



Penetrasjonstest rapport September 2024

Skrevet av Monica Jakobsen for emnet ETH2100.

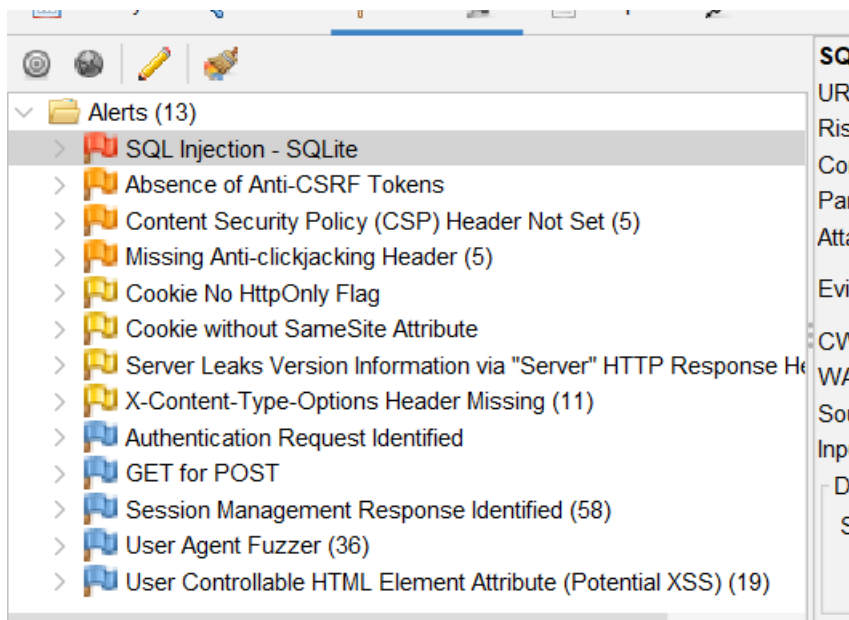
Sammendrag

IP: [REDACTED]	Beskrivelse: [REDACTED]
----------------	-------------------------

Denne penetrasjonstesten er utført mot webapplikasjoner [REDACTED] som eies av [REDACTED]. Testen er utført i sammenheng med arbeidskravet i emnet ETH2100 – Høsten 2024. Formålet med denne testen er å finne potensielle svakheter i webapplikasjonen og forbedre sikkerheten mot eventuelle ondsinnede angrep i fremtiden.

Testen blir gjennomført som en «White Box» test, det vil si at jeg som tester, har tilgang til alt jeg trenger, som tilgang til webserveren og kildekoden. Dette gjør det lettere for meg som en tester, å utføre en grundig og effektiv test, innenfor den tidsrammen jeg har fått tildelt.

Jeg kjørte en analyse på kildekoden og fant en sårbarhet der, som ga meg tilgang til alle passorder, mer detaljer om dette på egen side.



Innholdsfortegnelse

Alvorlighetsgrad.....	3
Rammeverk.....	4
Verktøy.....	5
Oppsummering av funn.....	6
Detaljert beskrivelse av funn.....	..
Sårbarhet 1.....	7
Sårbarhet 2.....	8
Sårbarhet 3.....	10
Sårbarhet 4.....	12
Sårbarhet 5.....	14
Sårbarhet 6.....	15
Sårbarhet 7.....	16
Sårbarhet 8.....	18
sårbarhet 9.....	19
Informativ.....	20
Oppsummering.....	21
Øvrige Kilder.....	21

Alvorlighetsgrad

HØY	Utgjør en stor akutt trussel mot serveren/systemet å bør håndteres så fort som mulig for å unngå innbrudd, tilganger og skader på serveren eller dataen.
MIDDELS	Utgjør en betydelig risiko, men er ikke noe som alle og enhver kan utnytte, krever innsats og vanligvis flere steg for å utgjør noe farlig.
LAV	Utgjør en lav risiko, enten er de for vanskelige til å utnytte, ellers så har de bare veldig liten innvirkning på systemet.
INFORMATIV	Informativ, gir informasjon om ting som kan være sårbarheter, men utgjør ikke noe trussel per nå, men bør informeres om.

Her kommer en oversikt over alvorlighetsgraden til funnene jeg fant under testing, og hvor mange som faller under hver kategori.

HØY	MIDDELS	LAV	INFORMATIV
2	4	3	5

Rammeverk.

Metodikk/Rammeverk:

I denne testen har jeg benyttet meg av «Penetration Testing Execution Standard (PTES)».

Kilder: [WSTG - Latest](#) | [OWASP Foundation](#)

[PTES Technical Guidelines - The Penetration Testing Execution Standard \(pentest-standard.org\)](#)

Denne modellen synes jeg passer godt til meg å hvordan jeg ønsker å utføre en pentest, den har klare rammeverk, delt inn i 7 faser, og jeg har forholdt meg til dette i denne testen, det finnes mange andre rammeverk, men siden vi har fokusert en del på owasp i forelesningene, så falt valget mitt på owasp.

- **Fase en – Pre-engagement Interaction.**

Fase en, er hvor jeg har fått beskjed om hva som skal gjøres, før testen begynner. I mitt tilfelle så henholder jeg meg til arbeidskrav-filen, og er inforstått med hva jeg skal gjøre, hva som skal testes og hva som er målet bak testen.

- **Fase to – Intelligence gathering.**

I denne fasen, så har jeg samlet data for og veilede meg i prosessen, her har jeg gjort meg kjent med serveren som skal testes, hva det er for noe og hva serveren gjør.

- **Fase tre – Threat modeling.**

i denne fasen har jeg ut ifra dataen jeg samlet inn i fase to, vurdert mulige trusler. Siden dette er en webapplikasjon, fokuserte jeg på sårbarheter knyttet til OWASP Top 10, som injeksjoner (SQL-injeksjoner, XSS) og andre mulige angrepsvektorer.

- **Fase fire – Vulnerability analysis.**

I denne fasen utførte jeg en sårbarhetsanalyse ved hjelp av verktøy som **Nmap**, **Nikto**, **Nessus**, og **Wireshark**. Jeg testet systemet for å finne åpne porter, utrygge tjenester, og kjente sårbarheter. Disse verktøyene hjalp meg med å identifisere potensielle svakheter på serveren.

- **Fase fem – Exploitation.**

I denne fasen har jeg testet de sårbarhetene jeg fant i forrige fase for å validere om de er reelle sårbarheter, testingen ble utført manuelt ved å utføre test-angrep mot serveren.

- **Fase seks – Post exploitation.**

Etter testingen av sårbarhetene, vurderte jeg hvilke tiltak som bør iverksettes for å sikre serveren mod videre angrep.

- **Fase syv – reporting.**

Den siste fasen er rapporteringen. Her har jeg skrevet om funnene mine, og tatt bilder og dokumentert testingen jeg har utført. Rapporten inneholder også anbefalinger for å forbedre sikkerheten basert på de funnene jeg gjorde

Liste over verktøy som ble brukt.

I denne testen så brukte jeg følgende verktøy:

- **NMAP** – NMAP brukes til å kartlegge webapplikasjonen og oppdage mulige åpne porter som vi som pentester(og ondsinnede angripere) kan utnytte. Med NMAP så benyttet jeg meg av enkelt scan og dypt scan, i tillegg så kjørte jeg et script scan for å finne sårbarheter.
- **NIKTO** – NIKTO er et helt enkelt verktøy for å scanne etter sårbarheter.
- **OWASP ZAP** – ZAP brukes til å scanne nettapplikasjoner mot sårbarheter, og mange andre ting som brute force angrep, man-in-the-middle angrep, men jeg benyttet den kun til å scanne for sårbarheter og brute force passord.
- **NESSUS** – NESSUS er det mest brukte scanningsverktøyet på markedet, Nessus tester hver eneste port på maskinen man har, og finner hvilke tjenester som er aktive og deretter sjekker disse tjenestene for å se om det finner noen sårbarheter.
- **WIRESHARK** – WIRESHARK brukes til å kunne se på nettverkstrafikken, ved å kunne se den kan man avdekke mange svakheter.
- **SQLmap** – Brukes for å teste for SQL-sårbarheter.
- **Fuzzer/Hydra** – Brute force verktøy som tester passorder.
- Jeg ville også bruke **Sslyze** og lignende verktøy, men serveren kjører kun på port 80, som ikke støttes av Sslyze.

I tillegg til dette så utførte jeg også manuell testing ved å bruke blant annet cross site scripting og SQL injections for å teste noen av sårbarhetene som testene avdekket.

NB! Har brukt både host maskin og VM i testingen/Scanningen så brukernavn vil være noe forskjellig.

Oppsummering av funn.

Her er en oppsummering av funnene som ble gjort under testingen, sortert etter alvorlighetsgrad.

SQL-Injeksjon-Sårbarhet 2	HØY
Åpen port 80 – Sårbarhet 3	HØY
Server tilgang – sårbarhet 4	MEDIUM
Cross site scripting – sårbarhet 5	MEDIUM
Anti clickjaking – sårbarhet 7	MEDIUM
uautorisert tilgang – sårbarhet 8	MEDIUM
Svake passord – sårbarhet 1	LAV
Server lekkasje – sårbarhet 6	LAV
Sensitivt innhold – sårbarhet 9	LAV
Informativ	Informativ

Root-tilgang med tildelt passord - Lav

Selv om dette ikke var en av de primære sårbarhetene som skulle undersøkes i denne testen, er det viktig å nevne. Jeg oppdaget at det tildelte brukernavnet og passordet for maskinen også ga meg **root-tilgang** når jeg brukte samme passord for root-kontoen.

Dette representerer en betydelig sikkerhetsrisiko, da det gir uautoriserte personer **administrativ tilgang** til systemet. Med root-tilgang har man full kontroll over systemet og kan utføre endringer som kan kompromittere systemets integritet og sikkerhet. Dette kan inkludere endringer i konfigurasjonsfiler, installasjon av ondsinnet programvare, eller å låse ut legitime brukere.

Anbefaling:

Det anbefales sterkt å sette egne og engangsbruket, sikre passord for root-kontoen og andre brukere.

```
~$ su  
Password:   
root@hostname -I  
root@
```


message	Administrator
site	user
user	

Administrator	
Host	
User	
Password	
Select_priv	
Insert_priv	
Update_priv	
Delete_priv	
Create_priv	

alle brukerne.

Åpen port 80 http – Sårbarhet 3.

Åpen port 80 - Høy

Ved hjelp av verktøyene Nessus, Nikto og Nmap ble det klart at port 80 er åpen, og dette er den eneste porten som ble oppdaget i skanningene. Port 80 brukes som standard for ukryptert HTTP-trafikk, noe som medfører betydelige sikkerhetsrisikoer.

Ved bruk av Wireshark kunne jeg enkelt overvåke og analysere trafikken som ble sendt over port 80. Under denne analysen oppdaget jeg at både brukernavn og passord blir sendt som klartekst i HTTP-forespørsler. Dette var lett å fange opp, spesielt da følgende kolonne fanget min umiddelbare oppmerksomhet:

- **768 POST/login.php HTTP/1.1**

Ved å klikke på denne forespørselen kunne jeg lese både brukernavnet og passordet som ble sendt til serveren. Jeg testet med følgende verdier:

- **username = TESTER**
- **password = ARBEIDSKRAV**

Disse opplysningene ble synlig som klartekst i datapakken.

Bruken av ukryptert HTTP via port 80 utsetter serveren for man-in-the-middle-angrep. Alle som avlytter nettverkstrafikken kan enkelt hente ut sensitive data som brukernavn og passord, noe som kan gi dem uautorisert tilgang til systemet.

Dette regnes som en kritisk sårbarhet fordi sensitive opplysninger, inkludert innloggingsdata, blir eksponert i klartekst. Dette kan utnyttes av angripere til å få full tilgang til serveren, og kan kompromittere hele serveren.

Anbefaling:

For å sikre systemet bør HTTPS (port 443) brukes for å kryptere all trafikk mellom klienter og serveren, noe som vil forhindre at sensitiv informasjon fanges opp av uautoriserte personer.

Her ser dere passord og brukernavn i klartekst.

```
01f0 0a 4f 72 69 67 6e 6e 3a 20 68 74 74 70 3a 2f 2f -Origin: http://
0200 31 39 32 2e 31 36 38 2e 31 31 39 2e 31 33 35 0d
0210 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 -Connect ion: kee
0220 70 2d 61 6c 69 76 65 0d 0a 52 65 66 65 72 65 72 p-alive Referer
0230 3a 20 68 74 74 70 3a 2f 2f 31 39 32 2e 31 36 38 /login.p
0240 2e 31 31 39 2e 31 33 35 2f 6c 6f 67 69 6e 2e 70 hp Cook ie: PHPS
0250 68 70 0d 0a 43 6f 6f 6b 69 65 3a 20 50 48 50 53 ESSID=pl oq42ehk9
0260 45 53 53 49 44 3d 70 6c 6f 71 34 32 65 68 6b 39 f00iq2fn herq0joe
0270 66 30 30 69 71 32 66 6e 68 65 72 71 30 6a 6f 65 f00iq2fn herq0joe
0280 0d 0a 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75 Upgrad e-Insecu
0290 72 65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a re-Reque sts: 1
02a0 50 72 69 6f 72 69 74 79 3a 20 75 3d 30 2c 20 69 Priority : u=0, i
02b0 0d 0a 0d 0a 75 73 65 72 6e 61 6d 65 3d 54 45 53 user name=TES
02c0 54 45 52 26 70 61 73 73 77 6f 72 64 3d 41 52 42 TER&pass word=ARB
02d0 45 49 44 53 4b 52 41 56 26 63 73 72 66 5f 74 6f EIDSKRAV &csrf_to
02e0 6b 65 6e 3d 64 38 65 65 66 62 61 31 39 39 32 63 ken=d8ee fba1992c
02f0 38 62 39 30 65 39 35 39 35 36 31 62 66 30 63 61 8b90e959 561bf0ca
02ff 30 65 38 26 6c 6f 67 69 6e 3d 6c 6f 67 69 6e 0e98&log in=logi
```

```
No. 1 Destination Protocol Length Info
TCP 54 45198 → 80 [ACK] Seq=705 Ack=2684 Win=131328 Len=0
TCP 54 [TCP Retransmission] 45180 → 80 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
TCP 54 45180 → 80 [RST, ACK] Seq=2 Ack=1 Win=0 Len=0
TCP 54 45198 → 80 [FIN, ACK] Seq=705 Ack=2684 Win=1049600 Len=0
TCP 66 45284 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
TCP 66 80 → 45284 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
TCP 54 45284 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
HTTP 768 POST /login.php HTTP/1.1 (application/x-www-form-urlencoded)
TCP 60 80 → 45284 [ACK] Seq=1 Ack=715 Win=30720 Len=0
TCP 54 [TCP Retransmission] 45198 → 80 [FIN, ACK] Seq=705 Ack=2684 Win=1049600 Len=0
TCP 54 [TCP Retransmission] 45198 → 80 [FIN, ACK] Seq=705 Ack=2684 Win=1049600 Len=0
TCP 54 [TCP Retransmission] 45198 → 80 [FIN, ACK] Seq=705 Ack=2684 Win=1049600 Len=0
TCP 1514 80 → 45284 [ACK] Seq=1 Ack=715 Win=30720 Len=1460 [TCP segment of a reassembled PDU]
HTTP 1276 HTTP/1.1 200 OK (text/html)
TCP 54 45284 → 80 [ACK] Seq=715 Ack=2683 Win=1049600 Len=0
TCP 54 [TCP Retransmission] 45198 → 80 [FIN, ACK] Seq=705 Ack=2684 Win=1049600 Len=0
TCP 60 80 → 45284 [FIN, ACK] Seq=2683 Ack=715 Win=30720 Len=0
TCP 54 45284 → 80 [ACK] Seq=715 Ack=2684 Win=1049600 Len=0
```

```
Ethernet II, Src: VMware 00:00:f
Internet Protocol Version 4, Src
Transmission Control Protocol, Src Port: 45284, Dst Port: 80, Seq: 1, Ack: 1, Len: 714
Hypertext Transfer Protocol
URL Form URL Encoded: application/x-www-form-urlencoded
Form item: "username" = "TESTER"
Key: username
Value: TESTER
Form item: "password" = "ARBEIDSKRAV"
Key: password
Value: ARBEIDSKRAV
Form item: "csrf_token" = "d8eeffa1992c8b90e959561bf0ca0e98"
Key: csrf token
Value: d8eeffa1992c8b90e959561bf0ca0e98
Form item: "login" = "login"
Key: login
Value: login
```

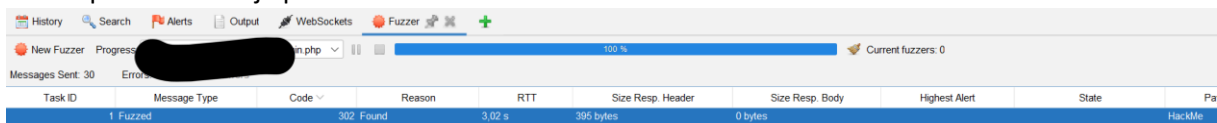
Server tilgang – Sårbarhet 4.

Servertilgang - Medium

Jeg klarte å oppnå tilgang til brukerkontoene på serveren ved å gjennomføre et målrettet brute force-angrep. Fordi brukernavnene var offentlig tilgjengelige, kunne jeg effektivt angripe kjente brukernavn på serveren, noe som resulterte i vellykket innlogging.

Jeg hadde allerede passorder fra kildekoden, men jeg valgte å teste at dette funker ved å kjøre noen passordfiler mot brukernavn, og fikk match, riktig passord var det eneste som ga 302 found kode på fuzzeren. Så lenge man har tilgang til brukernavnene, så kan man veldig enkelt utføre målrettet angrep mot spesifikke brukere å få tilgang til serveren.

Finner passord ved hjelp av fuzzer:




The screenshot shows a Fuzzer tool interface with a progress bar at 100%. Below the progress bar, a table displays the results of the fuzzing process. The first row indicates a 'Fuzzed' task with a '302 Found' reason, an RTT of 3.02 s, and a size of 395 bytes for the response header and 0 bytes for the response body. The state is marked as 'Medium'.

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Pa
1	Fuzzed	302 Found	302 Found	3.02 s	395 bytes	0 bytes	Medium		

Testet forskjellige funksjoner på forskjellige brukere for å se at alt funker:

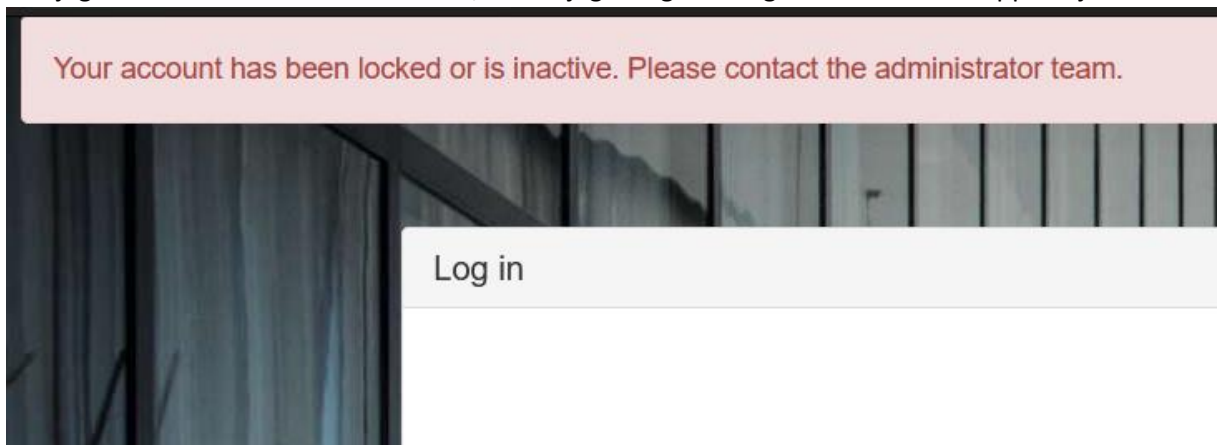
Tester å endre funksjoner/informasjon på brukerne:



The screenshot shows a user management interface with two rows of user data. The first row shows a user named 'Samuel' with a status of 'Inactive'. The second row shows a user named 'Samuel' with a status of 'Active'. A green message bar above the second row states 'The user is enabled successfully!'.

Name	First Name	Last Name	Role	Created At	Status	Action
Samuel			Collaborator	2021-11-05 05:52:24	Inactive	
Samuel			Collaborator	2021-11-05 05:52:24	Active	

Når jeg endrer status fra aktiv til inaktiv, så kan jeg stenge ute legitime brukere fra applikasjonen.



Tester å sende penger:

Sendte først en forespørsel på 500EUR.



The screenshot shows a table titled 'My Expense reports'. The table has columns for Date, Amount, Comment, Status, and Action. The first row shows a report for 2024-09-21 with an amount of 500 € and a status of 'Opened'.

Date	Amount	Comment	Status	Action
2024-09-21	500 €	Private	Opened	

Logget deretter inn på Financial approver bruker og godkjente forespørselen – og sendte betalingen til brukeren.



The screenshot shows a table with columns for Date, Collaborator's name, Amount, Comment, Status, and Action. The first row shows a report for 2024-09-21 with an amount of 500 € and a status of 'Submitted'.

Date	Collaborator's name	Amount	Comment	Status	Action
2024-09-21		500 €	Private	Submitted	

Are you sure to want to send for payment this expense report ?

Yes

No

Collaborators

Date	Collaborator's name	Amount	Comment	Status	Action
2024-09-21		500 €	Private	Validated	 

The expense report is sent for payment successfully !

Date	Amount	Comment	Status
2024-09-21	500 €	Private	Sent for payment

Så enkelt kunne jeg få tilgang til brukerne å utføre handlinger jeg egentlig ikke burde få til.

Anbefalinger:

Implementer Tofaktorautentisering (2FA) for å styrke innloggingssikkerheten.

Unngå bruk av enkle og forutsigbare passord som lett kan gjettes.

Cross site scripting - Sårbarhet 5.

XSS Injeksjoner – Middels.

Webserveren er sårbar for cross site scripting. Cross site scripting, også kjent som XSS er en type injeksjon der en ondsinna person kan injisere nettstedet med ondsignede kode på en ellers trygg nettapplikasjon. Dette foregår vanligvis ved at en hacker eller andre personer setter inn JavaScript i ett nettsted som ikke validerer eller renser brukerinput, dette gjelder ofte på nettsteder som har kommentarfelter eller gjestebøker.

Når en annen uskyldig bruker da åpner nettstedet som har blitt kompromittert med injeksjoner, så vil da det scriptet som er blitt injisert, kjøre på den uskyldige brukeren sin maskin. Eksempler på hva xss brukes til:

- Stjele Cookie informasjon – det tillater en ondsinna person å ta over en bruker sin sesjon eller bruker.
- Få tilgang til filer
- Sende virus/Trojaner
- Sende brukeren til andre nettsteder enn det hen hadde tenkt til.

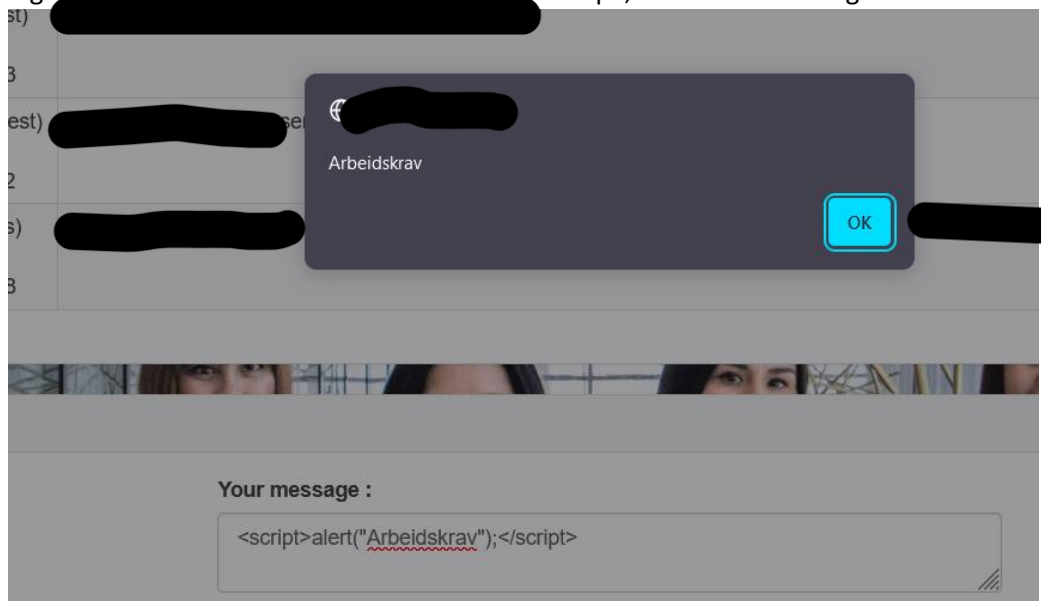
Et XSS angrep kan utføre mye skade, og kan brukes til mye forskjellig.

Jeg testet et helt enkelt angrep ved å sende en alert, det er helt ufarlig og skader ikke systemet/Serveren på noen som helst måte.

Jeg sendte følgende script inn i meldingene:

- `<script>alert(«Arbeidskrav»);</script>`

Det trigger deretter en alert på nettstedet for meg og alle andre som logget inn på serveren, det ble lagret i serveren å eneste måten å ikke trigge det på, er å slette meldingen min.



Anbefaling:

Legg til Content Security Policy (CSP) – Det er et ekstra lag med beskyttelse mot en rekke angrep, inkludert XSS. Webserveren mangler dette.

Sørg for at all brukerinputdata blir validert før de blir lagt til på serveren.

Server lekkasje – sårbarhet 6.

Server versjon – Lav.

Webserveren avslører hvilken server den kjører på, noe som gir enhver person muligheten til å finne ut hvilken programvare som benyttes. I vårt tilfelle kjører applikasjonen på «Apache 2.4.38 (Debian)», som er utdatert. Når serverinformasjonen er tilgjengelig, kan ondsinnede aktører enkelt søke etter kjente sårbarheter.

Apache 2.4.38 (Debian) er svært udatert og har flere kjente sårbarheter som kan utnyttes. Et raskt Google-søk avdekker følgende:

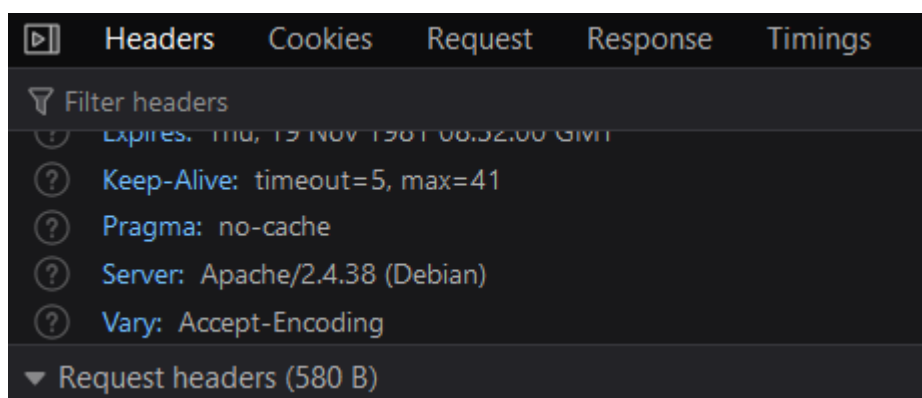
- **CVE-2019-0215:** Denne sårbarheten involverer en feil i mod_ssl. Når klientsertifikatverifisering brukes per plassering med TLSv1.3, kan det tillate en klient å omgå de konfigurerte tilgangskontrollbegrensningene. (Kilde: [CVE - CVE-2019-0215 \(mitre.org\)](#))
- **CVE-2023-27522:** En HTTP Response Smugling-sårbarhet i Apache HTTP Server via mod_proxy_uwsgi. Dette er en type angrep der en angriper kan utnytte måten HTTP-responser håndteres av servere og proxyer. (Kilde: [CVE - CVE-2023-27522 \(mitre.org\)](#))

i tillegg:

[Apache HTTP Server 2.4 vulnerabilities - The Apache HTTP Server Project](#)

En liste som viser alle de forskjellige kjente sårbarhetene.

Anbefaling: Oppdater serveren.



Anti clickjacking - Sårbarhet 7.


Anti clickjacking – **Medium**.

Nettserveren mangler beskyttelse mot clickjacking-angrep, også kjent som UI-redirigering. Clickjacking er en teknikk der en angriper plasserer et usynlig lag (for eksempel en iframe) over en nettside for å lure brukeren til å klikke på knapper eller lenker uten deres viten. Klikkene blir dermed hijacket, slik at handlingene som brukeren mente å utføre på den opprinnelige nettsiden, i stedet blir utført på en angriperes server.

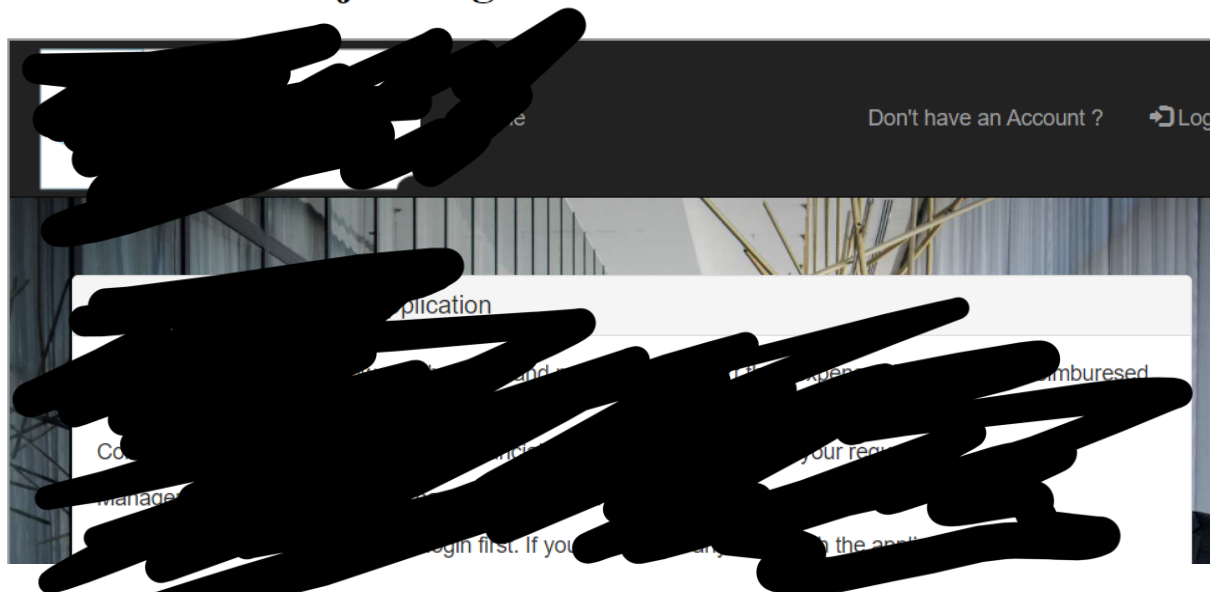
For å teste denne sårbarheten opprettet jeg et enkelt HTML-dokument som inneholdt en iframe med IP-adressen til serveren. Ved å laste inn dette dokumentet i en nettleser fikk jeg tilgang til serveren, uten at det oppstod noen feilmeldinger eller blokkeringer, noe som bekrefter at serveren er sårbar for clickjacking.

(Kilde: [Clickjacking | OWASP Foundation](#))

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>Tester for Clickjacking sårbarhet</h1>
10  <iframe src="127.0.0.1/index.php" width="800" height="600"></iframe>
11 </body>
12 </html>
```



Tester for Clickjacking sårbarhet



Slik skal det se ut hvis siden er beskyttet mot clickjacking:
Forsøket mitt ble avvist.



www.google.no avviste tilkoblingsforsøket.

Anbefaling:

Legg til Content-Security-Policy.

Uautorisert tilgang - Sårbarhet 8.

Uautorisert tilgang til sensitivt innhold på server – **Medium**.

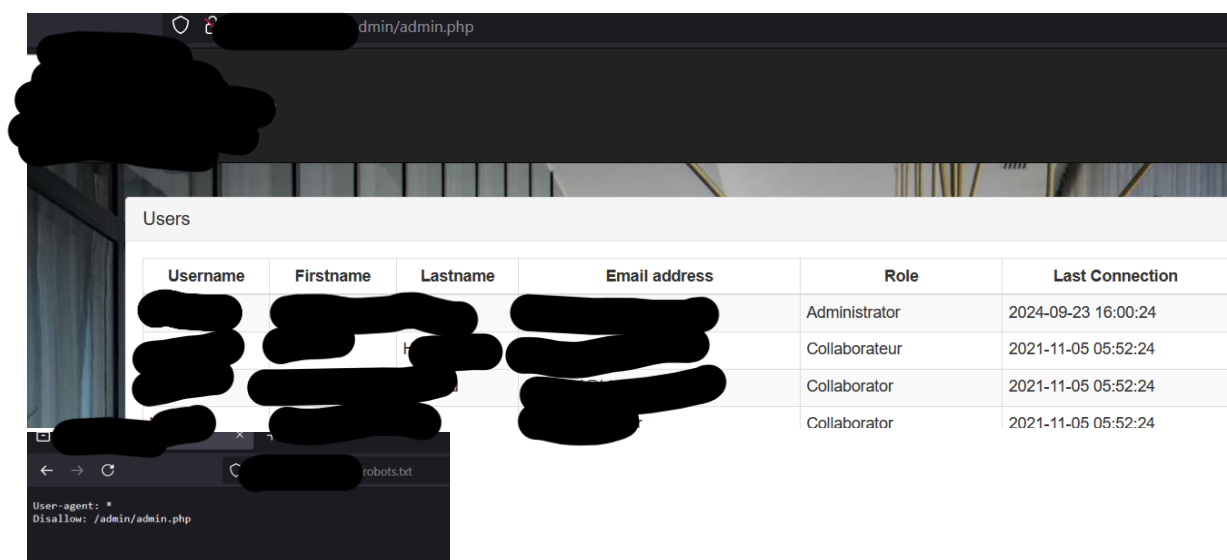
Serveren mangler tilstrekkelig beskyttelse mot uautorisert tilgang til sensitivt innhold. Ved å navigere direkte til følgende URL-er:

- [redacted]/admin/admin.php
- [redacted]robots.txt
- [redacted]config/setup.php

fikk jeg tilgang til innhold som ikke burde vært tilgjengelig uten riktig autentisering. **Admin-filen** inneholder sensitiv informasjon om brukerne, inkludert navn, e-postadresser, brukernavn og lignende data. **Robots.txt-filen** inneholder heldigvis lite informasjon, men det er fortsatt en fil som ikke bør ligge åpen for alle, da den kan avsløre viktige ressurser på serveren. **Setup.php** inneholder informasjon om databasen. Det vil også være enklere å brute force passorder når man vet brukernavnene.

Når en fil som admin.php eksponerer informasjon som e-postadresser, er det en betydelig risiko for at en angriper kan bruke denne informasjonen til å utføre målrettede phishing-angrep. Phishing er en form for angrep der angriperen utgir seg for å være en du stoler på, som en bank eller sjefen din, sender en e-post som forsøker å lure offeret til å dele sensitive opplysninger, som passord eller betalingsinformasjon.

Når e-postadresser og brukernavn er offentlig tilgjengelige, blir det enkelt for en angriper å gjennomføre et målrettet phishing-angrep, hvor de i tillegg kan bruke ofrenes navn til å finne mer informasjon om dem. Dette forsterker risikoen og gjør det enklere for angriperen å fremstå troverdig.



Anbefaling:

Fjern disse filene, og legg de bak en vegg som krever autentisering for å få tilgang.

Kritiske filer – Lav

Under en gjennomgang av kildekoden har jeg avdekket at svært sensitiv informasjon er tilgjengelig uten tilstrekkelig beskyttelse. Spesielt er det funnet hardkodete brukernavn og passord som ikke er hashet. Dette representerer en betydelig sikkerhetsrisiko, da enhver person med tilgang til serveren også kan få tilgang til disse filene. Når passord er hardkodet inn i kildekoden, blir de vanskelige å endre, noe som kan føre til at de forblir i bruk lenger enn anbefalt.

Denne sårbarheten kan potensielt utnyttes av uautoriserte aktører, som kan bruke informasjonen til å få tilgang til serveren og kompromittere hele systemet. Filene er plassert under `/var/www/html/config`, og mangelen på beskyttelse øker risikoen for at de ved en feil kan bli eksponert og lastet opp til en server, noe som vil gi uvedkommende mulighet til å få tilgang til sensitive data.

Anbefalinger:

Unngå å bruke hardkodet passord i kildekode.

Implementer nødvendige tilgangsrestriksjoner for å beskytte sensitive filer



Her kommer informativ informasjon som ble fanget opp via OWASP Zap og Nessus.

Informativ – 5Stk

- **Autentiseringsforespørsel identifisert**

Den angitte forespørselen er identifisert som en autentiseringsforespørsel. Feltet 'Other Info' inneholder en liste med key=value-linjer som identifiserer relevante felter. Hvis forespørselen befinner seg i en kontekst hvor autentiseringsmetoden er satt til "Auto-Detect", vil denne regelen endre autentiseringen til å matche den identifiserte forespørselen.

- **GET for POST**

En forespørsel som opprinnelig ble sendt som en POST, ble også akseptert som en GET. Dette problemet representerer ikke nødvendigvis en sikkerhetssvakhet i seg selv, men det kan forenkle andre angrep. For eksempel, hvis den opprinnelige POST-forespørselen er sårbar for Cross-Site Scripting (XSS), kan dette funnet indikere at et forenklet (GET-basert) XSS-angrep også kan være mulig.

- **Sesjonsstyringsrespons identifisert**

Den angitte responsen er identifisert som inneholdende en sesjonsstyringstoken. Feltet 'Other Info' inneholder en liste med headertokens som kan brukes i den header-baserte sesjonsstyringsmetoden. Hvis forespørselen er i en kontekst hvor sesjonsstyringsmetoden er satt til "Auto-Detect", vil denne regelen endre sesjonsstyringen til å bruke de identifiserte tokenene.

- **Brukeragent-Fuzzer**

Sjekker etter forskjeller i respons basert på modifisert User Agent (for eksempel mobilnettsteder, tilgang som en søkemotorcrawler). Sammenligner responsens statuskode og hashkode for responsinnholdet med den opprinnelige responsen.

- **Brukerkontrollerbart HTML-elementattributt**

Denne sjekken ser på brukerleverte input i spørringsstrengparametere og POST-data for å identifisere hvor visse HTML-attributtverdier kan kontrolleres. Dette gir "hotspot"-deteksjon for XSS (cross-site scripting), som krever videre vurdering av en sikkerhetsanalytiker for å avgjøre om det er utnyttbart.

(Kilde: Informativ feltets Deskripsjon i ZAP. Har direkte oversatt informasjonen til norsk!).

Oppsummering

Alle disse sårbarhetene ble funnet ved hjelp av en rekke verktøy og validert manuelt.

Jeg fant alle brukernavn og passord lett tilgjengelig i kildekoden, men brukte også Fuzzer til og brute force passord ved hjelp av tekstfiler.

Noen sårbarheter/Exploits har jeg ikke testet da jeg ikke er flink nok enda, det gjelder bla CVE-er til Apache serveren, jeg har ikke manuelt testet disse kjente exploitene, men vet at de er ekte og de er rapportert.

Har ikke gjort noen endringer på serveren annet enn det jeg testet, så har ikke slettet eller rotet til noen innstillinger som må fikses, har heller ikke lagt til filer for å få flere sårbarheter som det ble nevnt i forelesning at noen gjorde.

Øvrige kilder:

Kilde til forsidebildet: [Bildeskaper \(bing.com\)](https://bing.com)

Kilde til Logoen som jeg har brukt: [Bildeskaper \(bing.com\)](https://bing.com)
Begge bildene er AI-Generert med microsoft sin kreative AI.

Lagde bildene og logo ved å be BingAI om en passende logo og forside som har med pentesting å gjøre.