

## Sensors rapport for Penetrasjonstesting av webapplikasjoner for South-Dyersville industries

Prosjektgruppe: 3

Deltakere: Fredrik Bjerkø  
Krister Arntsberg  
Eivind Børstad  
Gard Klemetsen

Utgivelsesdato: 23.11.2017

Begrensning: Åpen

### Sammendrag

Denne rapporten beskriver hvordan prosjektgruppe 3 har gått frem for å penetrasjonsteste webapplikasjoner for den fiktive bedriften South-Dyersville industries i forbindelse med eksamen i DA-PEN3900. Det ble utført en mest mulig realistisk pentest med bruk av både automatiske verktøy og manuell testing, som avslørte en rekke svakheter på nettsidene.

I tillegg til denne rapporten har det blitt laget en administrativ og en teknisk rapport, som begge ligger som vedlegg nederst.

# Innholdsfortegnelse

<b>Innledning</b>	<b>2</b>
<b>1 Verktøy</b>	<b>3</b>
1.1 Nmap	3
1.2 Google Chrome	3
1.3 BurpSuite	3
1.4 Firefox og FoxyProxy	4
<b>2 Metodikk</b>	<b>5</b>
<b>3 Arbeidsfordeling</b>	<b>7</b>
<b>4 Resultater</b>	<b>8</b>
<b>Konklusjon</b>	<b>9</b>
<b>Litteraturliste</b>	<b>10</b>
<b>Vedlegg</b>	<b>11</b>

# Innledning

Den 4. november 2017 ble prosjektgruppen tildelt oppdraget å utføre en mest mulig realistisk pentest mot webapplikasjonene i det utpekte målet. IP-adressen det skulle testes mot var 10.250.200.3 - en privat adresse inne på skolens labnett. Disse nettsidene skulle forestille nettsider for den fiktive bedriften South-Dyersville industries. Fristen for prosjektet var satt til 30. november.

Prosjektgruppen består av Fredrik Bjerkø, Krister Arntsberg, Eivind Børstad og Gard Klemetsen. Ingen av deltakerne hadde noen erfaring med pentesting fra før av, med unntak av det som hadde blitt lært tidligere i kurset. Derfor ble utførelsen av prosjektet like mye å lese seg opp på de forskjellige aspektene innenfor datasikkerhet, som å utføre selve pentesten.

Gjennom denne rapporten forklares hvordan det ble gått frem for å teste siden på en mest mulig grundig og strukturert måte. I kap. 1 gjennomgås det hvilke verktøy som ble brukt for å finne frem til sårbarheter. Kap. 2 beskriver hvordan utførelsen av selve testene foregikk. Dette inkluderte bruk av både de automatiske verktøyene som ble omtalt i kap. 1, og manuell testing gjennom en nettleser. Videre gir kap. 3 en oversikt over arbeidsfordelingen blant deltakerne, før det i kap. 4 reflekteres noe over resultatene som ble funnet.

Det ble i tillegg til denne rapporten skrevet to andre, nemlig en administrativ og en teknisk rapport. Disse er i motsetning til denne skrevet som om dette var en ekte pentest, og som om webapplikasjonene var ute på det offentlige nettet. Begge disse rapportene er lagt som vedlegg nederst i denne, og vil i noen grad bli referert til. Det er likevel ikke nødvendig å lese de to andre for å få sammenheng av innholdet i denne rapporten.

Den administrative rapporten (vedlegg 1) er skrevet for ledelsen i bedriften, på en slik måte at alle kan forstå den. Det betyr at den ikke inneholder tekniske begreper. Målet med denne er å få ledelsen til å forstå at sårbarhetene må fikses, og at dette er verdt å bruke ressurser på. Den tekniske rapporten (vedlegg 2) er skrevet for tekniske personer, bl.a. de som skal rette opp i sikkerhetshullene. Denne inneholder tekniske begreper, beskriver sårbarhetene i større detalj, og gir råd om hvordan de kan fikses.

Gjennom hovedkapitlene i denne rapporten vil leseren få et godt overblikk over hvordan pentesten ble utført, uten å bli presentert for mange detaljer over hvilke sårbarheter som faktisk ble funnet.

# 1 Verktøy

Det har blitt utnyttet et rekke verktøy for å gjennomføre penetrasjonstesten. Hvert verktøy har blitt tildelt et underkapittel, der det gis en kort innføring av hva verktøyet er for noe, og hvordan gruppen har benyttet seg av det. Flere detaljer om hvordan verktøyene har blitt benyttet under testen, vil også bli gitt under kapittel 2, metodikk. For mer grundig beskrivelse av de forskjellige verktøyene, har det blitt vedlagt anbefalte referanser.

## 1.1 Nmap

Nmap er en nettverksskanner som brukes for å oppdage maskiner på et nettverk, og hvilke servicer/porter de har åpne. Det kan derfor sies at Nmap bygger et slags kart (map) over nettverket. For mer informasjon om Nmap anbefales [1].

Gjennom prosjektet har Nmap blitt brukt for å få oversikt over hvilke porter som var åpne på serveren. Det ble oppdaget at det hovedsakelig var tre porter av høy interesse, noe det kan leses mer om i begynnelsen av kap. 2.

## 1.2 Google Chrome

Nettleseren Google Chrome, som nødvendig trenger noen introduksjon, ble benyttet for å utføre de manuelle testene. Informasjonsinnsamling ble også gjort gjennom denne nettleseren.

## 1.3 BurpSuite

BurpSuite, eller bare Burp, er et grafisk verktøy med mange forskjellige funksjonaliteter som benyttes i forbindelse med pentester. Programmet tar utgangspunkt i en nettside ved å fange opp forespørselen som blir sendt når man prøver å laste inn siden i en nettleser. Hvordan dette settes opp forklares i kap. 1.4. For flere detaljer om BurpSuite, anbefales følgende kilder: [2][3].

BurpSuite ble benyttet som hovedverktøy i forbindelse med automatiske tester. Mange av de manuelle testene ble også utført på bakgrunn av resultater funnet gjennom Burp. Verktøyene i Burp som ble mest brukt var "Spider", "Scanner" og "Intruder". "Spider" benyttes for å finne alle de forskjellige undersidene på en nettside. "Scanner" tester deretter disse nettsidene automatisk mot en rekke vanlige sårbarheter. "Intruder" brukes for å teste mer detaljert dersom man tror det har blitt funnet en sårbarhet.

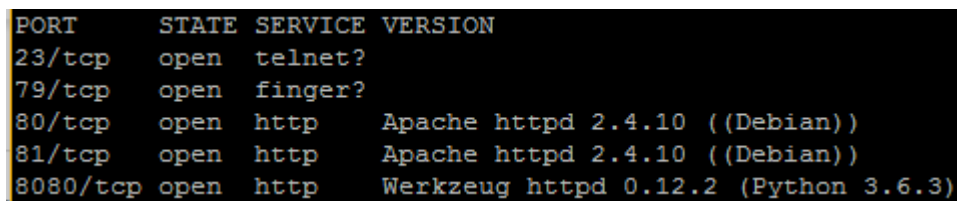
## 1.4 Firefox og FoxyProxy

Som nevnt i kap. 1.3 ble BurpSuite brukt slik at den først fanget opp en HTTP-forespørsel, og deretter begynte å utføre tester mot denne siden, og andre sider under samme IP-adresse og port. For å la Burp fange opp en nettside ble nettleseren Firefox benyttet, og programtillegget FoxyProxy. Sistnevnte er et verktøy som tillater at f.eks. BurpSuite plukker opp HTTP-forespørselen på vei mot måltjeneren. Mer om FoxyProxy kan leses her: [4].

## 2 Metodikk

Dette kapittelet gir et innblikk i hvordan det ble jobbet underveis i prosjektgjennomføringen. Det henger tett sammen med verktøyene i kap. 1, og arbeidsfordelingen i kap. 3, men kan leses selvstendig.

Det aller første som ble gjort var å utføre en portskan med bruk av Nmap, som det kan leses om i kap. 1.1. Se figur 2.1 for resultatet av denne skanningen.



PORT	STATE	SERVICE	VERSION
23/tcp	open	telnet?	
79/tcp	open	finger?	
80/tcp	open	http	Apache httpd 2.4.10 ((Debian))
81/tcp	open	http	Apache httpd 2.4.10 ((Debian))
8080/tcp	open	http	Werkzeug httpd 0.12.2 (Python 3.6.3)

**Figur 2.1 - Resultat fra Nmap**

Som man kan se var det fem porter som var åpne. Da gruppens oppdrag var å penteste webapplikasjoner, betyr det at port 80, 81 og 8080 var de aktuelle målene for testing. Ved å entre disse nettsidene via en nettleser ble det funnet ut at de var ganske forskjellige. Sårbarheter skulle testes mot alle tre.

I begynnelsen ble det utført en del ustrukturerte tester. Disse var stort sett manuelle, og ble gjort for å bli litt kjent med nettsidene det skulle testes mot. Gjennom dette fikk man et innblikk av oppdragets omfang, hvor sårbar siden var, og hvilke typer sikkerhetshull som var særlig til stede. Denne prosessen resulterte i funn av flere svakheter. Siden en slik gjennomføring ikke er strukturert, må man likevel regne med at det er en del potensielle sikkerhetshull man overser, så denne fremgangsmåten kan ikke brukes gjennom hele pentesten.

Neste fase gikk ut på i større grad å gå over til automatiske verktøy, og i vårt tilfelle hovedsakelig BurpSuite, se kap. 1.3. Det ble benyttet Burp sin søkerobot (Spider/Crawler på engelsk) for å få oversikt over de forskjellige sidene på hver port. Deretter ble det latt Burp gjøre en aktiv skanning av alle disse undersidene, der det ble testet mot en rekke av de vanligste sårbarhetene. Når Burp fant en sårbarhet, ble denne senere testet i mer detalj av prosjektgruppen. Dette innebar både manuell testing vha. en nettleser, og bruk av andre verktøy i Burp som "Intruder". Alle sårbarheter ble dokumentert gjennom tekst og bilder, som senere skulle brukes i den administrative og tekniske rapporten.

Samtidig med disse automatiske testene ble det gjennomført flere manuelle tester, og nettsidene ble gjennomført nøye. Dette var for å finne sårbarheter som det er

vanskelig for et automatisk verktøy som Burp å oppdage. Et eksempel er sensitiv informasjon som ligger tilgjengelig for alle, noe det ble funnet mye av.

Det største problemet ved gjennomføringen av pentesten, i tillegg til litt begrenset med tid, var at tjeneren bak nettsidene var litt treg og ustabil. Spesielt på port 8080 var det ofte problemer med tilkoblingen, noe som gjorde at det var svært tidskrevende å gjennomføre tester her. Burps automatiske tester fikk også store problemer mot denne porten, og kunne ikke gjennomføres ordentlig. Flere forsøk ble gjort, men testene kom aldri ordentlig i mål. Prosjektgruppen tror likevel de fleste sårbarhetene her har blitt avdekket, men noe kan ha passert uoppdaget på grunn av dette.

Etter at de fleste testene var gjennomført, begynte rapporten å skrives, samtidig som det ble utført flere manuelle tester i forsøk på å finne flere sårbarheter.

Underveis i testingen fikk gruppen blant annet mulighet til å kjøre kommandoer på maskinen gjennom såkalte "OS-injeksjoner". Dette, og enkelte andre sårbarheter, gjorde det mulig å gjøre endringer på maskinen under pentesten. Det ble hovedsakelig forsøkt med harmløse endringer, som f.eks. å opprette en tom mappe. Etter at disse testene var gjennomført ble det så godt som mulig forsøkt å fjerne alle slike spor. Målet er selvsagt at maskinene ser ut som da testen ble startet, men det er mulig det ligger igjen enkelte spor som ikke har blitt slettet. I så fall er dette kun objekter av type tomme mapper og filer, og ikke noe som utgjør en sikkerhetstrussel for systemet.

### 3 Arbeidsfordeling

Prosjektgruppen har ikke hatt noen veldig konkret fordeling av oppgaver. Deltakerne har jobbet tett sammen under hele gjennomføringen, og fordelt oppgaver etter hva man har lyst til å jobbe med. Generelt kan det likevel sies litt om hva hver person har jobbet mest med:

- Krister: BurpSuite og manuell testing
- Gard: Manuell testing og dokumentering
- Fredrik: Manuell testing gjennom BurpSuite
- Eivind: BurpSuite, manuell testing og dokumentering

Dette er altså kun en veiledning, og alle medlemmene har jobbet innenfor alle delene av prosjektet. Fredrik var opprinnelig i en annen prosjektgruppe, men da den ble oppløst ble han med i prosjektgruppe 3. Han har likevel vært med på mesteparten av prosessen.

Når det gjelder rapportskrivningen har det også vært fordelt, men Gard og Eivind har tatt hoveddelen av dette arbeidet. Hovedansvaret for hver av rapportene har vært fordelt slik:

- Sensors rapport: Eivind
- Administrativ rapport: Gard og Eivind
- Teknisk rapport: Gard og Eivind

Arbeidsfordelingen har fungert fint i den forstand at alle har fått jobbe med det de ønsker, og samtidig har alle nødvendige arbeidsoppgaver blitt fullført. Å fordele arbeidet på denne måten har også gjort at alle har tilegnet seg kunnskaper om alle aspektene av en pentest.



## 4 Resultater

For detaljert oversikt over resultater henvises det til den administrative rapporten (vedlegg 1), og for detaljerte beskrivelser henvises det til den tekniske rapporten (vedlegg 2). Her vil det i stedet bli drøftet kort rundt resultatene som ble funnet.

Det ble funnet en betydelig mengde kritiske sårbarheter i webapplikasjonene. Ved funn av slike feil hadde det med stor sannsynlighet vært mulig å utnytte disse videre. Et eksempel er at OS-injeksjoner antakelig kunne bli benyttet til å gjennomføre et vidt spekter av angrep mot systemet. Dette ble likevel kun valgt å gjøre i liten grad, da sårbarheten allerede var oppdaget, og oppdraget i hovedsak går ut på nettopp å finne sårbarhetene. Videre utnyttelse av samme sårbarhet ville vært svært ressurskrevende, samtidig som det ikke hadde avdekket flere sårbarheter - kun vist hva den allerede oppdagede sårbarheten faktisk kunne brukes til. Ettersom man allerede vet at dette er en kritisk sårbarhet som må fikses så snart som mulig, er det ikke det en effektiv bruk av prosjektgruppens ressurser.

I tillegg til de tre webportene var også port 23 (telnet) og port 79 (finger) åpne. Dette kan ikke regnes som webapplikasjoner, og gruppen avgjorde derfor at testing mot disse portene var utenfor prosjektområdet. For sikkerhetsskyld ble det likevel utført litt enkel testing mot port 79, noe som er kort omtalt i både den administrative og den tekniske rapporten.

# Konklusjon

Prosjektgruppen kan konkludere med at penetrasjonstesten ble gjennomført med suksess, og at en betydelig mengde sårbarheter ble avdekket. Webapplikasjonene som det ble testet mot ble funnet på tre forskjellige porter på tjenermaskinen, nemlig 80, 81 og 8080.

Verktøyet som hovedsakelig ble brukt for å utføre automatiske tester var BurpSuite. I tillegg til dette var Nmap, Google Chrome, Firefox og FoxyProxy viktige redskaper.

Metodikken som ble benyttet var først å få en oversikt over webapplikasjonene, og utføre litt generelle og tilfeldige tester. Deretter gikk gruppen mer strukturert til verks, og testet mot så mange sårbarheter som mulig ved hjelp av Burp. Disse sårbarhetene ble videre testet mer manuelt. Så langt det er mulig har gruppen prøvd å fjerne alle spor av at de var inne i systemet. Om noe skulle ligge igjen på systemet i ettertid er det i så fall bare en eller annen tom mappe o.l. som ikke har blitt slettet. I tillegg har det blitt opprettet et par brukere på port 81.

Det største problemet med å gjennomføre testene var at maskinen det ble testet mot var noe ustabil og til tider treg. Dette medførte tap av tid, og at enkelte automatiserte tester ikke kunne utføres ordentlig. Likevel føler prosjektgruppen at de har avdekket en høy andel av sårbarhetene som var på webapplikasjonene. I tillegg til dette var den relativt korte tidsfristen, i en allerede hektisk periode, en begrensning for hvor mange sårbarheter som ble funnet.

Arbeidsfordelingen har fungert slik at alle har samarbeidet tett for å vurdere sårbarhetene ved nettsidene. Alle deltakerne har vært involvert i alle deler av pentesten, og dermed tilegnet seg kunnskaper på et rekke områder. Det har heller ikke vært noe problem å få utført alle deloppgavene i et slikt prosjekt.

Resultatene er presentert i en administrativ rapport (vedlegg 1) og en teknisk rapport (vedlegg 2), og omtales derfor ikke i detalj i denne rapporten. Gruppen tolket prosjektoppgaven som at det i hovedsak kun var webapplikasjonene det skulle testes mot, og testing mot de andre åpne portene har derfor blitt nedprioritert.

Hovedmålet med prosjektet har vært å avdekke sikkerhetshull. Derfor har prosjektgruppen ikke brukt unødvendige ressurser på å undersøke hvor store skader man kan utgjøre ved å utnytte en sårbarhet etter at den er funnet. I stedet har sikkerhetshullet kun blitt dokumentert, siden det uansett må rettes opp i.

# Litteraturliste

[1] Nmap.org, *Nmap*, 2017. Hentet fra: <https://nmap.org/>. Hentet: 21.11.2017.

[2] Portswigger, *Getting started with Burp Suite*, 2017. Hentet fra: <https://support.portswigger.net/customer/portal/articles/1816883-getting-started-with-burp-suite>. Hentet: 21.11.2017.

[3] PentestGeek, *What is Burpsuite*, 2015. Hentet fra: <https://www.pentestgeek.com/what-is-burpsuite>. Hentet: 21.11.2017.

[4] FoxyProxy, *FoxyProxy*, 2017. Hentet fra: <https://getfoxyproxy.org/>. Hentet: 21.11.2017.

## Vedlegg

1 - Administrativ rapport - s. 12

2 - Teknisk rapport - s. 54

Administrativ rapport for  
Penetrasjonstesting av webapplikasjoner for  
South-Dyersville industries

Prosjektgruppe: 3

Deltakere: Fredrik Bjerkø  
Kristen Arntsberg  
Eivind Børstad  
Gard Klemetsen

Utgivelsesdato: 23.11.2017

Begrensning: Åpen

# Innholdsfortegnelse

<b>Kontaktinformasjon</b>	<b>14</b>
<b>Terminologi</b>	<b>15</b>
<b>Sammendrag</b>	<b>16</b>
<b>1 Prosjektbeskrivelse</b>	<b>17</b>
1.1 Bedriftsbeskrivelse	17
1.2 Forutsetninger for penetrasjonstesten	17
<b>2 Funn av sårbarheter</b>	<b>18</b>
2.1 Forklaring av DREAD-modellen	18
2.2 Oversikt over funn	21
2.3 Kritiske funn og sårbarheter (Score 40-50)	23
2.4 Betydelige funn og sårbarheter (Score 25-39)	31
2.5 Moderate funn og sårbarheter (Score 11-24)	44
2.6 Lite kritiske funn (Score 5-10)	49
<b>3 Vurdering og anbefaling</b>	<b>50</b>
<b>Litteraturliste</b>	<b>51</b>

## Kontaktinformasjon

<b>Kunde</b>	
<b>Bedrift</b>	South-Dyersville industries
<b>Kontaktperson</b>	Martin Sundhaug
<b>Telefon</b>	
<b>Epost</b>	martinsundhaug@gmail.com
<b>URL/IP-adresse</b>	10.250.200.3

<b>Pentestgruppe</b>	
<b>Bedrift</b>	Gruppe 3
<b>Kontaktperson</b>	Eivind Børstad
<b>Telefon</b>	47362130
<b>Epost</b>	eivindborstad@gmail.com
<b>Bedriftsadresse</b>	Raveien 215
<b>Postnummer</b>	3184 Borre
<b>URL</b>	Ingen

# Terminologi

Rapporten har blitt skrevet på norsk i den grad det lar seg gjøre. En del engelske fagbegreper er likevel såpass etablerte at de har blitt valgt å ikke oversettes. Flere av disse finnes det heller ingen gode ord for på norsk. En oversikt over disse kommer her:

**Blacklist** - Liste over verdier som er ulovlige. Alle andre verdier er tillatt.

**Clickjacking** - Teknikk for å lure en bruker til å klikke på noe annet.

**Cross-domain referer leakage** - Lekkasje av informasjon på tvers av domene.

**Cross-origin request forgery** - Forfalskning av forespørsel på tvers av domene.

**Cross-origin resource sharing** - Bruk av ressurser fra andre domener.

**Cross-site scripting (XSS)** - Kjøring av vilkårlig javascript-kode i nettleseren.

**Debugger** - Program som brukes til testing av feil.

**Denial of service (DOS)** - Benektelse av tjeneste. Hindre at folk får lastet inn siden.

**Domene** - Enkelt forklart er hver nettside med alle dets undersider et domene.

**Header** - Beskrivende data som er en del av HTML-forespørsler og -responser.

**Hijacking** - Kapring

**HTML** - Standard kode som brukes for å lage nettsider.

**Injection** - Injeksjon av ondsinnet data på en nettside.

**Input** - Data som sendes inn.

**Javascript** - Høynivå programmeringsspråk, brukes i webutvikling.

**Linux** - Et mye brukt operativsystem for webservere.

**OS** - Operativsystem

**Output** - Data som sendes ut.

**Port** - Adressen til prosessen som kjører på en maskin. Brukes ved kommunikasjon.

**Python** - Et mye brukt programmeringsspråk.

**Script** - Kode med kjørbare kommandoer, kalt "skript" på norsk.

**Server** - Tjener som websidene ligger på.

**Token** - Sikkerhetslegitimasjon som brukes for å identifisere en bruker.

**SQL** - Et språk som benyttes for å kommunisere med databaser.

**URL** - Det som skrives inn når man skal inn på en nettside, også kalt en link.

**Whitelist** - Liste over verdier som er lovlige. Alle andre verdier er forbudt.



# Sammendrag

Rapporten beskriver funn av sårbarheter etter at prosjektgruppe 3 utførte en rekke tester mot South-Dyersville industries sine webapplikasjoner fra et administrativt synspunkt. Det betyr at denne rapporten er skrevet for ledelsen og ikke-tekniske personer, og inneholder derfor minimalt med tekniske begreper.

Det ble funnet en rekke kritiske og mindre kritiske sårbarheter. Prosjektgruppen forholdt seg til "OWASPs Topp 10 sårbarheter"[1] når de utførte tester mot applikasjonene. De største funnene fra penetrasjonstesten er:

- Det er mulig å kjøre vilkårlig kode på serveren gjennom enkelte webapplikasjoner
- Flere tilfeller med SQL-injeksjon
- Flere filer med sensitiv informasjon er tilgjengelig for offentligheten
- En passordfil ligger tilgjengelig ute for alle å lese

I kap. 1 gis det en kort beskrivelse av prosjektet, og hvilke forutsetninger som var lagt. Deretter presenteres resultatene på en oversiktlig måte vha. DREAD-modellen i kap. 2. DREAD-modellen blir forklart i første underkapittel i kap. 2.

Det anbefales at South-Dyersville industries burde lage webapplikasjonene sine på nytt med et større fokus på sikkerhet i design- og implementeringsfasen. Dette fordi applikasjonene som kjører på serveren per dags dato er såpass sårbare at det vil være svært tid- og ressurskrevende å sikre alt. Det vil derfor mest sannsynlig kreve færre ressurser å starte på nytt. Alt dette er beskrevet i større detalj i kap. 3. Dersom dette likevel ikke er et alternativ, har prosjektgruppen kommet frem til endringer som bedriften burde foreta seg. Dette står presentert sammen med hver sårbarhet i kap. 2. De viktigste endringene som må gjøres står også listet kort her:

- Oppdatere alle komponenter av systemet til nyeste versjon
- Kryptere nettsideforespørsler med HTTPS
- Legge til et passord på brukeren "Guest" på port 8080, og fjerne filen med passord som ligger ute tilgjengelig
- Sjekke parameter-input for forskjellige typer injeksjoner

Ikke bli bekymret selv om ikke alle begrepene i lista over forstås. Denne rapporten er skrevet for nettopp deg. For en mer teknisk beskrivelse av sårbarhetene som har blitt funnet, kan man i stedet lese den tekniske rapporten.

# 1 Prosjektbeskrivelse

Den 4. november, 2017 fikk prosjektgruppe 3 oppdraget å utføre en omfattende penetrasjonstest mot webapplikasjonene som kjører på 10.250.200.3, for South-Dyersville industries. Det var forutsatt at penetrasjonstesten inkludert rapporter skulle være ferdig innen 30. november.

## 1.1 Bedriftsbeskrivelse

South-Dyersville industries er en nyoppstartet bedrift som benytter seg av en server med IP-adresse 10.250.200.3. Prosjektgruppen har ikke fått noen annen informasjon om bedriften, som preferanser i forhold til presentasjon av resultatene, eller detaljer om hva bedriften holder på med. Resultatene har derfor blitt presentert på en måte som skal virke mest mulig oversiktlig for bedriftsledelsen. Uten å vite hvor mye sensitiv informasjon bedriften holder på, og hvor kritiske operasjoner de utfører, er det vanskelig å vurdere skadepotensialet i forhold til sikkerhetshullene. Det har likevel blitt gjort etter beste evne i DREAD-modellen, som det kan leses om i kap. 2.

## 1.2 Forutsetninger for penetrasjonstesten

Målet med prosjektet var å finne sårbarheter i applikasjonene ved å utføre angrep fra et eksternt synspunkt. Dette innebar at prosjektgruppen kun fikk utdelt bedriftens IP-adresse, og ingenting annet. Ingen av deltakerne hadde vært i kontakt med bedriften tidligere, eller hadde noen form for bruker i systemet deres.

Alle sikkerhetshull som ble funnet har blitt grundig dokumentert. I tillegg har det blitt gitt forslag til hva South-Dyersville industries kan gjøre for å tette disse hullene. Dette presenteres i neste kapittel.

## 2 Funn av sårbarheter

Denne delen av rapporten forklarer de ulike observasjonene som ble funnet under penetrasjonstesten, hvordan disse påvirker bedriften og forslag til hva bedriften kan gjøre. For en mer detaljert beskrivelse av de forskjellige sårbarhetene, henvises det til den tilhørende tekniske rapporten.

Det har blitt funnet et betydelig antall sårbarheter av varierende alvorlighetsgrad som bør behandles. F.o.m kap. 2.2 presenteres disse sårbarhetene på en oversiktlig måte. Først gis en innføring i DREAD-modellen som brukes for å klassifisere og kategorisere sårbarhetene.

### 2.1 Forklaring av DREAD-modellen

En DREAD-modell benyttes for å evaluere observasjonene. Dette er en standard modell for å vurdere sårbarheter, og videre å presentere resultatene på en oversiktlig måte. DREAD er et akronym for de engelske ordene "Damage, Reproducibility, Exploitability, Affected users and Discoverability". Hver sårbarhet blir klassifisert etter disse fem kategoriene, og får en score mellom 1 og 10 i hver kategori, der 10 er mest alvorlig. Tabell 2.1 viser hvordan de forskjellige kategoriene defineres, og hva som skal til for at en sårbarhet får en bestemt score i hver av kategoriene. [2]

<b>Skade-kriterie</b>	<b>Skade-beskrivelse</b>	<b>Kritisk (Score: 9-10)</b>	<b>Høy (Score: 6-8)</b>	<b>Medium (Score: 3-5)</b>	<b>Lav (Score: 1-2)</b>
Skade-potensiale (Damage)	Hvor store skader det kan bli dersom sårbarheten blir utnyttet	En angriper kan få full tilgang til systemet, og kjøre kommandoer om root/administrator	En angriper kan få tilgang til ikke-priviligerte brukere, og lekke svært sensitiv informasjon	En angriper kan lekke sensitiv informasjon, og ta ned systemet	En angriper kan lekke noe informasjon
Mulighet for gjentakelse (Reproducibility)	Hvor vanskelig det er å gjenta angrepet	Angrepet kan reproduseres hver gang og så ofte man vil	Angrepet kan reproduseres på de fleste forsøk	Angrepet kan reproduseres, men det må ventes en viss tid mellom hver gang	Angrepet er svært vanskelig å gjenta, selv med kunnskap om sårbarheten
Utnyttelses-mulighet (Exploitability)	Hvor lett det er å gjennomføre angrepet	Ingen programmerings-kunnskaper kreves, det finnes automatiske verktøy for å utnytte sårbarheten	Lite kunnskaper innen hacking/programmering kreves, og angrepet kan gjennomføres på kort tid	En relativt datakyndig person kan gjennomføre angrepet, og en mindre god programmerer kan gjenta stegene	Angrepet krever en svært datakyndig person og høy kunnskap hver gang angrepet skal gjennomføres
Påvirkede brukere (Affected users)	Hvor stor andel av brukerne eller funksjonalitet som blir påvirket av et suksessfullt angrep	Alle brukere, sentral funksjonalitet	De fleste brukerne, vanlig funksjonalitet	Noen brukere, ikke-vanlig funksjonalitet	En veldig liten andel av brukerne, veldig lite brukt funksjonalitet
Sannsynlighet for oppdagelse (Discoverability)	Hvor vanskelig det er å oppdage sårbarheten	Sårbarheten kan oppdages ved bruk av automatiske verktøy	Det finnes publisert informasjon som beskriver angrepet, og sårbarheten ligger i en av de mest brukte funksjonene	Sårbarheten ligger i en lite brukt del av produktet, og få brukere vil komme over den	Det er lite sannsynlig at sårbarheten vil oppdages

**Tabell 2.1 - Beskrivelse av DREAD-modellen [2]**

Ved å legge sammen scoren i hver kategori får hver sårbarhet en DREAD-score mellom 5 og 50, der 50 er mest kritisk. Basert på denne scoren klassifiseres sårbarhetene i fire forskjellige risikokategorier, som forteller hvor raskt sårbarheten bør fikses.[2] Tabell 2.2 beskriver disse fire risikokategoriene.

Risikokategori	DREAD-score	Risikobeskrivelse
KRITISK	40-50	Et kritisk funn bør bli behandlet umiddelbart. En slik sårbarhet kan få veldig store følger ettersom de er enkle å gjennomføre og ofte kan gi angriperen administratortilgang. Dersom et slikt funn ikke løses raskt kan det raskt få negative konsekvenser for bedriften.
BETYDELIG	25-39	Et betydelig funn bør bli behandlet i løpet av kort tid. Slike sårbarheter kan typisk gi angriperen tilgang til systemet som en vanlig bruker (ikke administratortilgang).
MODERAT	11-24	Et moderat funn bør behandles etter at de kritiske og moderate sårbarhetene er fikset. Lekkasje av sensitiv informasjon er en vanlig konsekvens om slike sikkerhetshull blir utnyttet. Funn av denne typen utgjør en relativt liten trussel mot bedriften.
LITE KRITISK	5-10	Et lite kritisk funn bør behandles etter at alle andre sårbarheter er fjernet. Disse vil av stor sannsynlighet ikke bli utnyttet, og selv ved et suksessfullt angrep vil konsekvensene være minimale. De bør likevel fjernes når man har kapasitet til dette.

**Tabell 2.2 - Risikokategori [2]**

For hvert funn er det i tillegg definert et innsatsnivå som forteller hvor store tiltak som vil kreves for å fjerne sårbarheten. Disse er uavhengig av risikokategori. Tabell 2.3 viser hvordan dette er kategorisert:

Innsatsnivå	Innsatsbeskrivelse
HØY	Tiltak som krever en betydelig mengde ressurser. Kan ta en god stund, og forårsake store endringer i nettverksstruktur. Kjøp av nye sikkerhetsprodukter kan være nødvendig.
MIDDELS	Tiltak som krever en moderat mengde ressurser, og som gjerne kan fikses i løpet av noen få dager.
LAV	Tiltak som krever minimalt med ressurser og som kan fikses i løpet av en dag.

**Tabell 2.3 - Innsatsnivå**

## 2.2 Oversikt over funn

I resten av kap. 2 presenteres de ulike funnene av sårbarheter. Her presenteres en oversikt over alle funn, mens hver av de fire neste underkapitlene går mer detaljert inn på sårbarhetene som tilhører hver av de fire risikokategoriene.

Uttrykkene “port 80”, “port 81” og “port 8080” brukes mye i forklaringene av sårbarhetene. Dette er siden webapplikasjonene som det ble testet mot lå på nettopp disse tre forskjellige “portene”. Det gir dermed en oversiktlig forklaring for hvilke av applikasjonene de forskjellige sårbarhetene er aktuelle for. Hva ordet port betyr er ikke viktig for å forstå innholdet, men for den nysgjerrige leser anbefales [3].

Tabell 2.4 gir en totaloversikt over alle sårbarhetene som har blitt funnet under penetrasjonstesten. Observasjonene er sortert etter synkende DREAD-score, og dermed også etter synkende risikonivå.

Observasjoner	DREAD-score	Innsatsnivå
<b>KRITISK</b>	40-50	
#1 Passord for alle brukere på port 8080 ligger ute offentlig	49	LAV
#2 Lokal filinkludering (LFI)	42	MIDDELS
#3 Kjøring av vilkårlig python-kode	40	MIDDELS
#4 OS-injeksjoner	40	MIDDELS
#5 SQL-injeksjoner	40	MIDDELS

<b>BETYDELIG</b>	25-39	
#6 Tjeneren er sårbar mot DOS-angrep	38	HØY
#7 Ekstern filinkludering (RFI)	36	MIDDELS
#8 Interaksjon med ekstern tjeneste	36	MIDDELS
#9 Bildeopplastning sjekker ikke filtype	36	MIDDELS
#10 Mye sensitiv data ligger tilgjengelig	35	MIDDELS
#11 Webserverversjonen og andre brukte komponenter har kjente sårbarheter	34	HØY
#12 Finger-porten på tjeneren er åpen	34	LAV
#13 Ukryptert kommunikasjon	31	MIDDELS
#14 Cross-Site Scripting (XSS)	31	MIDDELS
#15 Passord sendes ukryptert ved innlogging og registrering	30	LAV
#16 Nettleser kan autofullføre passordfelt	27	LAV
<b>MODERAT</b>	11-24	
#17 Cross-Domain Referer Leakage	24	MIDDELS
#18 Ingen minimumskrav til passordlengde	24	LAV
#19 Cross-Origin Resource Sharing	23	MIDDELS
#20 Cross-Site Request Forgery	22	MIDDELS
#21 Skript fra andre domener inkludert	21	MIDDELS
#22 Potensiell clickjacking	18	LAV
<b>LITE KRITISK</b>	5-10	
Ingen sårbarheter ble funnet innenfor denne kategorien		

**Tabell 2.4 - Oversikt over funn**

## 2.3 Kritiske funn og sårbarheter (Score 40-50)

Et kritisk funn bør bli behandlet umiddelbart. En slik sårbarhet kan få veldig store følger ettersom de er enkle å gjennomføre og ofte kan gi angriperen administratortilgang. Dersom et slikt funn ikke løses raskt kan det raskt få svært negative konsekvenser for bedriften.

### **Funn #1: Passord for alle brukere på port 8080 ligger ute offentlig**

Innsatsnivå: **LAV**

Passord for alle brukerne på port 8080 ligger tilgjengelig på 10.250.200.3:8080/config. I tillegg er passordet på gjestebrukeren blankt, slik at det uansett er svært lett å gjette seg frem til.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	10	10	9	49	KRITISK

### **Bevis**

Brukernavn og passord på hver av brukeren:

```
{
  "user1_password": "",
  "user1_username": "guest",
  "user2_password": "xBQM",
  "user2_username": "admin",
  "user3_password": "eENB1kwC",
  "user3_username": "super"
}
```



## Innloggingssiden:

Please sign in

Login

Superbrukeren (brukeren med flest rettigheter) er ikke listet som et innloggingsvalg, men kan velges ved å modifisere innholdet på nettsiden, noe som er svært lett. Det gjør det heller ikke vanskeligere at innloggingsvalget faktisk ligger i koden, bare deaktivert som en kommentar:

```
<div class="form-group">
  <select class="form-control" name="level">
    <option value="1">Guest</option>
    <option value="2" selected="selected">Administrator</option>
    <!--<option value="3">Superuser</option>-->
  </select>
</div>
```

Slik ser det ut etter at man har logget inn:

leeTech R9001

System

Users

Networking

# System

mounts

```
overlay on / type overlay
(rw,relatime,lowerdir=/var/lib/docker/overlay2/2UG2z5YsJFF4HQJ3K7NLY7OUYES:/var/lib/docker/overlay2/2HFC5JWNYWOXLU7Q4YF5I
prec on /proc type proc (ro,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (ro,nosuid,size=65536k,mode=755)
devpts on /dev/pts type devpts (ro,nosuid,nodev,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,relatime,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (ro,nosuid,nodev,noexec,relatime,xattr,release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/net_cls,cls_nrt_prio type cgroup (ro,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/cpuset type cgroup (ro,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/pids type cgroup (ro,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/perf_event type cgroup (ro,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/devices type cgroup (ro,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/bfbk type cgroup (ro,nosuid,nodev,noexec,relatime,bfbk)
mqueue on /dev/mqueue type mqueue (ro,nosuid,nodev,noexec,relatime)
/dev/ida1 on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/ida1 on /etc/hostname type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/ida1 on /etc/passwd type ext4 (rw,relatime,errors=remount-ro,data=ordered)
shm on /dev/shm type tmpfs (ro,nosuid,nodev,noexec,relatime,size=65536k)
proc on /proc type proc (ro,relatime)
proc on /proc/fs type proc (ro,relatime)
proc on /proc/kcore type proc (ro,relatime)
proc on /proc/kpagecache type proc (ro,relatime)
tmpfs on /proc/kcore type tmpfs (ro,nosuid,size=65536k,mode=755)
tmpfs on /proc/lost_type type tmpfs (ro,nosuid,size=65536k,mode=755)
tmpfs on /proc/mounts type tmpfs (ro,nosuid,size=65536k,mode=755)
tmpfs on /proc/sched_debug type tmpfs (ro,nosuid,size=65536k,mode=755)
tmpfs on /proc/sys/fs/binfmt_misc type tmpfs (ro,nosuid,size=65536k,mode=755)

```

Processes

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 0 0.0 0.4336 716 ? Ss 09:47 0:00 /bin/bash -c FLASK_APP=router.py FLASK_DEBUG=1 WERKZEUG_DEBUG_PIN=off flask run --
host=0.0.0.0 --port=8080
root 11 0.0 0.2 204332 8564 ? Ss 09:47 0:00 /usr/local/bin/python /usr/local/bin/flask run --host=0.0.0.0 --port=8080
root 14 8.5 4.1 1472988 127272 ? Ss 09:47 0:24 /usr/local/bin/python /usr/local/bin/flask run --host=0.0.0.0 --port=8080
root 316 0.0 0.0 4336 768 ? Ss 09:52 0:00 /bin/bash -c ps aux
root 317 0.0 0.0 19186 2328 ? R 09:52 0:00 ps aux
```

Memory

```
total used free shared buffers cached
Mem: 307892 1294804 1783688 35412 90732 477140
+ buffers/cache: 726932 2351560
Swap: 1557568 1171620 379140
```

Processes

```
Architecture x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 2
On-line CPU(s) list: 0,1
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 2
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 37
Model name: Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz
Stepping: 2
CPU MHz: 3324.979
BogoMIPS: 6649.95
Hypervisor vendor: VMware
Virtualization type: full
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 4096K
NUMA node0 CPU(s): 0,1
```

## Diskusjon

Hvem som helst kan logge inn på en overvåkningsside med administratortilgang, noe som er svært alvorlig. Når angriperen er inne har han fått tak i mye sensitiv informasjon om systemet som benyttes. Videre har vedkommende mulighet for å gjøre endringer på systemet, inkludert å endre passord på brukerne. De aller fleste sikkerhetshullene på port 8080 kan ikke gjennomføres uten å være logget inn, så ved å fjerne denne sårbarheten blir det betydelig vanskeligere å utnytte mange av de andre også.

## Anbefalinger

Ikke la denne passordfilen ligge tilgjengelig for alle. Legg umiddelbart inn passord på gjestebBrukeren, eller fjern denne helt. Hvorfor skal en gjest ha tilgang til system- og nettverksegenskaper? Det bør i tillegg være en minimumslengde på passord, noe som blir tatt for seg som en egen sårbarhet, #18.

## Funn #2: Lokal filinkludering (LFI)

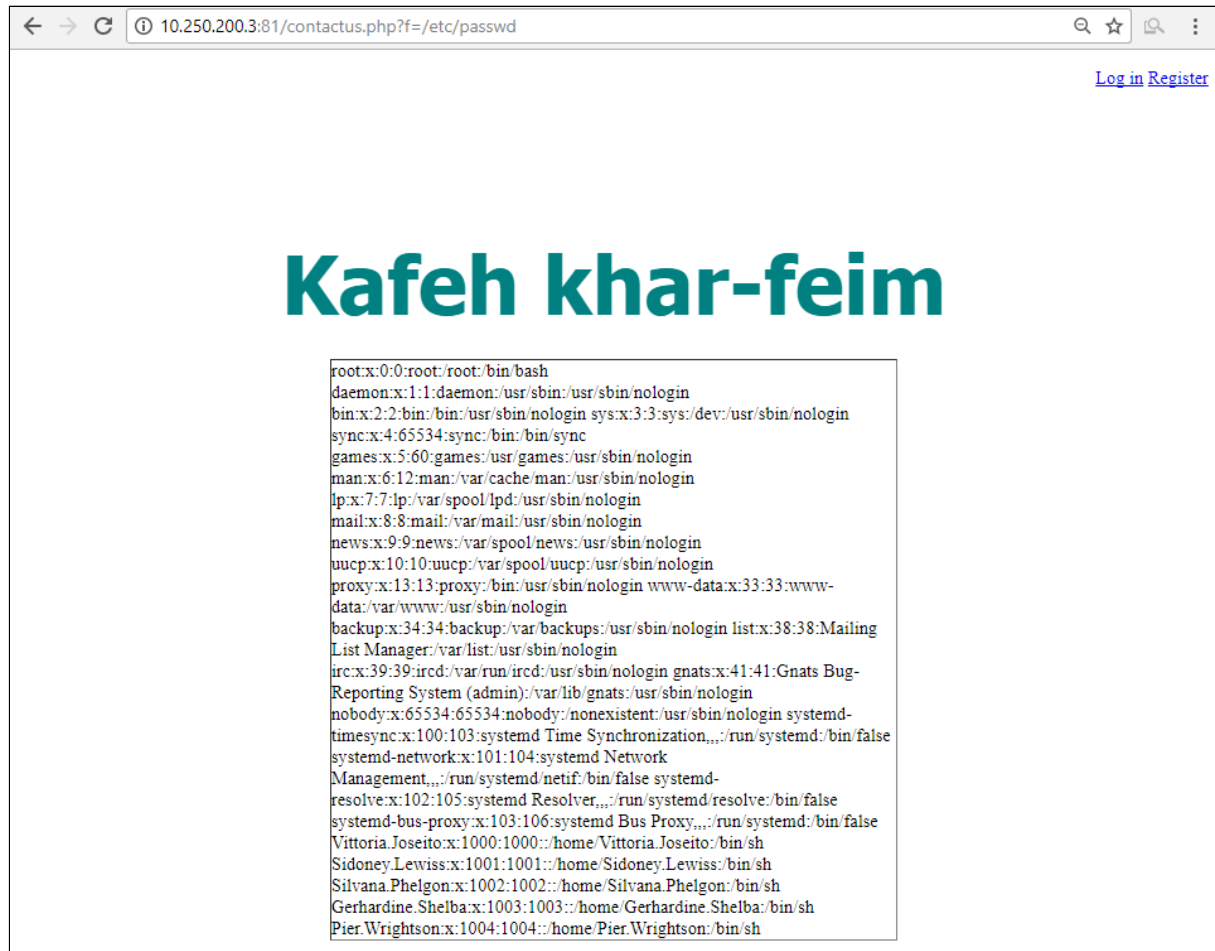
Innsatsnivå: **MIDDELS**

Lokal filinkludering (LFI) på nettsiden 10.250.200.3:81/contactus.php?f="filnavn".

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	8	8	7	42	KRITISK

## Bevis

Her vises innholdet av fila `/etc/passwd` på nettsiden. Dette regnes som en fil med relativt sensitivt innhold siden den inneholder opplysninger om alle brukerne på systemet.



## Diskusjon

En angriper kan få tak i filer som ligger lokalt på serveren ved å skrive dem inn i URL'en der det nå står "filnavn". Dette kan gi informasjon om brukere, serveren og mye annet. Eksempler på innhold prosjektgruppen fant var hashet (dvs. kryptert) passord for alle brukere, samt brukernavn og passord for databasen som benyttes på siden.

## Anbefalinger

Det anbefales å endre på kode, slik at websiden aldri automatisk godtar det brukeren skriver inn. Input fra brukeren skal aldri kunne kjøres direkte, men må valideres. Det anbefales heller ikke å bestemme hvilken fil som skal vises i URL'en. Se [4] for mer informasjon angående lokal filinkludering.

## Funn #3: Kjøring av vilkårlig python-kode

Innsatsnivå: **MIDDELS**

Python-kommandoer kan kjøres på serveren via debuggeren på 10.250.200.3:8080/syscmd og 10.250.200.3:8080/getetc.

Skade-potensiale	Mulighet for gjentakelse	Utnyttelses-mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	6	10	4	40	KRITISK

### Bevis

Her kjøres python-kommandoer i debuggeren. Via python-kommandoer kan man igjen utføre OS-kommandoer. I eksempelet under skrives det ut informasjon om alle filer på serveren som ligger i mappen "etc".

```
builtins.TypeError
TypeError: 'NoneType' object is not iterable

Traceback (most recent call last)
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 1997, in __call__
    return self.wsgi_app(environ, start_response)

[console ready]
>>> import subprocess
>>> output = subprocess.check_output(['ls','-l','/etc'])
>>> print(output)
b'total 536\ndrwxr-xr-x 2 root root 4096 Oct 9 22:30 ImageMagick-6\ndrwxr-xr-x 5 root root 4096 Oct 9 22:30
X11\n-rw-r--r-- 1 root root 2981 Oct 9 00:00 adduser.conf\ndrwxr-xr-x 1 root root 4096 Oct 9 22:30
alternatives\ndrwxr-xr-x 6 root root 4096 Oct 9 00:00 apt\n-rw-r--r-- 1 root root 1863 Nov 5 2016
bash.bashrc\ndrwxr-xr-x 1 root root 4096 Oct 9 22:29 bash_completion.d\n-rw-r--r-- 1 root root 367 Apr 9 2017
bindresvport.blacklist\ndrwxr-xr-x 2 root root 4096 Apr 8 2017 binfmt.d\ndrwxr-xr-x 3 root root 4096 Oct 9
22:28 ca-certificates\n-rw-r--r-- 1 root root 7727 Oct 9 22:28 ca-certificates.conf\ndrwxr-xr-x 2 root root 4096
Oct 9 00:00 cron.daily\ndrwxr-xr-x 3 root root 4096 Apr 8 2017 dbus-1\n-rw-r--r-- 1 root root 2969 Jun 17 11:30
debconf.conf\n-rw-r--r-- 1 root root 4 Jul 13 12:56 debian_version\ndrwxr-xr-x 2 root root 4096 Oct 9 00:00
default\n-rw-r--r-- 1 root root 604 May 15 2012 deluser.conf\ndrwxr-xr-x 1 root root 4096 Oct 10 02:30
dpkg\ndrwxr-xr-x 3 root root 4096 Oct 9 22:30 emacs\n-rw-r--r-- 1 root root 0 Oct 9 00:00 environment\ndrwxr-
xr-x 4 root root 4096 Oct 9 22:30 fonts\n-rw-r--r-- 1 root root 37 Oct 9 00:00 fstab\n-rw-r--r-- 1 root root
2584 Feb 7 2014 gai.conf\n-rw-r--r-- 1 root root 588 Oct 9 22:29 group\n-rw-r--r-- 1 root root 577 Oct 9
00:00 group-1\n-rw-r--r-- 1 root shadow 495 Oct 9 22:29 gshadow\n-rw-r--r-- 1 root root 487 Oct 9 00:00 gshadow-
\ndrwxr-xr-x 3 root root 4096 Oct 9 22:28 gss\n-rw-r--r-- 1 root root 9 Aug 7 2006 host.conf\n-rw-r--r-- 1
root root 13 Nov 15 11:18 hostname\n-rw-r--r-- 1 root root 174 Nov 15 11:18 hosts\ndrwxr-xr-x 2 root root
4096 Oct 9 00:00 init\ndrwxr-xr-x 1 root root 4096 Oct 9 22:30 init.d\n-rw-r--r-- 1 root root 1748 Aug 3 2014
inputrc\ndrwxr-xr-x 3 root root 4096 Nov 23 2012 insserv\n-rw-r--r-- 1 root root 859 Nov 23 2012
insserv.conf\ndrwxr-xr-x 2 root root 4096 Nov 23 2012 insserv.conf.d\ndrwxr-xr-x 2 root root 4096 Oct 9 00:00
iproute2\n-rw-r--r-- 1 root root 26 Jul 13 13:01 issue\n-rw-r--r-- 1 root root 19 Jul 13 13:01
issue.net\ndrwxr-xr-x 3 root root 4096 Oct 9 00:00 kernel\n-rw-r--r-- 1 root root 32419 Oct 10 02:30
...
```

## Diskusjon

Angriperen kan kjøre kode direkte på serveren fra denne debuggeren som administrator. Her er det ingen begrensninger på hva angriperen kan finne på å gjøre. Dette kan være tiltenkt funksjonalitet for en administrator, men slik det er nå kan også gjestebrukeren få tilgang til funksjonaliteten.

## Anbefalinger

Denne sårbarheten er ikke en veldig vanlig sårbarhet. Debuggeren er vanlig å benytte seg av under utvikling, men når applikasjonen er i bruk skal man ikke la denne debuggeren være aktivert [5]. Det anbefales å fjerne denne debuggeren. Om den skal beholdes må den i så fall kun være tilgjengelig for administrator. Sårbarheten henger tett sammen med #1, siden en sikker innlogging på siden betydelig vil redusere mulighetene en angriper har for å kunne benytte seg av dette. Se [5] for mer informasjon angående denne sårbarheten.

## Funn #4: OS-injeksjoner

Innsatsnivå: **MIDDELS**

OS-injeksjoner betyr at man kan kjøre kommandoer rett i operativsystemet. OS-injeksjoner er mulig på

10.250.200.3:8080/syscmd?command=date|"kommando som skal utføres"

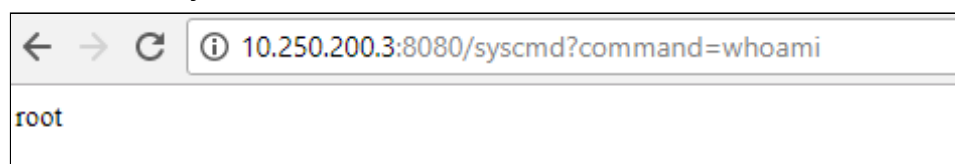
10.250.200.3:81/sys-test.php?test=|"kommando som skal utføres"

På den førstnevnte kjøres i tillegg kommandoene med administratorrettigheter, noe som er spesielt alvorlig.

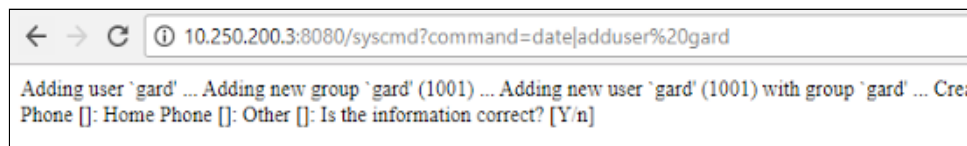
Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	7	10	3	40	KRITISK

## Bevis

I bildet ser man at all kode som kjøres er med administratorrettigheter (eller "root" som det kalles), ved at det har blitt kjørt kommandoen "whoami" som viser hvilken bruker man kjører kommandoer fra.

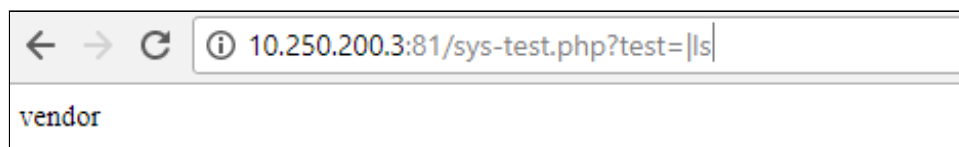


Her opprettes en ny bruker på systemet:



A screenshot of a web browser window. The address bar shows the URL `10.250.200.3:8080/syscmd?command=date|adduser%20gard`. The page content displays the output of a system command: `Adding user 'gard' ... Adding new group 'gard' (1001) ... Adding new user 'gard' (1001) with group 'gard' ... Creating user 'gard' ... Home Phone []: Other []: Is the information correct? [Y/n]`.

På port 81 var det litt mer begrenset hvilke kommandoer som gikk gjennom, men dette er fortsatt en alvorlig sårbarhet.



A screenshot of a web browser window. The address bar shows the URL `10.250.200.3:81/sys-test.php?test=|ls|`. The page content displays the output of a directory listing: `vendor`.

## Diskusjon

Den avdekkede sårbarheten er av høy alvorlighetsgrad. Angriperen har full kontroll over systemet. For sårbarheten på port 81 er heller ingen innlogging nødvendig.

## Anbefalinger

Unngå at brukerens input kan kjøres. Valider inputen ved å se etter tegn som ikke bør være der. Når det gjelder en slik kommando bør den heller ikke sendes gjennom URL'en. Se [6] for mer informasjon angående dette.

## Funn #5: SQL-injeksjoner

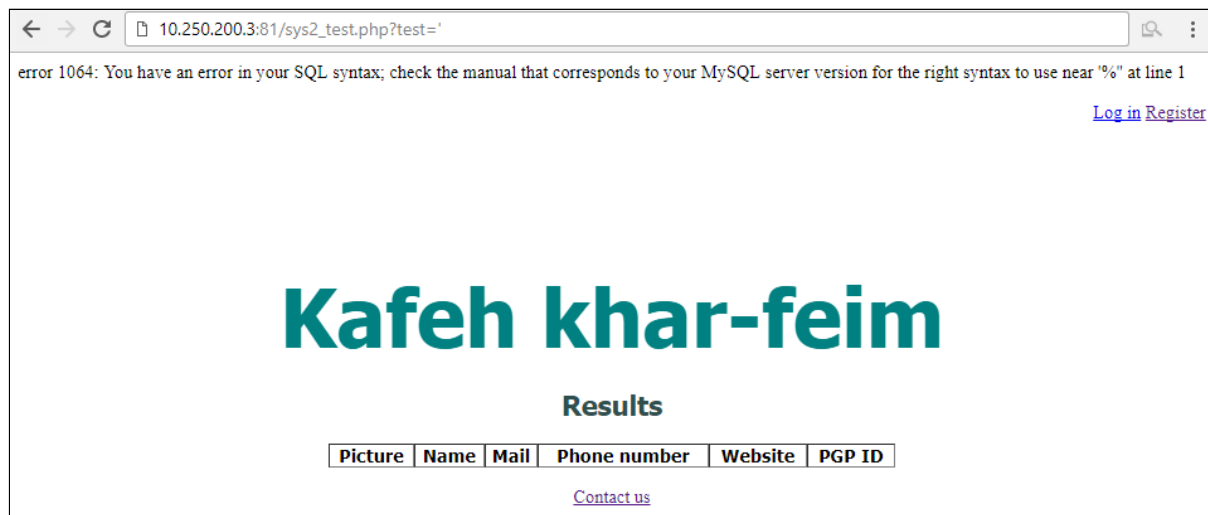
Innsatsnivå: **MIDDELS**

SQL-injeksjoner er mulig på  
`10.250.200.3:8080/syscmd?command=`  
`10.250.200.3:81/sys2_test.php?test=`  
`10.250.200.3:81/register?username=`  
`10.250.200.3:81/search?query=`  
`10.250.200.3:81/user/"NavnPåBruker"`

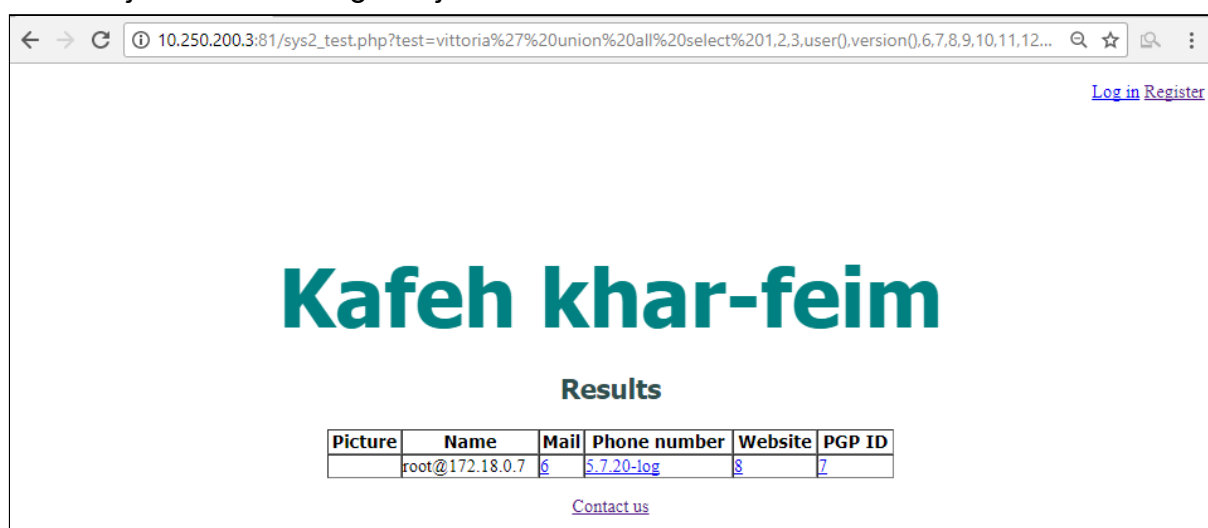
Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	7	10	3	40	KRITISK

## Bevis

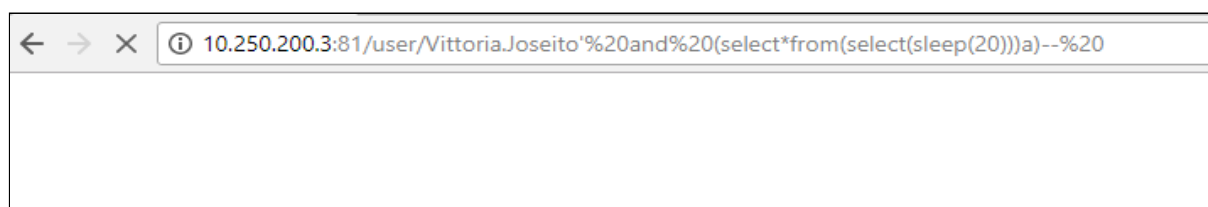
Her vises en SQL-feilmelding etter at det har blitt forsøkt å plassere en apostroff i URL'en på 10.250.200.3:81/sys2\_test.php?test='



Med dette kan man hente ut informasjon fra databasen. I bildet under vises informasjon om bruker og versjon:



Standard SQL-injeksjon fungerte ikke på de øvrige sidene, men noen form for SQL-injeksjon var mulig også der. På siden 10.250.200.3:81/user/Vittoria.Joseito fikk URL'en under siden til å stå å laste i 20 sekunder før noe skjedde, akkurat som kommandoen ber om.



## Diskusjon

SQL-injeksjoner går ut på å endre kommunikasjonen som sendes mellom serveren og databasen der dataene ligger lagret. Med SQL-injeksjon kan angriperen manipulere databasen som benyttes, som eksempelvis å hente sensitiv informasjon om brukere, slette, legge til osv. [7].

## Anbefalinger

Anbefales at all input blir validert og sjekket før noe sendes til databasen, se [7] for mer informasjon angående dette.

## 2.4 Betydelige funn og sårbarheter (Score 25-39)

Et betydelig funn bør bli behandlet i løpet av kort tid. Slike sårbarheter kan typisk gi angriperen tilgang til systemet som en vanlig bruker (ikke administratortilgang).

### Funn #6: Tjeneren er sårbar mot DOS-angrep

Innsatsnivå: **HØY**

DOS (Denial of service)-angrep går ut på å overbelaste serveren slik at ingen kan gå inn på nettsiden [8]. Serveren går raskt ned ved overbelastning, og det var når kun prosjektgruppen testet mot den. Ved mange besøkende kommer serveren til å være svært ustabil. Spesielt nettsidene på port 8080 gikk fort ned.

Skade-potensiale	Mulighet for gjentakelse	Utnyttelses-mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
4	9	9	10	6	38	BETYDELIG

## Bevis

Serveren gikk ned flere ganger under penetrasjonstesten, og brukte svært ofte lang tid på å laste ned sider.

## Diskusjon

Konsekvensene av et DOS-angrep er svært avhengig av hvor mange besøkende nettsiden er tiltenkt. Dersom nettsiden ikke har den store målgruppen, og ikke skal brukes til kritiske operasjoner som f.eks. banktransaksjoner, er det ikke nødvendigvis katastrofalt om serveren går ned en sjelden gang.



## Anbefalinger

Erstatte server med en som har mer ytelse. Implementere funksjonalitet som kan forhindre slike angrep [8]. Se [8] for mer informasjon angående et slikt angrep.

## Funn #7: Ekstern filinkludering (RFI)

Innsatsnivå: **MIDDELS**

Ekstern filinkludering (RFI) er mulig på 10.250.200.3:81/contactus.php?f="link til fil"

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	10	4	6	7	36	BETYDELIG

## Bevis

På bildet under vises innholdet av "http://google.com" som ren tekst på nettsiden.



## Diskusjon

Angriperen kan lure applikasjonen til å inkludere ekstern data på siden. I verste fall kan det føre til at maskinen kjører vilkårlig kode etter angriperens ønske [9]. Denne sårbarheten er svært tett koblet sammen med sårbarhet #2, lokal filinkludering (LFI).

## Anbefalinger

Anbefales å utføre det samme som ved lokal filinkludering, se sårbarhet #2. Må ha tester på input som sjekker at teksten som skal leses inn ikke er farlig. Det er også unødvendig å spesifisere filnavn på denne måten i URL'en. Se [4] og [10] for mer informasjon om dette.

## Funn #8: Interaksjon med ekstern tjeneste

Innsatsnivå: **MIDDELS**

Ekstern tjenesteinteraksjon med DNS og HTTP på

10.250.200.3:81/sys-test.php?test=google.com

10.250.200.3:81/contactus.php?f=google.com

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
10	10	3	10	3	36	BETYDELIG

## Bevis

Serveren kontakter en DNS-server for å få kontakt med google.com


--

## Diskusjon

Ekstern tjenesteinteraksjon forekommer når det er mulig å få en applikasjon til å arbeide med en ekstern tjeneste, som web- eller mailserver. Dette er ikke en sårbarhet i seg selv, men i noen tilfeller kan det være katastrofalt hvis dette er til stede uten at det dette var meningen. I slike situasjoner kan en ondsinnet angriper bruke applikasjonsserveren som en angreps-proxy mot de interne eller eksterne tjenestene applikasjonen samhandler med [11].

DNS og HTTP er to slike eksterne tjenester. Betydningen av dem er ikke viktig i denne sammenheng.

## Anbefalinger

Dersom en ekstern tjenesteinteraksjon ikke er nødvendig for funksjonalitet på siden, bør det settes opp en "whitelist" på alle tjenester serveren skal få lov til å bruke, mens den skal blokkere alle andre tjenester.

## Funn #9: Bildeopplastning sjekker ikke filtype

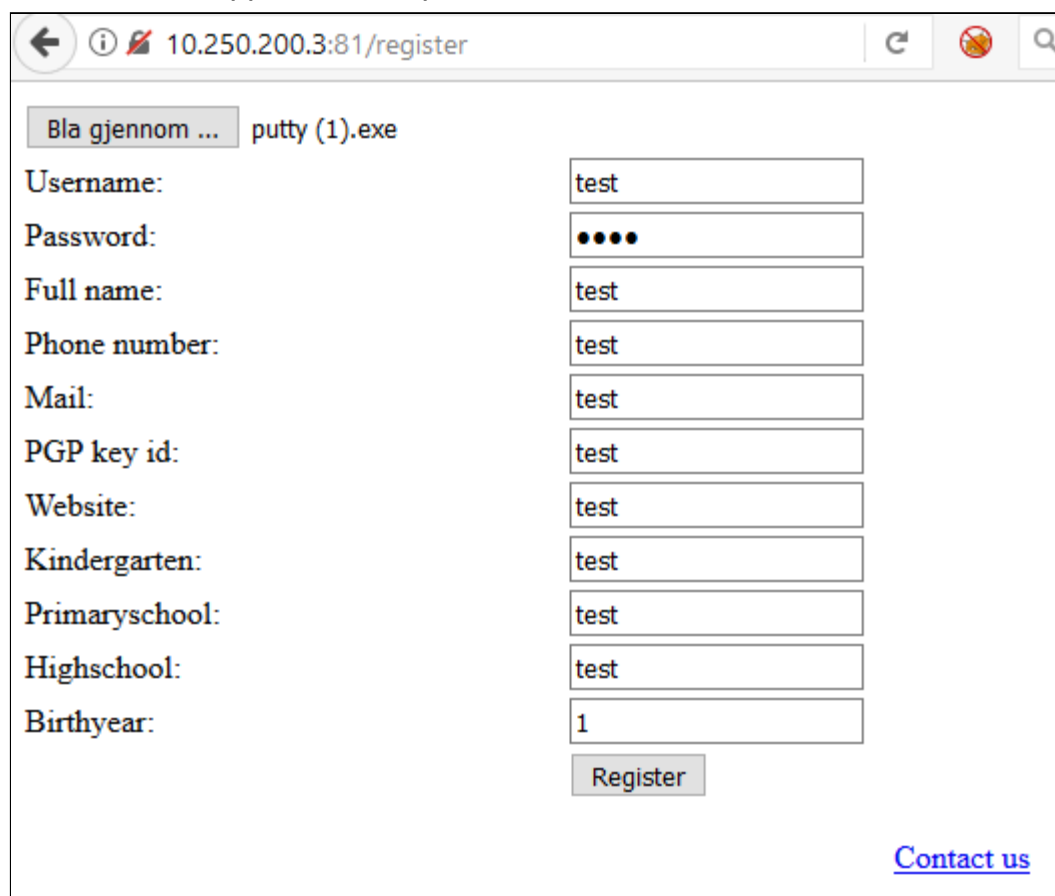
Innsatsnivå: **MIDDELS**

Når en ny bruker registrer seg på 10.250.200.3:81/register, er det mulig å laste opp et bilde. Det sjekkes imidlertid ikke hva slags fil som faktisk lastes opp. Dette betyr at en angriper potensielt kan laste opp en skadelig fil til systemet. Det samme er mulig når man oppdaterer en brukers info på 10.250.200.3:81/user/"Navn"/edit. Det er videre mulig for hvem som helst å endre informasjon om hvilken bruker som helst, noe som heller ikke er bra.

Skade-potensiale	Mulighet for gjentakelse	Utnyttelses-mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
8	10	3	7	8	36	BETYDELIG

### Bevis

Her lastes det opp en .exe-fil på serveren.



10.250.200.3:81/register

Bla gjennom ... putty (1).exe

Username: test

Password: ●●●●

Full name: test

Phone number: test

Mail: test

PGP key id: test

Website: test

Kindergarten: test

Primaryschool: test

Highschool: test

Birthyear: 1

Register

[Contact us](#)

## Diskusjon

En angriper kan potensielt laste opp en ondsinnet fil som senere kan kjøres fra serveren. Dette kan få svært alvorlige konsekvenser.

## Anbefalinger

Alle filer som lastes opp til serveren må skannes før de lagres. Her kan bedriften for eksempel bruke et antivirus. Applikasjonen bør kun akseptere opplastning av relevante filer som er forventet, i dette tilfellet filtypene .jpg, .png, eller en annen filtype som representerer bildefiler.

## **Funn #10: Mye sensitiv data ligger tilgjengelig**

Innsatsnivå: **MIDDELS**

Det ligger svært mye sensitiv informasjon tilgjengelig rundt om på nettsidene. Dette inkluderer opplysninger om server, brukere og data som ligger på den. Følgende filer inneholder sensitiv informasjon:

- 10.250.200.3:8080/config/
- 10.250.200.3:8080/admin-networking.html
- 10.250.200.3:8080/admin-system.html
- 10.250.200.3:8080/admin-users.html
- 10.250.200.3:8080/static/admin.js
- 10.250.200.3:8080/static/common.js
- 10.250.200.3:8080/static/js.cookie.js
- 10.250.200.3:81/
- 10.250.200.3:81/server-status
- 10.250.200.3:81/phpinfo.php
- 10.250.200.3:80/wp-json/
- 10.250.200.3:80/wp-json/oembed/1.0/
- 10.250.200.3:80/wp-content/themes/twentytwenty/js/html5.js
- 10.250.200.3:80/wp-includes/js/jquery/jquery.js
- 10.250.200.3:80/wp-includes/js/wp-embed.min.js
- 10.250.200.3:80/wp-includes/wlwmanifest.xml

I tillegg bør feilmeldinger standardiseres i større grad. En ikke-standardisert feilmelding vil gi angriperen informasjon om hva som gikk galt. Noe som kan brukes i nye angrepsforsøk.

- 10.250.200.3:81/register
- 10.250.200.3:81/login
- 10.250.200.3:81/contactus.php
- 10.250.200.3:81/sys-test.php
- 10.250.200.3:81/sys2\_test.php

10.250.200.3:8080/getetc  
10.250.200.3:8080/syscmd

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
6	10	4	10	5	35	BETYDELIG

**Bevis**  
Under vises et par av sidene nevnt i listen over.  
10.250.200.3:8080/admin-system.html

← → 10.250.200.3:8080/admin-system.html

leeTech R9001

System

Users

Networking

System

mounts

overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/21UG22VSVYJFA4HQUJ3K7NYL7OLYES:/var/lib/docker/overlay2/21F5CJ5WNYWXL17Q4YE5,proc on /proc type proc (rw,nosuid,nodev,noexec,relatime),tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755),devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666),sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime),tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,relatime,mode=755),cgroup on /sys/fs/cgroup/systemd type cgroup (ro,nosuid,nodev,noexec,relatime,xattr=release\_agent='lib/systemd/systemd-cgroups-agent,name=systemd'),cgroup on /sys/fs/cgroup/net\_cls,net\_prio type cgroup (ro,nosuid,nodev,noexec,relatime,net\_cls=net\_prio),cgroup on /sys/fs/cgroup/cpuset type cgroup (ro,nosuid,nodev,noexec,relatime,cpuset),cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpu,cpuacct),cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,noexec,relatime,hugetlb),cgroup on /sys/fs/cgroup/pids type cgroup (ro,nosuid,nodev,noexec,relatime,pids),cgroup on /sys/fs/cgroup/perf\_event type cgroup (ro,nosuid,nodev,noexec,relatime,perf\_event),cgroup on /sys/fs/cgroup/devices type cgroup (ro,nosuid,nodev,noexec,relatime,devices),cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory),cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,noexec,relatime,freezer),cgroup on /sys/fs/cgroup/bkno type cgroup (ro,nosuid,nodev,noexec,relatime,bkno),inotify on /dev/inotify type inotify (rw,nosuid,nodev,noexec,relatime),/dev/ida1 on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro,data=ordered),/dev/ida1 on /etc/hostname type ext4 (rw,relatime,errors=remount-ro,data=ordered),/dev/ida1 on /etc/passwd type ext4 (rw,relatime,errors=remount-ro,data=ordered),shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k),proc on /proc/bus type proc (ro,relatime),proc on /proc/fs type proc (ro,relatime),proc on /proc/irq type proc (ro,relatime),proc on /proc/sys type proc (ro,relatime),proc on /proc/sysrq-trigger type proc (ro,relatime),tmpfs on /proc/core type tmpfs (rw,nosuid,size=65536k,mode=755),tmpfs on /proc/timer\_list type tmpfs (rw,nosuid,size=65536k,mode=755),tmpfs on /proc/timer\_status type tmpfs (rw,nosuid,size=65536k,mode=755),tmpfs on /proc/sched\_debug type tmpfs (rw,nosuid,size=65536k,mode=755),tmpfs on /sys/firmware type tmpfs (ro,relatime)

Processes

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND  
root 1 0 0 0 4336 716 ? Ss 05:47 0:00 /bin/sh -c FLASK\_APP=order.py FLASK\_DEBUG=1 WERKZEUG\_DEBUG\_PIN=off flask run --host=0.0.0.0 --port=8080  
root 11 0 0 2 204332 8664 ? S 09:47 0:00 /usr/local/bin/python /usr/local/bin/flask run --host=0.0.0.0 --port=8080  
root 14 6.4 1 1472888 127272 ? Sl 09:47 0:24 /usr/local/bin/python /usr/local/bin/flask run --host=0.0.0.0 --port=8080  
root 316 0 0 0 4336 768 ? S 09:52 0:00 /bin/sh -c ps aux  
root 317 0 0 0 19188 2328 ? R 09:52 0:00 ps aux

Memory

total used free shared buffers cached  
Mem: 3076492 1294884 1781608 35412 90732 477140  
-/+ buffers/cache: 726932 2351560  
Swap: 1567568 1177028 379740

Processors

Architecture: x86\_64  
CPU op-mode(s): 32-bit, 64-bit  
Byte Order: Little Endian  
CPU(s): 2  
On-line CPU(s) list: 0,1  
Thread(s) per core: 1  
Core(s) per socket: 1  
Socket(s): 2  
NUMA node(s): 1  
Vendor ID: GenuineIntel  
CPU family: 6  
Model: 37  
Model name: Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz  
Stepping: 2  
CPU MHz: 3324.979  
BogoMIPS: 6649.95  
Hypervisor vendor: VMware  
Virtualization type: full  
L1d cache: 32K  
L1i cache: 32K  
L2 cache: 256K  
L3 cache: 4096K  
NUMA node0 CPU(s): 0,1

10.250.200.3:81/server-status

10.250.200.3:81/server-status

← ⓘ 10.250.200.3:81/server-status

📄 ↺ 🚫 🔍 Søk

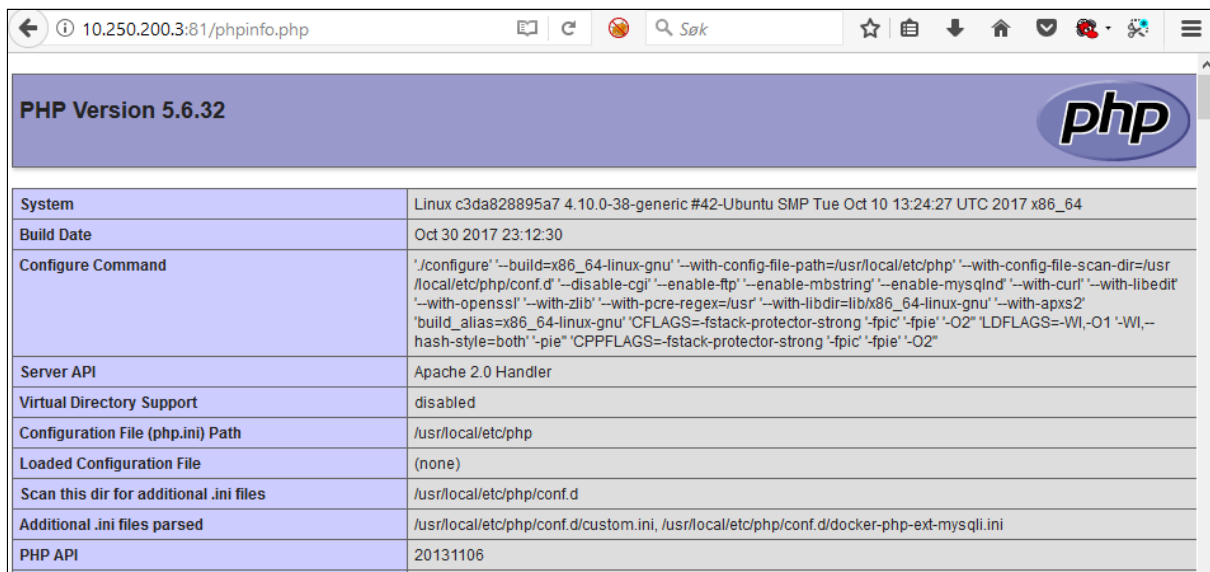
★ 📁 ⬇️ 🏠 🔄 🚫 🛠️

# Apache Server Status for 10.250.200.3 (via 172.18.0.7)

Server Version: Apache/2.4.10 (Debian) PHP/5.6.32  
Server MPM: prefork  
Server Built: Sep 20 2017 04:37:43

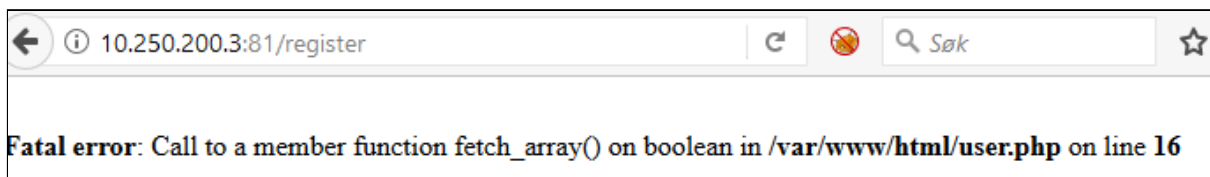
Current Time: Sunday, 19-Nov-2017 16:00:29 UTC  
Restart Time: Sunday, 19-Nov-2017 03:31:34 UTC  
Parent Server Config. Generation: 1  
Parent Server MPM Generation: 0  
Server uptime: 12 hours 28 minutes 54 seconds  
Server load: 0.00 0.00 0.00  
Total accesses: 149 - Total Traffic: 13.7 MB

10.250.200.3:81/phpinfo.php



<b>PHP Version 5.6.32</b>	
System	Linux c3da828895a7 4.10.0-38-generic #42-Ubuntu SMP Tue Oct 10 13:24:27 UTC 2017 x86_64
Build Date	Oct 30 2017 23:12:30
Configure Command	./configure '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--disable-cgi' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-pcre-regex=/usr' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' 'build_alias=x86_64-linux-gnu' 'CFLAGS=-fstack-protector-strong -fPIC -fPIE -O2' 'LDFLAGS=-Wl,-O1,-Wl,-hash-style=both -pie' 'CPPFLAGS=-fstack-protector-strong -fPIC -fPIE -O2'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/custom.ini, /usr/local/etc/php/conf.d/docker-php-ext-mysqli.ini
PHP API	20131106

Eksempelet under viser 10.250.200.3:81/register som ikke har standardiserte feilmeldinger:



## Diskusjon

Det er ikke skadelig at informasjon som dette ligger ute i utgangspunktet, men en angriper kan bruke denne informasjonen til å finne andre sikkerhetshull. I tillegg kan det være at enkelte av dataene som ligger ute bryter med personvern o.l.

## Anbefalinger

Det anbefales at kun administrator kan lese informasjon som dette. Ikke la sensitiv informasjon ligge ute på sider som bare kan skrives rett inn i URL'en.

## Funn #11: Webserverversjonen og andre brukte komponenter har kjente sårbarheter

Innsatsnivå: **HØY**

Serveren benytter Apache 2.4.10 som webserver på port 80 og 81. Dette er en gammel versjon som har kjente sårbarheter [12]. Også Wordpress v4.6, som brukes for å generere sidene på port 80, er gammel og har kjente sårbarheter [13]. I tillegg ser det ut til at serveren bruker operativsystemet Ubuntu v4.10 [14]. Denne er også utdatert og har kjente sårbarheter. En gammel versjon av PHP bruker på port 81. Det

ser ikke ut til å være noen kjente sårbarheter mot den, men en så gammel versjon bør uansett oppdateres.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	5	2	10	8	34	BETYDELIG

## Bevis

Apache-versjon ble funnet gjennom en portskanning:

```
PORT      STATE SERVICE VERSION
23/tcp    open  telnet?
79/tcp    open  finger?
80/tcp    open  http   Apache httpd 2.4.10 ((Debian))
81/tcp    open  http   Apache httpd 2.4.10 ((Debian))
8080/tcp  open  http   Werkzeug httpd 0.12.2 (Python 3.6.3)
```

Wordpress-versjon:

```
<?xml version="1.0" encoding="UTF-8"?><rss version="2.0"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:wfw="http://wellformedweb.org/CommentAPI/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
>

<channel>
  <title>South-Dyersville industries</title>
  <atom:link href="http://10.250.200.3/feed/" rel="self" type="application/rss+xml" />
  <link>http://10.250.200.3</link>
  <description>Just another WordPress site</description>
  <lastBuildDate>Mon, 23 Oct 2017 15:22:16 +0000</lastBuildDate>
  <language>en-US</language>
  <sy:updatePeriod>hourly</sy:updatePeriod>
  <sy:updateFrequency>1</sy:updateFrequency>
  <generator>https://wordpress.org/?v=4.6</generator>
</channel>
</rss>
```

Operativsystemversjon ble funnet ved å sjekke fila /proc/version gjennom sårbarhet #2, lokal filinkludering (LFI):

```
Linux version 4.10.0-38-generic (buildd@lgw01-amd64-032) (gcc version
6.3.0 20170406 (Ubuntu 6.3.0-12ubuntu2) ) #42-Ubuntu SMP Tue Oct 10
13:24:27 UTC 2017
```

[Contact us](#)

PHP-versjon ble funnet på siden 10.250.200.3:81/phpinfo.php:



PHP Version 5.6.32	
System	Linux c3
Build Date	Oct 30 2

## Diskusjon

Det har blitt funnet flere sårbarheter i Apache 2.4.10, se [12] for mer informasjon om mulige angrep. Dette er en gammel versjon av Apache og burde helst ikke brukes. I tillegg bruker serveren WordPress-, Ubuntu- og PHP-versjoner som er utdaterte [13][14].

Gammel programvare har ofte kjente sårbarheter i seg, og det anbefales at man alltid oppdaterer til nyeste versjon. Mange komponenter kan settes til å oppdateres automatisk.

## Anbefalinger

Det anbefales at alle tjenester som kjører på serveren er av de nyeste versjonene. Man bør enten få slik programvare til å oppdateres automatisk, eller holde seg oppdatert slik at man får med seg når nye versjoner kommer ut, og når det blir oppdaget sårbarheter mot spesifikke versjoner.

## Funn #12: Finger-porten på tjeneren er åpen

Innsatsnivå: **LAV**

Finger-tjenesten er åpen på 10.250.200.3:79.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
3	10	8	10	3	34	BETYDELIG



## Bevis

```
root@panda:~# finger @10.250.200.3
Vittoria.Joseito:OK
Sidoney.Lewiss:OK
Silvana.Phelgon:OK
Gerhardine.Shelba:OK
Pier.Wrightson:OK
```

## Diskusjon

Finger-protokollen finnes i noen standarddistribusjoner av Linux. Protokollen brukes for å utveksle informasjon om brukerne på serveren [15]. Man kan anskaffe brukernavn, sist de var innlogget, m.m med finger-kommandoen. En angriper kan derfor benytte seg av denne til å skaffe informasjon som ikke skal være tilgjengelig[15].

## Anbefalinger

Det anbefales å deaktivere finger-protokollen eller ihvertfall erstatte den med en annen protokoll dersom dette er nødvendig for bedriften[16].

## Funn #13: Ukryptert kommunikasjon

Innsatsnivå: **MIDDELS**

Kommunikasjon mellom klient og server foregår ukryptert.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	8	4	2	8	31	<b>BETYDELIG</b>

## Bevis

Webapplikasjonene benytter seg av HTTP. Dette er en applikasjonsprotokoll som benyttes ved overføring av informasjon mellom server og klient [17]. Eksempel som viser at 10.250.200.3 bruker HTTP: `http://10.250.200.3:81/server-status`.

## Diskusjon

HTTP sender data ukryptert mellom klient og server. En angriper som får tak i disse pakkene vil kunne lese dem, siden pakkene er i klartekst [17]. Dette er svært usikkert og skal helst ikke brukes i noen sammenhenger - spesielt ikke på sider der en bruker er logget inn.

## Anbefalinger

Det anbefales å bytte til HTTPS, som er en sikker versjon av HTTP. HTTPS krypterer informasjonen før den sendes mellom klient og server [17]. På denne måten kan ikke angriperen lese noe av informasjonen før han eventuelt har gjort en dekryptering. En slik dekryptering er svært vanskelig å gjennomføre.

## Funn #14: Cross-Site Scripting (XSS)

Innsatsnivå: **MIDDELS**

Reflektert Cross-Site Scripting (XSS) på

10.250.200.3:81/register?username=

10.250.200.3:81/user/"Navn"/key.asc

10.250.200.3:81/user/"Navn"/me.jpg

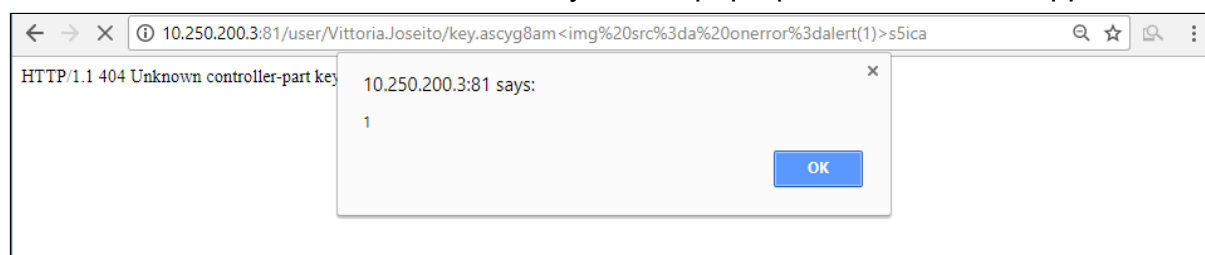
I tillegg er det spesifisert at nettlesere som har innebygde XSS-filtre ikke skal bruke disse.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	3	7	4	8	31	BETYDELIG

## Bevis

Her for javascript-kode plassert rett i URL'en til

10.250.200.3:81/user/Vittoria.Joseito/key.asc en pop-up boks til å dukke opp:



## Diskusjon

En angriper kan bruke (XSS) til å få en annen bruker til å kjøre et farlig skript. Brukeren mottar en link fra angriper, og brukerens nettleser vet ikke at skriptet ikke skal kjøres, og vil kjøre skriptet [18]. En angriper kan derfor anskaffe data om brukeren, som f.eks. kan brukes til å utgi seg som brukeren på nettsiden.

Reflektert XSS betyr at programkoden kjøres umiddelbart etter at den har blitt sendt inn til nettsiden. En enda mer alvorlig versjon er såkalt lagret XSS, der den

ondsinnede koden blir plassert i en database på serveren, og vil bli kjørt hver gang den hentes ut av databasen [18][19].

### Anbefalinger

Sjekk at brukerinntut ikke inneholder ting det ikke skal gjøre. Det vanligste å gjøre er å filtrere all input fra brukerne. La bl.a. kun noen typer tegn kunne bli skrevet inn. Ved vanlig tekst holder det som regel at bruker kan skrive bokstaver og tall, og trenger ikke spesialtegn [19]. Det anbefales også å ta en titt på [19] for å unngå denne sårbarheten. XSS er en veldig vanlig sårbarhet, og klarer man å eliminere denne sårbarheten er man godt rustet.

Skru på nettleseres innebygde XSS-filtre.

### **Funn #15: Passord sendes ukryptert ved innlogging og registrering**

Innsatsnivå: **LAV**

Passord krypteres ikke før de sendes til serveren. Dette gjelder ved innlogging både på port 80, 81 og 8080, samt ved registrering på port 81.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	8	4	2	7	30	BETYDELIG

### Bevis

Her vises en bit av HTML-koden ved innlogging på side 8080. Passord sendes direkte avgårde uten noen form for kryptering først.

```
<form action="/login" role="form">
  <div class="form-group">
    <select name="level" class="form-control">
      <option value="1">Guest</option>
      <option value="2" selected="selected">Administrator</option>
      <!--<option value="3">Superuser</option>-->
    </select>
  </div>
  <div class="form-group">
    <input type="password" name="password" class="form-control" placeholder="Password" />
  </div>
  <input type="submit" value="Login" class="btn btn-success btn-default" />
</form>
```

### Diskusjon

Se også funn #13. En angriper kan lytte på avsendingene og lese passordet direkte fra pakkene [20].

## Anbefalinger

Bedriften må benytte seg av en sikker overføringsprotokoll som f.eks HTTPS. Se [17] for mer informasjon. I tillegg bør passordet krypteres for seg. Et passord i klartekst skal aldri sendes av gårde, og heller aldri ligge lagret i databasen på serveren - kun en kryptert versjon.

## Funn #16: Nettleser kan autofullføre passordfelt

Innsatsnivå: **LAV**

På alle innloggingssider er det heller ikke spesifisert at nettlesere ikke skal huske på passord. Dette betyr at hvis man logger inn fra en offentlig maskin, kan noen andre senere logge inn ved at nettleseren husker på brukernavn og passord for forrige bruker.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	3	5	2	8	27	BETYDELIG

## Bevis

Her vises et utdrag av HTML-koden for innlogging på port 80. For at et passord ikke skal automatisk utfylles, må man spesifisere dette med `autocomplete="off"`, noe som ikke er gjort her.

```
<form name="loginform" id="loginform" action="http://10.250.200.3/wp-login.php" method="post">
  <p>
    <label for="user_login">Username or Email<br />
    <input type="text" name="log" id="user_login" class="input" value="" size="20" /></label>
  </p>
  <p>
    <label for="user_pass">Password<br />
    <input type="password" name="pwd" id="user_pass" class="input" value="" size="20" /></label>
  </p>
  <p class="forgetmenot"><label for="rememberme"><input name="rememberme" type="checkbox"
id="rememberme" value="forever" /> Remember Me</label></p>
  <p class="submit">
    <input type="submit" name="wp-submit" id="wp-submit" class="button button-primary button-large"
value="Log In" />
    <input type="hidden" name="redirect_to" value="http://10.250.200.3/wp-admin/" />
    <input type="hidden" name="testcookie" value="1" />
  </p>
</form>
```

## Diskusjon

Ved innlogging fra en offentlig maskin, vil passordet i noen tilfeller ligge lagret. Dette gjør at en angriper kan logge seg inn når personen har forlatt maskinen.

## Anbefalinger

Det anbefales å sette `autocomplete="off"` i HTML-koden til nettsiden.

## 2.5 Moderate funn og sårbarheter (Score 11-24)

Et moderat funn bør behandles etter at de kritiske og moderate sårbarhetene er fikset. Lekkasje av sensitiv informasjon er en vanlig konsekvens om slike sikkerhetshull blir utnyttet. Funn av denne typen utgjør en relativt liten trussel mot bedriften.

### **Funn #17: Cross-Domain Referer Leakage**

Innsatsnivå: **MIDDELS**

Forekommer når en nettside henter inn innhold fra andre nettsider [21], og har blitt funnet på følgende sider:

10.250.200.3/

10.250.200.3/wp-login.php

10.250.200.3:81/sys2\_test.php

10.250.200.3:8080/

10.250.200.3:8080/getetc

10.250.200.3:8080/login

10.250.200.3:8080/syscmd

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
4	10	3	2	5	24	MODERAT

### **Bevis**

Her ser man at nettsiden henter inn ressurser fra eksterne domener. Dette er fra 10.250.200.3:80/

```
<link rel='dns-prefetch' href='//fonts.googleapis.com'>
<link rel='dns-prefetch' href='//s.w.org'>
<link rel="alternate" type="application/rss+xml" title="South-Dyersville industries &raquo; Feed" href="http://10.250.200.3/feed/" />
```

### **Diskusjon**

Når nettsiden forespør andre nettsider om slikt innhold, sendes det informasjon fra nettsiden som forespør til den andre. Hvis det for eksempel ligger sensitiv informasjon i URL'en til nåværende side, vil dette bli overført til den andre siden. På denne måten kan sensitiv informasjon lekke ut. [21]

## Anbefalinger

Applikasjonene burde ikke sende sensitiv informasjon i URL'en. Vurder også om sidene man henter eksterne ressurser fra, kan stoles på.

### **Funn #18: Ingen minimumskrav til passordlengde**

Innsatsnivå: **LAV**

Passord kan være så korte man vil på sidene på port 81 og 8080.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
9	5	3	3	4	24	MODERAT

## Bevis

Et bevis på dette er gjestebrukeren på port 8080 som har blankt passord. I tillegg har det blitt opprettet brukere på port 81 som bare hadde et tegn langt passord.

## Diskusjon

Ikke alle brukere er like datakyndige. For å gjøre det vanskeligere for en angriper, bør man kreve at passordet har en viss lengde. Jo lenger og mer komplisert et passord er, jo lengere tid vil det ta for en angriper å gjette seg frem til det.

## Anbefalinger

Det bør implementeres et minimumskrav til passordlengde. Det går også an å ha en minimumskompleksitet, som f.eks. at passordet må inneholde minst ett tall.

### **Funn #19: Cross-Origin Resource Sharing**

Innsatsnivå: **MIDDELS**

Funnet på sidene:

10.250.0.3:80/wp-json

10.250.0.3:80/wp-admin/admin-ajax.php

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
6	6	4	5	2	23	MODERAT

## Bevis

Fra HTML-koden til 10.250.0.3:80/wp-json.

```
Allow: GET
Access-Control-Allow-Origin: http://ldhvyqfrmljh.com
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Credentials: true
Content-Length: 1007
Connection: close
Content-Type: application/json; charset=UTF-8
```

## Diskusjon

Cross-Origin-Resource-Sharing (CORS) er en mekanisme som tillater websiden å forespørre data fra andre domener utenfor webside-domenet. Dette utsetter webapplikasjonen for en potensiell sårbarhet som gjør det mulig for en angriper å utføre et såkalt “cache poisoning angrep”, som kan lure brukere inn på en ondsinnet side i stedet for siden vedkommende egentlig ønsket å besøke [22].

## Anbefalinger

Det bør implementeres en “whitelist” for domener som er stolt på, og blokkere alle andre for å minimere risiko. Dersom websidene som implementerer CORS ikke har funksjonalitet som er avhengig av dette, anbefales det å fjerne mekanismen. Se [22] og [23] for mer informasjon om denne sårbarheten.

## Funn #20: Cross-Site Request Forgery

Innsatsnivå: **MIDDELS**

Sårbarheten ble funnet på følgende sider:


10.250.200.3/wp-login.php

10.250.200.3:81/login

10.250.200.3:81/search

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
4	10	2	2	4	22	MODERAT

## Bevis

 <b>Cross-site request forgery</b>	
Issue:	Cross-site request forgery
Severity:	Medium
Confidence:	Tentative
Host:	http://10.250.200.3:8080
Path:	/syscmd
<b>Issue detail</b>	
The request appears to be vulnerable to cross-site request forgery (CSRF) attacks against authenticated users.	

## Diskusjon

Dette er et angrep som tvinger sluttbrukeren til å utføre uønskede kommandoer på en webapplikasjon som de er autoriserte på [24]. Dette kan utføres ved at man lurer brukeren til å trykke på en link, og deretter kan angriperen utføre en rekke operasjoner på webapplikasjonen som sluttbrukeren er autorisert på. Seriositetsgraden av sårbarheten varierer med hvilken sluttbruker som er utsatt. Da operasjonene som utføres er begrenset til hva brukeren er autorisert til, vil et angrep rettet mot en systemadministrator sette hele systemet i fare [24].

## Anbefalinger

God opplæring av brukerne kan være til hjelp. Ellers er det noen andre triks man kan gjøre for å hindre dette problemet. Det går an å implementere en test som sjekker standard "headers" for å verifisere at forespørselen er fra samme maskin som brukeren sitter på, og sjekke etter CSRF-tokener [25]. Se [25] for en mer detaljert forklaring.

## Funn #21: Skript fra andre domener inkludert

Innsatsnivå: **MIDDELS**

Bruk av eksterne skripts har blitt funnet på følgende sider:

10.250.200.3:8080/

10.250.200.3:8080/admin-networking.html

10.250.200.3:8080/admin-system.html

10.250.200.3:8080/admin-users.html



Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
4	4	1	9	3	21	MODERAT

## Bevis

Dette er fra HTML-koden til 10.250.200.3:8080/

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/startbootstrap-sb-admin-2/3.3.7+1/css/sb-admin-2.css">
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/metisMenu/2.7.0/metisMenu.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/startbootstrap-sb-admin-2/3.3.7+1/js/sb-admin-2.js"></script>
```

## Diskusjon

Når en applikasjon har en kjørbare kode inkludert fra et eksternt domene, så kan den kjørbare koden kjøres av nettleseren. Skriptet kan derfor gjøre alt det applikasjonen har rettigheter til, som for eksempel aksessere data og utføre kommandoer [26]. Det anbefales kun å inkludere slik kode fra domener man stoler fullt på.

## Anbefalinger

Kjørbare koder burde ikke være inkludert fra domener som man ikke stoler på. Eksterne skript burde legges inn lokalt på maskinen og kjøres derfra i stedet for å importere de fra et annet domene [26]. Se [26] for mer informasjon.

Gå gjennom alle slike eksterne skripts og vurder om domenet de ligger på kan stoles på.

## Funn #22: Potensiell clickjacking

Innsatsnivå: **LAV**

Muligheter for såkalt "clickjacking" ble funnet på nærmest alle sider av webapplikasjonene.

Skade- potensiale	Mulighet for gjentakelse	Utnyttelses- mulighet	Påvirkede brukere	Sannsynlighet for oppdagelse	Totalt	Risikokategori
5	4	2	5	2	18	MODERAT

## Bevis

På siden 10.250.200.3:80/ er det ingenting som beskytter mot Clickjacking:

```
HTTP/1.1 200 OK
Date: Wed, 15 Nov 2017 09:16:54 GMT
Server: Apache/2.4.10 (Debian)
X-Powered-By: PHP/5.6.25
Link: <http://10.250.200.3/wp-json/>; rel="https://api.w.org/"
Vary: Accept-Encoding
Content-Length: 9542
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en-US" class="no-js">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="profile" href="http://gmpg.org/xfn/11
```

## Diskusjon

“Clickjacking” kan forekomme når en angriper bruker flere transparente lag til å lure brukeren til å klikke på en link til en annen side, når det egentlig var mening å klikke på noe annet. På denne måten kaprer angriperen museklikkene deres, og dette leder dem inn på en annen side der man igjen man bli lurt av andre ting [27]. Et annet eksempel er å få brukeren til å skrive inn passordet i feil tekstboks, så blir dette sendt til angriperen.

## Anbefalinger

Det anbefales å ta en titt på [27] for å beskytte seg mot “clickjacking”.

## 2.6 Lite kritiske funn (Score 5-10)

Et lite kritisk funn bør behandles etter at alle andre sårbarheter er fjernet. Disse vil av stor sannsynlighet ikke bli utnyttet, og selv ved et suksessfullt angrep vil konsekvensene være minimale. De bør likevel fjernes når man har kapasitet til dette.

Ingen sårbarheter ble funnet i denne risikokategorien.

### 3 Vurdering og anbefaling

Penetrasjonstesten har avslørt et bredt antall sikkerhetshull, derav flere klassifisert som kritiske. Prosjektgruppen fikk i utgangspunktet ikke noe informasjon om hva bedriften driver med. Det ble ikke nevnt noe om hvor sensitiv informasjon bedriften har lagret, hvilke kritiske prosesser de kjører og heller ikke hvilke webapplikasjoner de tilbyr. Det er derfor vanskelig å avgjøre hvor alvorlige de forskjellige sårbarhetene er, men mange av de avdekkede feilene er likevel så alvorlige at de kan få negative konsekvenser innen ethvert bruksområde.

Webapplikasjonene på port 80 var de med klart færrest sårbarheter. Det brukes en eldre versjon av wordpress, og dette bør oppdateres snarest mulig. Om passord i tillegg gjøres mer sikkert ved bl.a. kryptering, kan disse webapplikasjonene fortsette å brukes. Noen mindre alvorlige sårbarheter nevnt over bør likevel fikses i løpet av relativt kort tid.

Applikasjonene på port 81 og 8080 inneholder mer kritiske hull, både kvalitativt og kvantitativt. Disse alvorlige sårbarhetene kan gi en angriper administratortilgang, noe som selvsagt vil være katastrofalt for bedriften. Ettersom det er et så bredt spekter av sikkerhetshull vil antakelig det minst ressurskrevende være å bygge opp nye webapplikasjoner på nytt fra bunnen av, der man tenker mer på sikkerhet underveis i design- og utviklingsprosessen. Denne anbefaling er også tatt på bakgrunn av at webapplikasjonene ikke er altfor omfattende.

Om det i stedet blir valgt å forsøke å rette opp i alle hullene på de nåværende applikasjonene, må man være klar over at dette kommer til å bli en svært ressurs- og tidskrevende prosess. Det er da også viktig å være klar over at siden en angriper allerede kan ha hatt tilgang til systemet en viss tid, kan det ha blitt lagt inn sårbarheter allerede. Disse kan være svært vanskelig å finne, og derfor blir det vanskelig å være helt sikker på at siden er trygg, selv etter at man har fjernet alle de opprinnelige sårbarhetene.

# Litteraturliste

- [1] OWASP, *Owasp Top 10 Vulnerabilities*, 2017. Hentet fra: [https://www.owasp.org/images/b/b0/OWASP\\_Top\\_10\\_2017\\_RC2\\_Final.pdf](https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf). Hentet: 13.11.2017.
- [2] OWASP, *Threat Risk Modeling*, 2017. Hentet fra: [https://www.owasp.org/index.php/Threat\\_Risk\\_Modeling](https://www.owasp.org/index.php/Threat_Risk_Modeling). Hentet: 19.11.2017.
- [3] TechTarget, *port*, 2006. Hentet fra: <http://searchnetworking.techtarget.com/definition/port>. Hentet: 19.11.2017.
- [4] Acunetix, *What is local file inclusion (LFI)?*, 2017. Hentet fra: <https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/>. Hentet: 17.11.2017.
- [5] C. Keigher, *Remote code execution on misconfigured systems using Werkzeug*, 2014. Hentet fra: <http://blog.keigher.ca/2014/12/remote-code-execution-on-misconfigured.html>. Hentet: 17.11.2017.
- [6] OWASP, *OS Injection*, 2017. Hentet fra: [https://www.owasp.org/index.php/OS\\_Injection](https://www.owasp.org/index.php/OS_Injection). Hentet: 19.11.2017.
- [7] A. Weiss, *How to prevent SQL-injection attacks*, 2016. Hentet fra: <https://www.esecurityplanet.com/hackers/how-to-prevent-sql-injection-attacks.html>. Hentet: 17.11.2017.
- [8] Aaron Weiss, *How to prevent DoS Attack*, 2017. Hentet fra: <https://www.esecurityplanet.com/network-security/how-to-prevent-dos-attacks.html>. Hentet: 19.11.2017.
- [9] OWASP, *Testing for Remote File Inclusion*, 2017. Hentet fra: [https://www.owasp.org/index.php/Testing\\_for\\_Remote\\_File\\_Inclusion](https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion). Hentet: 19.11.2017.
- [10] Acunetix, *What is remote file inclusion (RFI)?*, 2017. Hentet fra: <https://www.acunetix.com/blog/articles/remote-file-inclusion-rfi/>. Hentet: 17.11.2017.
- [11] Portswigger Web Security, *External service interaction (DNS)*, 2017. Hentet fra: [https://portswigger.net/kb/issues/00300200\\_externalserviceinteractiondns](https://portswigger.net/kb/issues/00300200_externalserviceinteractiondns). Hentet: 17.11.2017.

[12] CVE Details, *Apache 2.4.10 Security Vulnerabilities*, 2017. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-45/product\\_id-66/version\\_id-177881/Apache-Http-Server-2.4.10.html](https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-177881/Apache-Http-Server-2.4.10.html). Hentet: 19.11.2017.

[13] CVE Details, *Wordpress Security Vulnerabilities*, 2017. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-2337/product\\_id-4096/](https://www.cvedetails.com/vulnerability-list/vendor_id-2337/product_id-4096/). Hentet: 19.11.2017.

[14] CVE Details, *Ubuntu Security Vulnerabilities*, 2010. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-51/product\\_id-80/Ubuntu-Ubuntu-Linux.html](https://www.cvedetails.com/vulnerability-list/vendor_id-51/product_id-80/Ubuntu-Ubuntu-Linux.html). Hentet: 19.11.2017.

[15] D. Zimmermann, *The Finger User Information Protocol*, 1991. Hentet fra: <https://tools.ietf.org/html/rfc1288>. Hentet: 17.11.2017.

[16] Penetration Testing Lab, *Unix User Enumeration*, 2012. Hentet fra: <https://pentestlab.blog/tag/finger/>. Hentet: 17.11.2017.

[17] Instant SSL, *HTTP to HTTPS*, Ukjent. Hentet fra: <https://www.instantssl.com/ssl-certificate-products/https.html>. Hentet: 17.11.2017.

[18] OWASP, *Cross-Site Scripting (XSS)*, 2016. Hentet fra: [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)). Hentet: 17.11.2017.

[19] OWASP, *XSS Prevention Cheat Sheet*, 2017. Hentet fra: [https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet). Hentet 17.11.2017.

[20] Portswigger Web Security, *Cleartext submission of password*, Ukjent. Hentet fra: [https://portswigger.net/kb/issues/00300100\\_cleartextsubmissionofpassword](https://portswigger.net/kb/issues/00300100_cleartextsubmissionofpassword). Hentet: 19.11.2017.

[21] Portswigger Web Security, *Cross-domain Referer leakage*, Ukjent. Hentet fra: [https://portswigger.net/kb/issues/00500400\\_crossdomainrefererleakage](https://portswigger.net/kb/issues/00500400_crossdomainrefererleakage). Hentet: 19.11.2017.

[22] MDN Web Docs, *Cross-Origin Resource Sharing (CORS)*, 2017. Hentet fra: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. Hentet: 19.11.2017.

[23] Veracode, *Cache Poisoning Attack*, 2017. Hentet fra: <https://www.veracode.com/security/cache-poisoning>. Hentet: 19.11.2017.

[24] OWASP, *Cross-site Request Forgery*, 2017. Hentet fra:  
[https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)). Hentet:  
19.11.2017.

[25] OWASP, *Cross-site Request Forgery (CSRF) Prevention Cheat Sheet*, 2017.  
Hentet fra:  
[https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet). Hentet: 19.11.2017.

[26] Portswigger Web Security, *Cross-domain script include*, 2017. Hentet fra:  
[https://portswigger.net/kb/issues/00500500\\_crossdomainscriptinclude](https://portswigger.net/kb/issues/00500500_crossdomainscriptinclude). Hentet:  
19.11.2017.

[27] OWASP, *Clickjacking*, 2017. Hentet fra:  
<https://www.owasp.org/index.php/Clickjacking>. Hentet: 19.11.2017.

Teknisk rapport for  
Penetrasjonstesting av webapplikasjoner for  
South-Dyersville industries

Prosjektgruppe: 3

Deltakere: Fredrik Bjerkø  
Kristen Arntsberg  
Eivind Børstad  
Gard Klemetsen

Utgivelsesdato: 23.11.2017

Begrensning: Åpen

# Innholdsfortegnelse

<b>Kontaktinformasjon</b>	<b>56</b>
<b>Terminologi</b>	<b>57</b>
<b>Sammendrag</b>	<b>58</b>
<b>1 Prosjektbeskrivelse</b>	<b>59</b>
1.1 Bedriftsbeskrivelse	59
1.2 Forutsetninger for penetrasjonstesten	59
1.3 Verktøy	59
<b>2 OWASP Topp 10</b>	<b>61</b>
2.1 A1 – Injection	62
2.2 A2 – Broken Authentication and Session Management	63
2.3 A3 – Sensitive Data Exposure	64
2.4 A4 – XML External Entity (XXE)	66
2.5 A5 – Broken Access Control	66
2.6 A6 – Security Misconfiguration	67
2.7 A7 – Cross-Site Scripting (XSS)	70
2.8 A8 – Insecure Deserialization	71
2.9 A9 – Using Components with Known Vulnerabilities	71
2.10 A10 – Insufficient Logging and Monitoring	72
2.11 - Andre funn	72
<b>Litteraturliste</b>	<b>75</b>



## Kontaktinformasjon

<b>Kunde</b>	
<b>Bedrift</b>	South-Dyersville industries
<b>Kontaktperson</b>	Martin Sundhaug
<b>Telefon</b>	
<b>Epost</b>	martinsundhaug@gmail.com
<b>URL/IP-adresse</b>	10.250.200.3

<b>Pentestgruppe</b>	
<b>Bedrift</b>	Gruppe 3
<b>Kontaktperson</b>	Eivind Børstad
<b>Telefon</b>	47362130
<b>Epost</b>	eivindborstad@gmail.com
<b>Bedriftsadresse</b>	Raveien 215
<b>Postnummer</b>	3184 Borre
<b>URL</b>	Ingen

# Terminologi

Rapporten har blitt skrevet på norsk i den grad det lar seg gjøre. En del engelske fagbegreper er likevel såpass etablerte at de har blitt valgt å ikke oversettes. Flere av disse finnes det heller ingen gode ord for på norsk. En oversikt over disse kommer nedenfor, men siden dette er en teknisk rapport vil ikke grunnleggende tekniske begreper bli forklart.

**Blacklist** - Liste over verdier som er ulovlige. Alle andre verdier er tillatt.

**Clickjacking** - Teknikk for å lure en bruker til å klikke på noe annet.

**Cross-domain referer leakage** - Lekkasje av informasjon på tvers av domene.

**Cross-origin request forgery** - Forfalskning av forespørsel på tvers av domene.

**Cross-origin resource sharing** - Bruk av ressurser fra andre domener.

**Cross-site scripting (XSS)** - Kjøring av vilkårlig javascript-kode i nettleseren.

**Social Engineering** - Lure noen som har tilgang til en side gjennom det virkelige liv.

**Whitelist** - Liste over verdier som er lovlige. Alle andre verdier er forbudt.

# Sammendrag

Rapporten beskriver funn av sårbarheter etter at prosjektgruppe 3 utførte en rekke tester mot South-Dyersville industries sine webapplikasjoner fra et teknisk synspunkt. Den går dypt inn på hver sårbarhet og beskriver hvordan disse kan fikses ved bruk av et teknisk språk. For bedriftens ledelse og andre ikke-tekniske personer anbefales det i hovedsak å lese den administrative rapporten i stedet.

I kap. 1 gis det en kort beskrivelse av prosjektet. Dette kapittelet er identisk med det første kapittelet i den administrative rapporten. Unntaket er at det her er lagt til et underkapittel 1.3, som kort reder for hvilke verktøy som har blitt brukt for å gjennomføre penetrasjonstesten.

Kap. 2 gir en grundig gjennomgang av "OWASPs Topp 10 sårbarheter" fra 2017 og klassifiserer funnene i disse ti kategoriene. Det er blitt funnet sårbarheter innenfor nesten alle kategorier, og bedriften har en stor jobb foran seg for å rette opp i disse.

Noen av de mest kritiske sårbarhetene som har blitt funnet er:

- Muligheter for OS-injeksjon gjennom URL-parametre
- Muligheter for SQL-injeksjon gjennom URL-parametre
- Flere filer med sensitiv informasjon som er tilgjengelig for alle
- En passordfil ligger tilgjengelig for alle, som gir tilgang til en monitoreringsside
- Cross-Site Scripting (XSS) muligheter
- En side med muligheter for både LFI og RFI

Gjennom den administrative rapporten anbefaler prosjektgruppen at bedriften burde vurdere å bygge opp webapplikasjonene fra bunn av igjen. Hvis det likevel avgjøres at de eksisterende applikasjonene heller skal gjøres trygge, så er denne rapporten et godt utgangspunkt for å gå gjennom sårbarheter og fjerne dem. For hver sårbarhet gis det først oversikt over alle funn, deretter relevant teori om sårbarheten, og til slutt generelle mottiltak som bør utføres for å fjerne sårbarheter av denne typen.

# 1 Prosjektbeskrivelse

Den 4. november, 2017 fikk prosjektgruppe 3 oppdraget å utføre en omfattende penetrasjonstest mot webapplikasjonene som kjører på 10.250.200.3, for South-Dyersville industries. Det var forutsatt at penetrasjonstesten inkludert rapporter skulle være ferdig innen 30. November.

## 1.1 Bedriftsbeskrivelse

South-Dyersville industries er en nyoppstartet bedrift som benytter seg av en server med IP-adresse 10.250.200.3. Prosjektgruppen har ikke fått noen annen informasjon om bedriften, som preferanser i forhold til presentasjon av resultatene, eller detaljer om hva bedriften holder på med. Resultatene har derfor blitt presentert på en måte som skal virke mest mulig oversiktlig for bedriftsledelsen. Uten å vite hvor mye sensitiv informasjon bedriften holder på, og hvor kritiske operasjoner de utfører, er det vanskelig å vurdere skadepotensialet i forhold til sikkerhetshullene. Det har likevel blitt gjort etter beste evne i DREAD-modellen, som det kan leses om i kap. 2.

## 1.2 Forutsetninger for penetrasjonstesten

Målet med prosjektet var å finne sårbarheter i applikasjonene ved å utføre angrep fra et eksternt synspunkt. Dette innebar at prosjektgruppen kun fikk utdelt bedriftens IP-adresse, og ingenting annet. Ingen av deltakerne hadde vært i kontakt med bedriften tidligere, eller hadde noen form for bruker i systemet deres.

Alle sikkerhetshull som ble funnet har blitt grundig dokumentert. I tillegg har det blitt gitt forslag til hva South-Dyersville industries kan gjøre for å tette disse hullene. Dette presenteres i neste kapittel.

## 1.3 Verktøy

Både manuell testing via nettleser, og automatiske verktøy, har blitt brukt i forbindelse med penetrasjonstesten. Dette har blitt gjort på en slik måte at prosjektdeltakerne først har forsøkt å finne sårbarheter manuelt, for deretter å benytte automatiske verktøy. Deretter har det blitt gjort nye manuelle tester på bakgrunn av funnene de automatiske verktøyene fant.

Verktøyet som hovedsakelig har blitt brukt for å utføre automatiske tester er BurpSuite. I dette programmet benyttet gruppen seg hovedsakelig av funksjonalitetene "Spider", "Scanner" og "Intruder". "Spider" brukes for å finne alle

undersider på et domene. "Scanner" benyttes for å utføre tester mot et bredt spekter av vanlige sårbarheter automatisk. "Intruder" er til å teste mye forskjellige input mot en konkret sårbarhet etter at det er oppdaget.

Andre verktøy som har blitt brukt er Nmap, Google Chrome, Firefox og FoxyProxy.

## 2 OWASP Topp 10

Ettersom prosjektgruppen ikke har hatt noen kontakt med bedriften etter at de fikk oppdraget, har ingen informasjon om preferanse i forhold til hvordan funnene ønskes presentert blitt formidlet. Det har derfor blitt valgt å organisere sikkerhetshullene etter sårbarhetstype, i stedet for hvilken port og nettside sårbarheten ligger på.

For å presentere alle funnene på en oversiktlig og god måte blir “OWASP Topp 10”-modellen fra 2017 benyttet [1]. Alle sårbarheter er klassifisert etter den kategorien de hører inn i. Navnene på disse kategoriene er såpass etablerte begreper på engelsk, at de presenteres med det engelske ordet først, før en fri norsk oversettelse i parentes. En oversikt over dem gis her først:

- A1 - Injection (Injeksjon)
- A2 - Broken Authentication and Session Management (Utrygg autentisering og sesjonsstyring)
- A3 - Sensitive Data Exposure (Lekkasje av sensitive data)
- A4 - XEE: XML External Entity (Eksternt enhetsangrep gjennom XML)
- A5 - Broken Access Control (Utrygg tilgangskontroll)
- A6 - Security Misconfiguration (Sikkerhetsfeilkonfigurasjon)
- A7 - XSS: Cross-Site Scripting (XSS)
- A8 - Insecure Deserialization (Usikker deserialisering)
- A9 - Using Components with Known Vulnerabilities (Bruk av komponenter med kjente sårbarheter)
- A10 - Insufficient Logging and Monitoring (Utilstrekkelig logging og overvåking) [1]

Hver av de 10 sårbarhetskategoriene har fått sitt eget underkapittel, fra 2.1 - 2.10. I overskriftene i disse underkapitlene brukes kun de engelske uttrykkene. Sårbarheter som ikke passer inn i noen kategori er plassert nederst, i kap. 2.11. For hver kategori er det en oversikt over hvilke funn som er gjort, en kort teoretisk beskrivelse av hva denne type sårbarhet egentlig er, og forslag til tiltak som kan gjøres for å fjerne sårbarheten.

For å holde listen oversiktlig har det blitt valgt å ikke legge ved bilder som beviser sårbarhetene her. Om man likevel ønsker å se disse finnes de i kap. 2 i den administrative rapporten.

## 2.1 A1 – Injection

### Funn

Webapplikasjonene er sårbare for både OS- og SQL-injeksjoner. OS-injeksjoner er mulig på følgende sider:

10.250.200.3:8080/syscmd?command=date|"kommando som skal utføres"

10.250.200.3:81/sys-test.php?test=|"kommando som skal utføres"

Den førstnevnte er spesielt kritisk da kommandoene blir kjørt med rottilgang.

SQL-injeksjoner er mulig på følgende sider:

10.250.200.3:8080/syscmd?command=

10.250.200.3:81/sys2\_test.php?test=

10.250.200.3:81/register?username=

10.250.200.3:81/search?query=

10.250.200.3:81/user/"NavnPåBruker"

Det var noe varierende hva slags injeksjoner som kunne gjennomføres. Mest kritisk var 10.250.200.3:81/sys2\_test.php?test=, der man kunne hente informasjon rett ut av databasen. På de andre sidene ble ikke informasjonen skrevet rett ut på siden, men det var fortsatt mulig å kjøre injeksjoner.

### Teori

Både OS- og SQL-injeksjoner er svært kritiske, og må behandles umiddelbart.

OS-injeksjoner kan tillate en angriper å kjøre fritt valgte kommandoer på systemet, noe som kan gi nærmest ubegrenset med alvorlige konsekvenser [2]. Når man i ettertid vet at et system har vært åpent for OS-injeksjoner, vil det også være nødvendig å gjøre en evaluering i forhold til hva en angriper allerede kan ha lagt inn i systemet. Dette kan være nye sårbarheter og bakdører som angriperen ønsker å benytte seg av senere, også etter at den initiale OS-injeksjonssårbarheten er fikset.

SQL-injeksjoner er også svært alvorlige da det gir en angriper tilgang til store deler av databasen. Dette vil både kunne føre til lekkasje av sensitiv informasjon, men også muligheter for at angriperen kan manipulere dataene i databasen [3].

## Mottiltak

Valider data fra feltene som tar i mot bruker-input, og GET-variabler som sendes gjennom URL'en. Ha en liste over godkjente tegn som kan skrives inn. F.eks. at det kun er lov å taste inn store og små bokstaver, samt tall. Dette kalles whitelisting [4]. Et alternativ er blacklisting som heller spesifiserer noen tegn det ikke er lov å taste inn [4]. Dette må gjøres på serversiden, da man aldri kan stole på noe som kommer fra klienten. [3][4]

Aldri spesifiser i URL'en hvilken OS-kommando som skal kjøres. Om man absolutt må sende en kommando på denne måten fra klientsiden, bør den sendes som en POST-variabel. I tillegg bør det ikke være selve kommandoen som sendes, men heller en annen tekst eller tallverdi, som deretter oversettes til rett kommando via en tabell på serversiden. Et eksempel på dette er at hvis en variabel sendes til serveren med verdi 5, vet serveren at det betyr at den skal utføre kommandoen "lag en ny fil".

## 2.2 A2 – Broken Authentication and Session Management

### Funn

Applikasjonene har lite sikkerhet som hindrer at en ondsinnet person kan logge inn på andres brukere. Sårbarhetene nevnt under er identiske i forbindelse med innlogging både på side 80, 81 og 8080.

Passord sendes ukryptert, noe som gjør at en angriper som får tak i HTTP-forespørselen ved innlogging umiddelbart kan se passordet til vedkommende bruker. Heller ikke selve forespørselen er kryptert med HTTPS. Det sistnevnte blir hovedsakelig tatt for seg under A6, men bør også gjøres for å hindre at en angriper kan stjele sesjonsinformasjonskapselen til en bruker, og dermed utgi seg for å være vedkommende.

Det er ingen minimumskrav til lengde og kompleksitet på passord. Det er heller ikke spesifisert at passord ikke skal kunne autofullføres av nettleseren.

Spesielt alvorlig er det at gjestebrukeren på port 8080 har blankt passord, noe som gjør at det er svært enkelt å komme inn på denne siden. Dette er en side der man absolutt ikke vil ha uautorisert adgang. Om det er nødvendig å ha en gjestebruker på en slik monitoreringsside, bør også vurderes. I tillegg til dette ligger brukernavn og passord på alle disse brukerne åpent i fila 10.250.200.3:8080/config/. Dette tas næyere for seg i A3.



## **Teori**

Informasjonskapsler er variabler som blir brukt for å identifisere de ulike brukerne på webapplikasjonen. Dersom disse ikke blir håndtert på en trygg og sikker måte, kan angripere overta en annen brukers sesjon på siden. [5]

Uten kryptering av passord risikerer man at en såkalt “Man in the middle” får tak i forespørselen som blir sendt, der passordet står i klar tekst. Da er det bare for angriperen å logge inn på siden med vedkommendes identitet.

Passord som genereres uten noe minimumskrav kan være veldig enkle å knekke.

## **Mottiltak**

Krypter passord på klientsiden før det sendes til serveren. Hele forespørselen bør også krypteres med HTTPS.

Spør om passord på nytt etter en viss tid, og automatisk logg ut når siden lukkes. Sett krav til lengde, symboler og tall av passord. Sett autocomplete=”off” i HTML-koden for å hindre at nettlesere automatisk fyller ut passord [6].

Gi gjestebrukeren på port 8080 et mer komplisert passord, eller fjern brukeren helt.

## **2.3 A3 – Sensitive Data Exposure**

### **Funn**

Det ligger mye unødvendig informasjon tilgjengelig rundt om på nettsidene. Dette gjelder både info om brukere o.l., men i hovedsak informasjon om serveren som en angriper kan bruke til sin fordel. Det mest alvorlige er følgende:

10.250.200.3:8080/config/

Monitoreringsdata som er tilgjengelig etter å ha logget inn med gjestebrukeren på port 8080

10.250.200.3:8080/static/admin.js

10.250.200.3:8080/static/common.js

10.250.200.3:8080/static/js.cookie.js

All kontaktinformasjon inkl. pgp-nøkkel for alle brukere på port 81

10.250.200.3:81/server-status

10.250.200.3:81/phpinfo.php

10.250.200.3:80/wp-json/

10.250.200.3:80/wp-json/oembed/1.0/

10.250.200.3:80/wp-content/themes/twenty十六teen/js/html5.js

10.250.200.3:80/wp-includes/js/jquery/jquery.js  
10.250.200.3:80/wp-includes/js/wp-embed.min.js  
10.250.200.3:80/wp-includes/wlwmanifest.xml

Spesielt den øverste fila her er alvorlig. Den inneholder brukernavn og passord på alle brukerne på port 8080, og er kanskje det mest kritiske sikkerhetsbruddet av alle. Med dette kan man logge seg inn på en monitoreringsside der man har mulighet til å gjøre mye skade med brukerne gjest, admin og super. Superbrukeren kan riktignok ikke velges i innloggingsmenyen, men ved å endre på HTML-koden på siden er det likevel mulig å logge inn med denne. Les mer om dette i A2 og A6.

LFI (Lokal filinkludering) tillater en angriper å lese fritt valgte filer på serveren, og gjør at nærmest ingen informasjon er beskyttet [7]. Dette er for øyeblikket mulig på følgende side:

10.250.200.3:81/contactus.php?f="filnavn"

Det er her også mulig å gjøre ekstern filinkludering (RFI), som står nøyere forklart i A5.

Det er mangel av kryptering på siden. Dette blir tatt for seg i A6, men nevnes her siden det også øker sjansene for at sensitiv informasjon slipper ut. Der blir det også tatt for seg at ikke alle feilmeldinger er standardiserte, noe de bør være.

Penetrasjonstestens mål har vært webapplikasjoner, men det er verdt å nevne at fingerporten (port 79) står åpen. En angriper kan bruke denne porten for å få tak i informasjon om bl.a. brukerne på systemet [8].

## Teori

Sensitiv informasjon må behandles på en sikker måte slik at ikke uønskede får tak i den. En besøkende på nettsidene bør kun se den informasjonen som er nødvendig for dem, og all annen informasjon bør ligge trygt og utilgjengelig på serveren.

LFI (lokal filinkludering) kan utnyttes når innholdet av en fil skal vises på nettsiden, og hvilken fil dette er har blitt spesifisert i URL'en. En angriper kan endre navnet på filen til en annen fil vedkommende ønsker å se. Uten begrensninger på hva slags filnavn som kan skrives inn, har en angriper i praksis tilgang til alt i filsystemet [7].

Finger-protokollen (port 79) brukes for å gi informasjon om et system eller en person som bruker systemet. Den kan fortelle hvilke brukere som er logget på systemet, samt info av typen e-postadresse og navn. Det sier seg selv at å la hvem som helst få tak i slik informasjon kan få negative følger [8].

## **Mottiltak**

Gå gjennom alle sider som ligger ute på webserveren, og i hvertfall de som er nevnt over, og vurder hva av dette som egentlig bør ligge ute for allmennheten. Spesielt fila 10.250.200.3:8080/config/ må fjernes umiddelbart!

Fjern muligheten for LFI ved å spesifisere filnavn på en annen måte enn i URL'en. Om filnavn må sendes fra klientsiden på denne måten, bruk en slags oversettelse, der en variabelverdi tilsvarer en bestemt fil, men at verdien ikke er selve filnavnet.

Steng fingerporten fra å kunne nås utenfra. Dette kan gjøres i brannmuren på serveren.

## **2.4 A4 – XML External Entity (XXE)**

### **Funn**

Det er ikke funnet noen sårbarheter av denne typen [1].

## **2.5 A5 – Broken Access Control**

### **Funn**

Det er flere sårbarheter som kunne vært klassifisert under dette, men som i stedet har blitt satt i kategorier de passer bedre under. Dette gjelder lokal filinkludering (LFI) og sensitiv data som ligger ute, som kan leses om i A3, og python-debuggeren, som kan leses om i A6.

Det er også mulig å gjøre ekstern filinkludering (RFI) på siden:

10.250.200.3:81/contactus.php?f="url"

Dette innebærer at en angriper f.eks. kan plassere URL'en til en nettside som parameterverdi, og innholdet på den siden vil bli presentert på siden [9][10]. Innholdet vises kun i tekst, dvs. at koden ikke blir tolket og oversatt. Dette reduserer faren noe, men det bør fortsatt fjernes. Dette er på akkurat samme side og parameter som problemet med LFI som ble gjennomgått i A3.

Ekstern tjenesteinteraksjon på siden 10.250.200.3:81/sys-test.php?test=google.com.

"Cross-origin resource sharing" ble funnet på siden 10.250.200.3:80/wp-json.

## Teori

Ekstern filinkludering (RFI) fører som regel til at en ekstern fils innhold blir vist på siden, men kan i verste fall føre til at kode kjøres på serveren eller hos klienten. Det minner veldig om lokal filinkludering (LFI), men forskjellen ligger i at filen som leses ikke ligger på serveren, men et annet sted på internett [9].

RFI har ikke altfor mye med aksesskontroll å gjøre, men har likevel blitt plassert i denne kategorien. Årsaken til dette er at det i "OWASP Topp 10 2013" tilhørte kategorien "Insecure Direct Object References", som i 2017 er en av to kategorien som har blitt slått sammen til denne [1].

Ekstern tjenesteinteraksjon går ut på å få applikasjonen til å arbeide med en ekstern tjeneste, for eksempel web- eller mailserver [7]. Ekstern tjenesteinteraksjon er ikke en sårbarhet i seg selv, men angriperen kan bruke dette til sin fordel. Ved forekomsten på denne siden må webserveren forespørre en DNS-server for å få finne IP-adressen til google.com [11]. Se [11] for mer informasjon.

"Cross-origin resource sharing" er en sårbarhet som gjør det mulig for et angrep kalt for "cache poisoning angrep" [12]. Se [12] for mer informasjon.

## Mottiltak

Hindre RFI ved å fjerne URL-parametre som spesifiserer filnavn. Avgjør heller hvilken fil som skal åpnes på serversiden, og bare send en variabelverdi som serveren kan oversette til et filnavn. Se også mottiltak for LFI i A3. RFI og LFI oppstår som regel sammen. [7][9][10]

For å unngå ekstern tjeneste-interaksjon kan det være lurt å sette opp en "whitelist" på alle tjenester serveren skal få lov til å bruke, mens den skal blokkere alle andre tjenester [4][11]. Hvilke nettsadresse som skal sjekkes bør heller ikke sendes i URL'en.

For å hindre et "cache poisoning angrep", kan man implementere en "whitelist" for domener som er stolt på og blokkere alle andre tjenester [4][12].

## 2.6 A6 – Security Misconfiguration

### Funn

Webapplikasjonen har en rekke svakheter når det kommer til sikkerhetskongfigurasjoner. Dette inkluderer:

Det benyttes ikke standard feilmeldinger på alle sider. Ved spesifikke feilmeldinger får angriperen informasjon om hva som gikk galt, noe som kan utnyttes videre. Sider dette gjelder på er:

10.250.200.3:81/register  
10.250.200.3:81/login  
10.250.200.3:81/contactus.php  
10.250.200.3:81/sys-test.php  
10.250.200.3:81/sys2\_test.php  
10.250.200.3:8080/getetc  
10.250.200.3:8080/syscmd

Det er en gjestebroker tilgjengelig som kan logge seg inn på siden <http://10.250.200.3:8080> med blankt passord. Denne siden tar for seg overvåking/monitorering av serveren. Det at en gjest skal få tilgang til dette er unødvendig og en høy sikkerhetsrisiko. En fil med passord til alle brukerne ligger også ute på [10.250.200.3:8080/config/](http://10.250.200.3:8080/config/). Dette kan leses mer om i A2 og A3.

Etter å ha logget inn på siden nevnt over, har man tilgang til en python-debugger på:

<http://10.250.200.3:8080/getetc>  
<http://10.250.200.3:8080/syscmd>

Her er det mulig å kjøre python-kommandoer, og via dette også OS-kommandoer på serveren. Dette gjør at en angriper kan få til nesten hva som helst, spesielt siden kommandoene blir kjørt med rottilgang.

Kommunikasjon på siden skjer ukryptert med HTTP. Dette øker faren for at en inntrenger kan tukle med kommunikasjonen. Spesielt alvorlig er det at passord blir mottatt i klartekst ved brukerinlogging, noe det kan leses mer om i A2.

Såkalte robots.txt-filer ligger ute på:

10.250.200.3/robots.txt  
10.250.200.3:81/robots.txt

Dette er ikke et sikkerhetsbrudd, men det er viktig å være klar over at mange angripere vil søke spesielt gjennom filene som er listet opp her. Grunnen til dette er at dette er filer webdesigner har bedt søkemotorer om ikke å oppgi.

Mulighet for "clickjacking" ble funnet på nærmest alle sider.

Finger-porten er åpen, noe som er tatt for seg i A3.

## Teori

Feilkonfigurasjoner innen sikkerhet kan få svært alvorlige konsekvenser, i den forstand at en angriper kan finne smutthull for å utføre operasjoner eller se informasjon vedkommende ikke var tiltenkt.

HTTPS er en kryptert versjon av HTTP, som kjører på port 443 i stedet for 80. Dette minker betydelig sannsynlighet for at en angriper kan få tak i innholdet som sendes frem og tilbake mellom klient og webserver [13].

Robots.txt filer benyttes for å gi søkemotorer informasjon om hvilke undersider på et domene som skal listes opp, og er en del av såkalt søkemotoroptimalisering (SEO) [14]. I disse filene listes det blant annet opp hvilke filer man ikke ønsker at søkemotorer skal oppgi [14]. Dette kan f.eks. være fordi det ligger informasjon på disse sidene som man ikke vil alle skal se. For en angriper vil det derfor være naturlig å sjekke innholdet i disse filene nøye.

“Clickjacking” kan forekomme når en angriper bruker flere “lag” med bokser som popper opp for å lure brukeren til å klikke på en link til en annen side. På denne måten blir sluttbrukeren “kapret” [15].

## Mottiltak

Gi standard feilmeldinger når noe uventet oppstår. Ikke skriv ut en detaljert beskrivelse av hva som gikk galt.

Sett et bedre passord på gjestebrukeren på port 8080, eller fjern brukeren helt. Ikke la fila med passord ligge ute så alle kan se den!

Helst fjern python-debuggeren, og dersom den trengs må den kun være tilgjengelig for administrator/superbruker.

Krypter alt som sendes mellom klienten og serveren ved å bruke HTTPS i stedet for HTTP. Passordet må i det minste krypteres ved innlogging.

Det finnes flere metoder for å beskytte seg mot “clickjacking”, og det anbefales å se på [15] og [16]. Spesielt [16] inneholder flere gode metoder. Et eksempel går på å få HTTP-responsheaderen til å indikere om en nettleser skal få lov til å gjengi en nettside. Nettsider kan unngå “clickjacking” ved å sørge for at innholdet på nettsiden ikke er en del av en annen nettside og omvendt [16].

## 2.7 A7 – Cross-Site Scripting (XSS)

### Funn

Det er påvist tilfeller hvor webapplikasjonen er sårbar for reflektert Cross-Site Scripting (XSS). Cross-Site Scripting kan plasseres i parameteren i URL'en på siden:

10.250.200.3:81/register?username=

I tillegg kan javascript-kode plasseres direkte i URL-filstien her:

10.250.200.3:81/user/Pier.Wrightson/key.asc

10.250.200.3:81/user/Pier.Wrightson/me.jpg

Koden plasseres rett etter slutten på den opprinnelige URL'en. Pier.Wrightson er eksempelet på en registrert bruker på siden, og dette fungerer på samtlige brukere.

I tillegg er det spesifisert at dersom nettlesere har innebygde filtre som hindrer XSS, skal disse ikke benyttes.

### Teori

XSS oppstår når data fra en HTTP-forespørsel blir kopiert rett inn i en HTTP-respons, uten å valideres. Det er da mulig å legge inn javascript-kode som kjøres hos klienten når nettleseren tolker responsen. En angriper kan utnytte dette ved f.eks. å sende en link til en bruker, som ved å trykke på linkene går inn på siden, der det kjøres javascript-kode etter angriperens ønske. Dette kan for eksempel sende brukerens sesjonsinformasjonskapsel til angriperen, som gjør at angriperen senere kan gå inn på siden innlogget som brukeren. [17]

Reflektert XSS vil si at javascript-koden kjøres umiddelbart etter at den er sendt inn. Det motsatte er lagret XSS, der koden blir lagret i en database eller lignende. Hver gang en bruker henter ut innholdet fra databasen, vil koden kjøres. Dette er altså mer skadelig enn reflektert XSS. [17]

### Mottiltak

En rekke tiltak kan tas i bruk for å unngå Cross-Site Scripting. Definer regler for hva som ikke er god tatt av input. Dersom disse brytes, avbryt slik at eventuell ondsinnet kode ikke kjøres [18]. Det kan brukes whitelisting eller blacklisting, der man enten tillater eller forbyr visse tegn/sekvenser av tegn [4].

Skrus også på nettleseres innebygde filtre ved å sette "X-XSS-Protection: 1;" i HTML-koden. [19]

## 2.8 A8 – Insecure Deserialization

### Funn

På registreringsiden på port 81 (10.250.200.3:81/register), er det mulig å laste opp et bilde. Det er ingen sjekk for at det som blir lastet opp faktisk er et bilde, og ikke en annen type fil, med f.eks. kjørbare kommandoer. Det er ikke nødvendigvis sikkert man får kjørt disse kommandoene, og at dette får noen konsekvenser. For sikkerhetsskyld bør det allikevel være en funksjonalitet som validerer filinnholdet og filtypen.

### Teori

Ved å la hvem som helst kunne laste opp selvvalgte filer til systemet, er det mulig å plassere ondartede koder og programmer på serveren. Om disse kan bli satt til å kjøres, kan det få alvorlige konsekvenser.

### Mottiltak

Valider filtypen, og aller helst innholdet også, med en enkel sjekk. Om det er mistanker rundt fila, slettes den umiddelbart.

## 2.9 A9 – Using Components with Known Vulnerabilities

### Funn

Webapplikasjonene på port 80 og 81 kjøres med en Apache v2.4.10 webserver. Denne versjonen er utdatert og er utsatt for kjente sårbarheter. Dette inkluderer blant annet mulighet for DOS-angrep, buffer overflyt, segmentfeil og autentiseringssvakheter. En full oversikt over kjente sårbarheter er tilgjengelig på [20].

Operativsystemet på serveren ser ut Ubuntu v4.10. Denne versjonen er også utdatert og er sårbar for lokalt DOS-angrep. Se [21] for mer informasjon.

På port 80 kjøres webapplikasjonene gjennom WordPress. Det brukes en gammel versjon av WordPress, v4.6. Denne har også kjente sårbarheter, se [22].

Applikasjonene på port 81 kjører en gammel versjon av PHP, v5.6.32. Denne bør også oppdateres.



## **Teori**

Ved å bruke komponenter med kjente sårbarheter i systemet, vil man også gjøre sitt eget system sårbart. Dette medfører at det kan være sårbarheter i en applikasjon uten at programmereren eller designer har gjort noen direkte feil. Programvare bør alltid være oppdatert til nyeste versjon.

## **Mottiltak**

Ved oppdatering av programvare fjerner man disse sårbarhetene. Oppdater enten manuelt hver gang en ny utgivelse er ute, eller ta i bruk automatiske oppdateringer. Når man tar i bruk et nytt program eller komponent, bør man alltid sjekke om det finnes noen kjente sårbarheter med versjonen.

## **2.10 A10 – Insufficient Logging and Monitoring**

### **Funn**

Testing mot webapplikasjonene gir ikke grunnlag til å vurdere om det blir gjort nok logging og monitorering på serveren.

Det har ikke blitt funnet noe som indikerer om det er nok eller ikke er det, og dette bør i stedet vurderes av noen som har tilgang til selve serveren.

## **2.11 - Andre funn**

### **Funn**

Under penetrasjonstestene har det blitt erfart at serveren er svært ustabil og går ned ved mange forespørsler over et kort tidsrom. Dette gjør den svært sårbar for Denial-of-Service (DOS)-angrep. En angriper kan sende en mengde forespørsler til serveren ved bruk av et automatisk verktøy, og serveren vil gå ned. Konsekvensene av dette vil være avhengig av hvor mange besøkende nettstedet er tiltenkt. Spesielt applikasjonene på port 8080 er sårbare mot DOS-angrep.

På flere sider inkluderes det også skript fra andre domener. Disse vil kunne kjøres med samme rettigheter som skript som ligger på selve serveren, og kan når som helst endres av eierne av disse domenene. Dette er ingen sårbarhet i seg selv, men må kun brukes om man stoler 100 % på de som står bak siden man henter skriptet fra. Følgende sider inkluderer slike skript:

10.250.200.3:8080/  
10.250.200.3:8080/admin-networking.html  
10.250.200.3:8080/admin-system.html  
10.250.200.3:8080/admin-users.html

“Cross-site request forgery” kan få en sluttbruker til å utføre uønskede handlinger på en nettside, og følgende sider er sårbare mot dette:

10.250.200.3:80/wp-login.php  
10.250.200.3:81/login  
10.250.200.3:81/search

“Cross-Domain referer leakage” gjør at sensitiv informasjon fra siden kan bli sendt til andre sider, og skjer for øyeblikket på følgende steder:

10.250.200.3:80/  
10.250.200.3:80/wp-login.php  
10.250.200.3:81/sys2\_test.php  
10.250.200.3:8080/  
10.250.200.3:8080/getetc  
10.250.200.3:8080/login  
10.250.200.3:8080/syscmd

## Teori

DOS-angrep utføres ved å sende mange pakker til en webserver over et kort tidsrom, og på den måten overbelaste den. Målet er at denne overbelastning vil hindre andre å laste inn nettsiden [23]. Dette er katastrofalt for nettsider med tusenvis eller millioner av besøkende, som kan tape store beløp bare ved korte nedetider. For nettsider med færre besøkende, og som ikke driver med kritiske operasjoner som banktransaksjoner, er ikke DOS-angrep like alvorlig, så lenge de ikke skjer ofte.

En mer sofistikert form for slike angrep er DDOS-angrep, eller distribuerte DOS-angrep. Dette går ut på å få mange datamaskiner til å utføre DOS-angrep samtidig. Dette benyttes spesielt på nettsider med kraftige servere, der et vanlig DOS-angrep ikke er nok til å ta ned serveren [23].

Inkludering av skript fra andre nettsider gir fordelene at det er tidsbesparende, og ikke tar plass på serveren. Det er likevel viktig å være klar over at det er noen andre som faktisk styrer innholdet i fila, og kan endre dette når de vil. Det er derfor mye tryggere å kopiere innholdet fra skriptet til en egen fil på sin egen server, med mindre man stoler fullt på de som står bak nettsiden man inkluderer fra.

“Cross-site request forgery” er et angrep som tvinger sluttbrukeren til å utføre uønskede kommandoer på en web applikasjon som de er autoriserte på [24]. Dette kan utføres med litt hjelp av “social engineering”, ved at man får brukeren til å trykke på en link. Deretter kan angriperen utføre en rekke operasjoner på webapplikasjonen som sluttbrukeren er autorisert på [24].

“Cross-Domain referer leakage” oppstår når nettsider henter ressurser fra andre sider. Informasjon om siden selv sendes gjerne sammen med denne forespørselen, og gjør at siden man ber om ressursen fra mottar informasjon om denne siden [25]. Dette er ikke veldig kritisk, men dersom man forespør en side man ikke stoler på, må man forsikre seg om at ingen sensitiv informasjon sendes.

## **Mottiltak**

Vurder hvor alvorlig et DOS-angrep er mot nettsiden. Vil en kort nedetid ha store konsekvenser? Hvis svaret er ja bør det umiddelbart bli gjort tiltak. Et åpentbart, men dyrt forslag er å oppgradere webserverene. Maskinvare er likevel dyrt, og ikke alltid en aktuell løsning. Et annet tiltak som kan gjøres, er å benytte såkalte “SYN-informasjonskapsel”. Dette kan leses mer om i [23]. Det anbefales også å oppdatere noen av komponentene som benyttes, som nevnt i A9. Flere av disse er spesielt sårbare mot DOS-angrep.

Vurder om alle skript som er inkludert kommer fra nettsider som kan stoles fullt og helt på. Ved det minste tvil skal man ikke inkludere skript fra disse sidene. Om skriptet likevel virker trygt på nåværende tidspunkt, kan man heller kopiere innholdet til en fil på sin egen server.

God opplæring av brukerne kan være til hjelp og bistå mot “cross-site request forgery”. Det går an å implementere en test som sjekker standard “headers” for å verifisere at forespørselen er fra samme maskin som brukeren sitter på, og sjekke etter CSRF-tokener [13]. Se [26] for en mer detaljert forklaring.

Vurder om sidene man forespør ressurser fra kan stoles på. Hvis ikke må man forsikre seg om at ingen sensitiv informasjon sendes sammen med slike forespørsler. Dette vil i stor grad forhindre “Cross-Domain referer leakage”. Se [25] for mer informasjon.

# Litteraturliste

[1] OWASP, *Owasp Top 10 Vulnerabilities*, 2017. Hentet fra:  
[https://www.owasp.org/images/b/b0/OWASP\\_Top\\_10\\_2017\\_RC2\\_Final.pdf](https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf). Hentet:  
13.11.2017.

[2] OWASP, *OS Injection*, 2017. Hentet fra:  
[https://www.owasp.org/index.php/OS\\_Injection](https://www.owasp.org/index.php/OS_Injection). Hentet: 19.11.2017.

[3] A. Weiss, *How to prevent SQL-injection attacks*, 2016. Hentet fra:  
<https://www.esecurityplanet.com/hackers/how-to-prevent-sql-injection-attacks.html>.  
Hentet: 17.11.2017.

[4] Finjan Cybersecurity, *Blacklisting vs Whitelisting - Understanding the Security Benefits of Each*, 2017. Hentet fra:  
<https://blog.finjan.com/blacklisting-vs-whitelisting-understanding-the-security-benefits-of-each/>. Hentet: 17.11.2017.

[5] OWASP, *Session hijacking attack*, 2014. Hentet fra:  
[https://www.owasp.org/index.php/Session\\_hijacking\\_attack](https://www.owasp.org/index.php/Session_hijacking_attack). Hentet: 21.11.2017.

[6] W3Schools, *HTML <input> autocomplete Attribute*, 2016. Hentet fra:  
[https://www.w3schools.com/tags/att\\_input\\_autocomplete.asp](https://www.w3schools.com/tags/att_input_autocomplete.asp). Hentet: 21.11.2017.

[7] Acunetix, *What is local file inclusion (LFI)?*, 2017. Hentet fra:  
<https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/>. Hentet: 17.11.2017.

[8] D. Zimmermann, *The Finger User Information Protocol*, 1991. Hentet fra:  
<https://tools.ietf.org/html/rfc1288>. Hentet: 17.11.2017.

[9] OWASP, *Testing for Remote File Inclusion*, 2017. Hentet fra:  
[https://www.owasp.org/index.php/Testing\\_for\\_Remote\\_File\\_Inclusion](https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion). Hentet  
19.11.2017.

[10] Acunetix, *What is remote file inclusion (RFI)?*, 2017. Hentet fra:  
<https://www.acunetix.com/blog/articles/remote-file-inclusion-rfi/>. Hentet: 17.11.2017.

[11] Portswigger Web Security, *External service interaction (DNS)*, 2017. Hentet fra:  
[https://portswigger.net/kb/issues/00300200\\_externalserviceinteractiondns](https://portswigger.net/kb/issues/00300200_externalserviceinteractiondns). Hentet  
17.11.2017.

[12] MDN Web Docs, *Cross-Origin Resource Sharing (CORS)*, 2017. Hentet fra: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. Hentet: 19.11.2017.

[13] Instant SSL, *HTTP to HTTPS*, Ukjent. Hentet fra: <https://www.instantssl.com/ssl-certificate-products/https.html>. Hentet: 17.11.2017.

[14] MOZ, *Robots.txt*, 2017. Hentet fra: <https://moz.com/learn/seo/robotstxt>. Hentet: 21.11.2017.

[15] OWASP, *Clickjacking*, 2017. Hentet fra: <https://www.owasp.org/index.php/Clickjacking>. Hentet: 19.11.2017.

[16] OWASP, *Clickjacking Defense Cheat Sheet*, 2017. Hentet fra: [https://www.owasp.org/index.php/Clickjacking\\_Defense\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet). Hentet: 19.11.2017.

[17] OWASP, *Cross-Site Scripting (XSS)*, 2016. Hentet fra: [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)). Hentet: 17.11.2017.

[18] OWASP, *XSS Prevention Cheat Sheet*, 2017. Hentet fra: [https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet). Hentet 17.11.2017.

[19] MDN Web Docs, *X-XSS-Protection*, 2017. Hentet fra: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>. Hentet: 21.11.2017.

[20] CVE Details, *Apache 2.4.10 Security Vulnerabilities*, 2017. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-45/product\\_id-66/version\\_id-177881/Apache-Http-Server-2.4.10.html](https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-177881/Apache-Http-Server-2.4.10.html). Hentet: 19.11.2017.

[21] CVE Details, *Ubuntu Security Vulnerabilities*, 2010. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-51/product\\_id-80/Ubuntu-Ubuntu-Linux.html](https://www.cvedetails.com/vulnerability-list/vendor_id-51/product_id-80/Ubuntu-Ubuntu-Linux.html). Hentet: 19.11.2017.

[22] CVE Details, *Wordpress Security Vulnerabilities*, 2017. Hentet fra: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-2337/product\\_id-4096/](https://www.cvedetails.com/vulnerability-list/vendor_id-2337/product_id-4096/). Hentet: 19.11.2017.

[23] Aaron Weiss, *How to prevent DoS Attack*, 2017. Hentet fra:  
<https://www.esecurityplanet.com/network-security/how-to-prevent-dos-attacks.html>.  
Hentet: 19.11.2017.

[24] OWASP, *Cross-site Request Forgery*, 2017. Hentet fra:  
[https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)). Hentet:  
19.11.2017.

[25] Portswigger Web Security, *Cross-domain Referer leakage*, Ukjent. Hentet fra:  
[https://portswigger.net/kb/issues/00500400\\_crossdomainrefererleakage](https://portswigger.net/kb/issues/00500400_crossdomainrefererleakage). Hentet:  
19.11.2017.

[26] OWASP, *Cross-site Request Forgery (CSRF) Prevention Cheat Sheet*, 2017.  
Hentet fra:  
[https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet). Hentet: 19.11.2017.