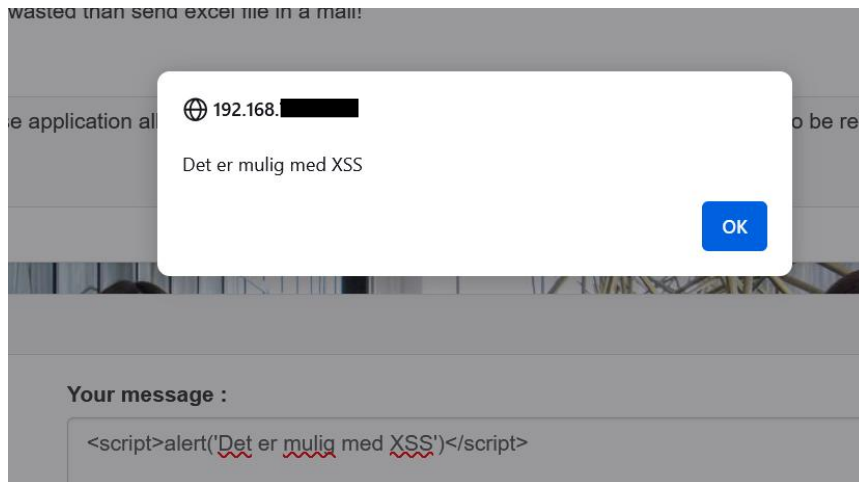
Your message :

<script>alert('Det er mulig med XSS')</script>

Post your message



Etter meldingen ble postet, førte skriptet til en popup-melding som sa «Det er mulig med XSS». Dette bekrefter at nettsiden er sårbar for XSS-angrep. Akkurat denne type XSS, er en refleksiv XSS.



Finn ut mer:

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- https://developer.mozilla.org/en-US/docs/Web/Security/Types_of_attacks#cross-site_scripting_xss



Sårbarhet 11

Sårbarhet 11	Content Security Policy (CSP) Header Not Set
Middels	Nettsiden er sårbar for cyberangrep som kan skade brukere eller stjele data.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<https://192.168.1.100/sitemap.xml>

Beskrivelse:

Content Security Policy (CSP) er et sikkerhetstiltak som bidrar til å oppdage og forhindre angrep som XSS og SQL-Injection. Disse angrepene kan føre til datatyveri, skade nettsider eller spre skadelig programvare. CSP gir administratorer muligheten til å bestemme godkjente innholdskilder gjennom vanlige HTTP-headere, som nettlesere kan laste inn på nettsiden. Dette inkluderer innholdstyper som JavaScript, CSS, HTML-rammer, fonter, bilder og innebygde objekter som Java-applikasjoner, lyd og videofiler.

Forebyggende tiltak:

For å forebygge denne sårbarheten bør serveren konfigureres til å sende en Content-Security-Policy-header. Dette bidrar til å begrense hvilke typer innhold som kan legges ut på nettsiden, og sikrer at kun godkjente materialer som bilder og skript kan legges inn. Ved å korrekt konfigurere en CSP-header, kan angrep som Cross-Site Scripting og andre former for ondsinnet kodeinjeksjon reduseres. Alle servere som håndterer nettsiden, inkludert webservere bør bruke dette sikkerhetstiltaket.

Finn ut mer:

- <https://content-security-policy.com/>
- <https://web.dev/articles/csp>



Sårbarhet 12

Sårbarhet 12	Missing Anti-clickjacking Header
Middels	Brukere kan bli lurt til å trykke på noe som kan føre til uønskede handlinger.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP og manuell undersøkelse.

Området som ble påvirket:

<http://192.168.███/login.php>

Beskrivelse:

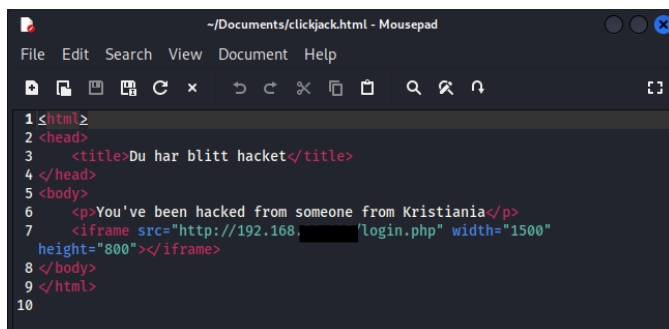
ClickJacking er en metode der angripere skjuler skadelig innhold bak synlige elementer på en nettside, som for eksempel en falsk «Last ned» knapp. Dette kan føre til at brukeren utfører uønskede handlinger på nettsiden, som å dele sensitiv informasjon som passord eller personlige data.

Forebyggende tiltak:

Moderne nettlesere støtter HTTP-headerne Content-Security-Policy og X-Frame-Options. Sørg for at en av dem er satt på alle nettsider. Bruk «SAMEORIGIN» hvis siden bare skal rammes inn av deres egen server, eller «DENY» hvis den aldri skal rammes inn. Alternativt så kan dere vurdere «frame-ancestors» direktivet i Content Security Policy.

Utførelse:

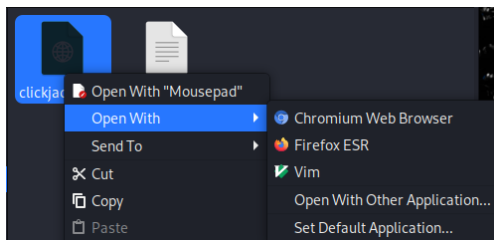
Jeg startet med å skrive kommandoen mousepad i kali Linux terminalen for å åpne teksteditoren. Der skrev jeg ned koden i HTML-filen, og sørget for at koden pekte mot URL-en <http://192.168.███/login.php>. Deretter lagret jeg filen som clickjack.html i Documents-mappen.



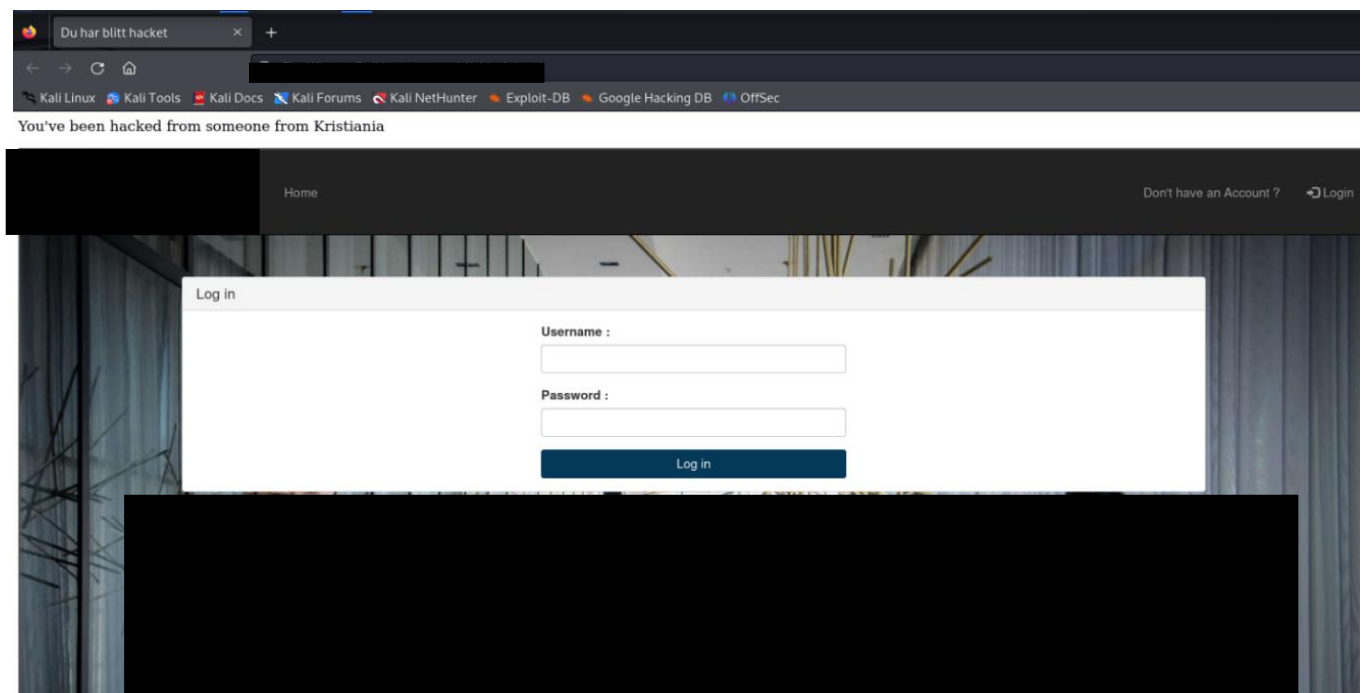
```
~/Documents/clickjack.html - Mousepad
File Edit Search View Document Help
1 <html>
2 <head>
3   <title>Du har blitt hacket</title>
4 </head>
5 <body>
6   <p>You've been hacked from someone from Kristiania</p>
7   <iframe src="http://192.168.███/login.php" width="1500"
8     height="800"></iframe>
9 </body>
10 </html>
```



Etter å ha lagret filen, åpnet jeg den i Firefox-nettleseren ved å høyreklikke på filen, velge «Open With» og deretter «Firefox ESR».



Når filen ble åpnet i nettleseren, viste den teksten jeg la inn i HTML-filen sammen med `http://192.168. [redacted] /login.php` inni en iframe. Dette bekrefter at nettsiden er sårbar for clickjacking angrep.



Finn ut mer:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>
- https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html



Sårbarhet 13

Sårbarhet 13	Absence of Anti-CSRF Tokens
Middels	Uten Anti-CSRF Tokens kan brukere ubevisst utføre handlinger de ikke ønsker, noe som kan sette sikkerheten i fare.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.100/signup.ph>

Beskrivelse:

Cross-Site Request Forgery (CSRF) er et angrep der en bruker uvitende blir lurt til å sende en forespørsel til et nettsted der de er logget inn. Angrepet er spesielt skadelig hvis brukeren har en aktiv sesjon eller er autentisert på nettsiden. Hvis nettstedet er i tillegg sårbart for XSS, øker risikoen fordi XSS kan brukes til å forsterke Cross-Site Request Forgery angrep.

Forebyggende tiltak:

For å forebygge Cross-Site Request Forgery bør man bruke for eksempel rammeverk som forhindrer CSRF-angrep. I tillegg er det lurt å implementere anti-CSRF-pakker som OWASP CSRFGuard, som beskytter siden ved å lage og validere tokens for hver forespørsel, slik at man kan være sikker på at forespørselen kommer fra en troverdig kilde.

Finn ut mer:

- <https://cwe.mitre.org/data/definitions/352.html>
- https://cheatsheetseries.owasp.org/cheatsheets/CrossSite_Request_Forgery_Prevention_Cheat_Sheet.html



Sårbarhet 14

Sårbarhet 14	Missing X-Content-Type-Options Header
Middels	X-Frame-Options bidrar til å beskytte nettsider mot clickjacking-angrep ved å forhindre at innholdet vises i rammer på andre sider.

Identifisering:

Sårbarheten ble identifisert ved bruk av Nikto.

Området som ble påvirket:

<http://192.168.1.100/admin/admin.php>

Beskrivelse:

X-Frame-Options er en sikkerhetsheader som kontrollerer om en nettside kan vises i rammer på andre nettstedet, noe som beskytter mot angrep som clickjacking. X-Frame-Options hindrer at siden blir rammet inn på andre sider, for å hindre uønskede handlinger fra brukeren. Tidligere ble clickjacking nevnt og hvordan denne type angrep kan føre til at sensitiv informasjon blir delt uten at brukeren er klar over det. Slike angrep kan stoppes med X-Frame-Options.

Forebyggende tiltak:

X-Frame-Options bør implementeres på alle sider med sensitivt innhold eller funksjoner for å forhindre uautoriserte innramminger. Som tidligere nevnt under forebyggende tiltak for clickjacking, anbefales det å sette X-Frame-Options til «DENY» for å blokkere all innramming, eller «SAMEORIGIN» for å tillate innramming kun fra samme domene. Dette bidrar til å opprettholde sikkerheten og integriteten til nettsiden.

Finn ut mer:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>



Sårbarhet 15

Sårbarhet 15	Cookie No HttpOnly Flag
Lav	JavaScript kan kjøre skadelig kode og kapre en sesjon ved å sende den til andre nettsteder.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP og manuell undersøkelse.

Området som ble påvirket:

<http://192.168.1.100/site.php?id=1> | <http://192.168.1.100/login.php>

Beskrivelse:

En cookie uten HttpOnly-flagget kan nås av JavaScript, noe som gjør det mulig for ondsinnede skript å lese cookien og sende den til en annen nettside. Angripere kan ta over brukerens aktive sesjon og få tilgang til kontoen eller sensitiv informasjon.

Forebyggende tiltak:

For å redusere risikoen for at JavaScript får tilgang til cookies, bør HttpOnly-flagget settes for alle cookies. Selv om nettsiden blir utsatt for et XSS-angrep, hindrer dette flagget skript fra å lese eller endre informasjonskapselen. Dette beskytter brukerne mot kapring av deres aktive økter og sikrer at sensitive cookies som sesjons ID-er ikke er tilgjengelige for angripere.

Utførelse:

Jeg startet med å høyreklikke på nettsiden og trakk på «Inspect». Deretter navigerte jeg til «Storage» og klikket på «Cookies». Her valgte jeg cookien knyttet til nettsiden, som i dette tilfellet er «PHPSESSID». HttpOnly-flagget er satt til «false», når den bør være satt til «true», slik at cookien kun kan leses av serveren og ikke av JavaScript i nettleseren.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	[REDACTED]	192.168.1.100	/	Session	35	false	false	None	Fri, 27 Sep 2024 23:5...

Finn ut mer:

- <https://owasp.org/www-community/HttpOnly>



Sårbarhet 16

Sårbarhet 16	Cookie without SameSite Attribute
Lav	En cookie som ikke har SameSite-attributt kan sendes mellom flere nettsider, noe som øker sannsynligheten for et angrep.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.100/>

Beskrivelse:

En cookie uten SameSite-attributtet kan sendes i forespørsler fra andre nettsider, noe som øker risikoen for CSRF-angrep. Dette åpner også for angrepsteknikker som timing angrep og Cross-Site Script Inclusion (XSSI) som lar en angriper få tilgang til eller manipulere sensitive data. SameSite-attributtet bidrar til å redusere disse truslene ved å kontrollere når cookien kan sendes til nettsiden.

Forebyggende tiltak:

For å forebygge angrep via informasjonskapsler altså cookies, bør SameSite-attributtet settes til «lax» eller «strict» for alle cookies. Dette reduserer risikoen for ulike angrep ved å begrense når nettlesere sender cookies til serveren. «Strict» gir høyest sikkerhet ved å sørge for at cookies kun sendes når brukeren navigerer direkte på nettsiden, imens «lax» gir en viss beskyttelse mot CSRF-angrep.

Utførelse:

Jeg startet med å høyreklikke på nettsiden og trakk på «Inspect», deretter navigerte jeg til «Storage» og klikket på «Cookies». Her kan man se detaljene for cookien og at SameSite-attributtet er satt til «None», som man ser på bildet til høyre.



Finn ut mer:

- <https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site>



Informasjon 1

Informasjon 1	Session Management Response Identified
Informasjon	Nettsiden mottok en respons med en sesjonsstyrings tok, noe som er for å administrere brukersesjoner. Det er en liten sannsynlighet at det utgjør en sikkerhetsrisiko.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.1/>

Beskrivelse:

Systemet har oppdaget at en respons inneholder et sesjonstoken, som brukes til å overvåke brukersesjoner, som når en bruker er logget inn. Sesjonstokener er viktig for å sikre at brukersesjoner forblir sikre. En angriper kan potensielt fange sesjon token og ta over brukerens aktive sesjon. Dette kan resultere i uautorisert tilgang til sensitiv informasjon.

Finn ut mer:

- <https://www.zaproxy.org/docs/desktop/addons/authentication-helper/session-mgmt-id>



Informasjon 2

Informasjon 2	Authentication Request Identified
Informasjon	En forespørsel er identifisert som en autentiseringsforespørsel. Dersom autentiseringsmetoden er satt til «Auto-Detect», kan autentisering automatisk endres basert på informasjonen som er gitt.

Identifisering:

Sårbarheten ble identifisert ved bruk av OWASP ZAP.

Området som ble påvirket:

<http://192.168.1.1/login.php>

Beskrivelse:

Dette indikerer at forespørselen er identifisert som en autentiseringsforespørsel. Når autentiseringen er satt til «Auto-Detect», justerer systemet automatisk autentiseringen basert på informasjonen som kommer med forespørselen. Autentiseringsprosessen kan manipuleres, noe som kan være en sikkerhetsrisiko. Som et resultat bør dette undersøkes nærmere for å sikre at autentiseringen fungerer som den skal og ikke kan brukes.

Finn ut mer:

- <https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/>