# Applikationssäkerhet - WEB

## Vad är OWASP?

OWASP är ett öppet, gratis, projekt på Internet för att öka säkerheten i arbete med webbapplikationer. Inom OWASP finns grupper som arbetar både med ”säker kod” av själva applikationerna, med skydd och test av applikationsskydd, med konfigureringstips för webb, applikations- och databasservrar, med säkerhet inom API etc. OWASP har en site med mängder av information och guider för diverse utveckling av säkra webapplikationer – gratis – att ladda ned närhelst man sitter i t.ex. ett webbutvecklingsprojekt.

## Vad är OWASP TOP 10?

OWASP Top 10 är en lista som kommer ut med några års mellanrum, med de vanligaste attackerna/säkerhetshålen på webben, och sedan rankade efter både hur lätta de är att exploatera, vad för följder de kan bära med sig och/eller hur svåra de är att åtgärda. Den nuvarande listan kom 2021 och det ”surras” om OWASP att de kommer att komma med en ny, uppdaterad version under 2022 eller 2023. Vi får väl se. Så länge de här applikationshålen fortfarande är ”vanligast” vid olika säkerhetsproblem kommer de vara kvar på listan. Ändringarna som kom 2021 från den föregående listan (som kom 2017, tror jag) är få, och mindre, och man klumpade ihop några olika punkter som tidigare varit punkter för sig, för att få in ett par nya, uppdaterade, efter att man släppt några nya versioner på mjukvaror, protokoll och serverprocesser mellan de olika versionerna.

Den senaste definitionen på OWASP Top 10 finns här:

<https://owasp.org/www-project-top-ten/>



Lista med OWASP Top 10

Som vi ser när man laser det här finns det normalt sett två sätt att lösa de olika säkerhetsproblemen: Antingen A) konfigurerar man bort dem (eller “debuggea” bort dem när man skriver sin webapplikation) men på själva web- app- eller db-servern. Eller så B) Sätter man in någon form av granskande funktion för att kontrollera att dessa buggar/säkerhetsfel aldrig kan exploateras – i själva nätverket. Som en WAF (Eller olika former av brandväggar och proxy-löningar där den skadliga trafiken kan analyseras och beslut tas om trafikval).

Det finns emellertid en punkt som är mycket svår att bygga bort rent tekniskt. Vilken? Varför?

Svaret är givetvis punkten 10. Med otillräcklig loggning och/eller monitorering.

Det här är ett flerdelat problem – dels finns idag ingen standard för hur loggar ska se ut, vare sig formatmässigt (som med t.ex. syslog) eller vad loggarna ska innehålla. Dels tar loggning tid både i sekunder och processorcykler. Det gör att på gravt lastade tjänster kommer loggmeddelandet vara för sent, i det fall att processorlasten i övrigt är allokerad till att faktiskt betjäna klientens förfrågningar, eller i värsta fallet inte loggas alls. Ofta har olika webbservermjukvaruutvecklare olika lösningar på de problemen – att de loggar kortare rader – eller att man kan ställa in vilka loggmeddelanden man vill ska skrivas ut och hur.

Och här kommer vi till nästa problem – hur det i så fall skall tolkas när man monitorerar tjänsten. En systemadministratör av en webbtjänst ska inte bara veta vad för loggmeddelanden som kan tänkas komma (och beroende på hur tjänstemjukvaran är skriven – hur meddelandet ser ut) eller när det kan förväntas att inte komma – utan också hur man ska kunna övervaka det, för att kunna filtrera ut vilka meddelanden som är viktiga och indikerar på ett säkerhetsproblem med applikationen – och vilka som inte gör det.

Du förstår. Det är rörigt.

Och det är därför det här är ett problem. Det står att genomsnittstiden för att upptäcka säkerhetsproblem genom loggläsning från webbloggar är 200 dagar – alltså mer än ett halvår! (Lägg därtill till att det här är \_just nu\_ och att \_genomsnittstiden\_ förut har varit mycket högre. Men det säger å andra sidan inte så mycket. Det här är ju \_genomsnittstiden\_ - flertalet kommer det ta mycket längre att upptäcka – och dessutom \_av de som hittas\_. Det kan alltså finnas otaliga säkerhetsproblem som aldrig hittas, eller aldrig hittats, genom just logmonitorering!)

Idag finns ett antal – för uppgiften särskilt utvecklade ”säkerhetshändelsemonitoreringssystem” med olika grader av ”machine learning” som kan matas med loggar, för att kunna identifiera säkerhetsproblem. Dessa system brukar kallas SIEM – ”Security Information and Event Management”-system. Här kan logginformationen ”taggas” eller märkas upp för att man snabbt ska kunna jämföra den med tidigare loggmaterial, hitta avvikelser och filtrera bort de loggrader man inte är behjälpt av att monitorera.

(Om du vill kan du nu surfa runt lite med ”fiddler” igång på din dator och du kommer att få OERHÖRT mycket information i fiddler – men där väldigt många kommer att innehålla log-koden ”200 OK” från både server- och klient-sidan i HTTP. Ganska snart är det hundratals. Och att ha hundratusentals, kanske miljontals loggrader som berättar att saker är ”200 OK” är givetvis ingen större nytta med. Vad man är intresserad av är givetvis de gångerna när något visar att det inte är ”OK”...)

Några vanliga SIEM-system är t.ex. Splunk och Logpoint. Det finns betydligt fler – men dessa två har i Sverige blivit något av ”de-facto standard”, där Splunk är allra, allra störst och Logpoint kommer på god andraplats med mindre än 10% av Splunks marknadsandel. Logpoint har emellertid blivit rätt populärt i Sverige, inte minst bland Svenska banker och finansbolag (och betalväxlar som Klarna, Paypal, Payex m.fl) då det är just en svensk produkt.

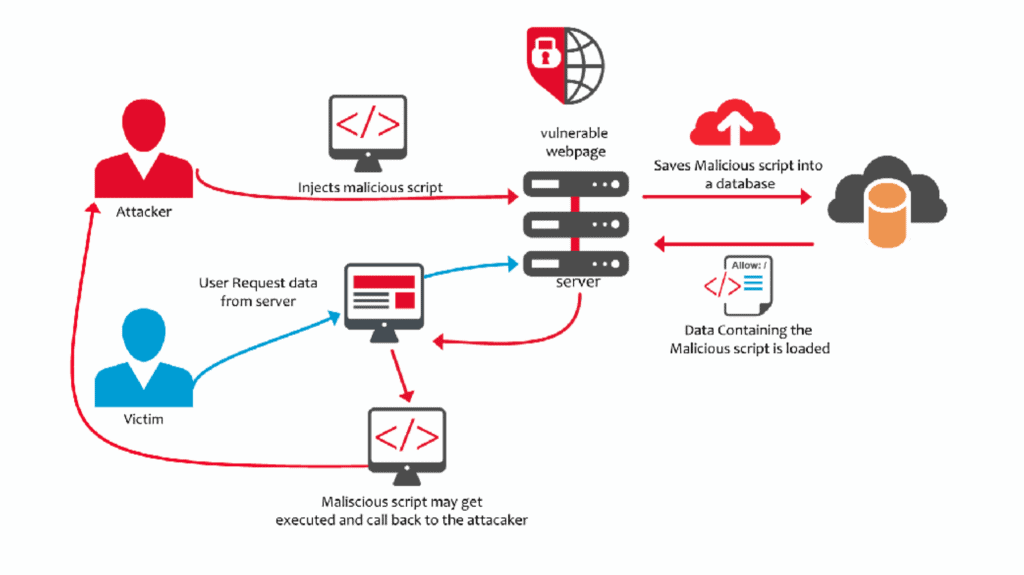
För att lära sig lite mer om SIEM – kolla in <https://www.youtube.com/watch?v=ZuLazPgFtBE>

Ett annat problem med loggning och monitorering som bidrar till de 200 dagarnas genomsnittiliga upptäcktstid är givetvis att loggar – av en hacker – skulle kunna manipuleras, så att säkerhets”events” aldrig rapporteras i loggen. Det vanligaste är emellertid att loggningstjänsterna inte är påslagna, eller att de är så svåra att tyda eller praktiskt filtrera sig igenom att det inte är mänskligt möjligt att praktiskt monitorera dem på ett effektivt sätt.

## Vad är XSS (Cross Site Scripting)?

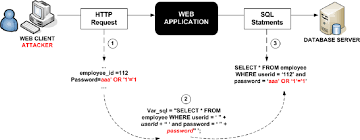
Cross site scripting (XSS) är en teknik som körs i webklienten (browsern) och kommer av att browsern tolkar klientsidesscript och kör dessa för att rendera HTML den kan läsa. En attackerare kan alltså skicka script som svar på en ställd HTTP-fråga till en användare som efterfrågar ett dokument (eller delar av ett dokument) på en webserver – och få detta script att köra i klienten. Typexempel är vid inloggningsrutor och/eller liknande webformulär, där en attackerare – genom att läsa källkoden till själva HTML-sidan – kan ta reda på variabelnamn (och andra liknande parametrar) och sedan skicka script tillbaka till klienten som svar på webklientens fråga/ifyllande av formuläret.

Scripten – å sin sida – ligger inte nödvändigtvis hos den ursprungliga webapplikationen, utan kan finnas lagrade på attackerarens egna server.



## Vad är SQL-injections?

SQL-injections är en attack ”från andra hållet” i klient-server-modellen. Där en klient, i ett webformulär, kan skicka med läsbara instruktioner (t.ex. hos SQL) för att få databasservern i en webb-backend att köra dessa. Detta görs genom att man ”escape:ar” det efterfrågade datat och efter escapekoden skickar med sina körbara kommandon.



Läs nu in OWASPs information om Cross site scripting

<https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)>

Och SQL injections

<https://owasp.org/www-project-top-ten/2017/A1_2017-Injection>