Attackvektorer och mitigerings strategier för webbapplikationer

Miika Salminen
Kandidatavhandling i datavetenskap
Handledare: Dragos Truscan
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2023

Referat

Som efterföljd av digitalisering har internet blivit en eftertraktad utvecklingsplattform för applikationer, men är samtidigt en av de osäkraste. I dag lagras majoriteten av våra data i moln, vilket vi har tillgång till via olika webbapplikationer. Det kan handla om penningtransaktioner, personuppgifter eller hälsovårdsinformation, som finns lagrade i databaser och vanligtvis kan nås genom en webbapplikation. I denna avhandling behandlas de vanligaste attackvektorerna som drabbar dagens webbapplikationer, vilka sårbarheter de utnyttjar och vilka mitigerande strategier som finns. Detta utförs genom att sammanställa dagens webbsäkerhet, med hjälp av att sammanfatta säkerhetsrapporter och vetenskapliga artiklar. Syftet med denna avhandling är att ge läsaren en överblick över dagens webbsäkerhet, kunna identifiera potentiella attackvektorer i sin omgivning och kunna skapa säkrare webbapplikationer i framtiden.

Innehållsförteckning

1. Introduktion

När internet ursprungligen utvecklades var säkerheten inte ännu i fokus. Antaganden på angrepp visade sig vara felaktiga och inställningen till mitigering var problematisk. Ett citat som sammanfattar den tidiga inställningen till säkerhet kommer från Janet Abbate, en datavetenskaplig forskare: “Personer rånar inte banker för att de är osäkra, de rånar banker för det är där pengarna finns”.[1] I dag sker största delen av penningtransaktioner på internet. På grund av detta har säkerheten förstärkts i takt med utvecklingen genom åren, men det finns ännu en mångfald av säkerhetsbrister.

Internet har varit den populäraste plattformen för programvaruproduktion under det senaste årtiondet. JavaScript, ett av de huvudsakliga programmeringsspråken för webbutveckling, firade år 2022 sitt tionde år i följd som det populäraste programmeringsspråket. [2]

Syftet med det här arbetet är att lyfta fram de vanligaste attackvektorerna för dagens webbapplikationer samt diskutera mitigeringsstrategier.

2. Bakgrund

I detta kapitel förklaras termer och begrepp som förekommer i arbetet.

2.1 Webbapplikation

En webbapplikation är en programvara som är lagrad och hyst på en webbserver som kan nås med hjälp av en webbläsare genom en internetanslutning. Fördelar med webbapplikationer i jämförelse med traditionella applikationer, är att användaren inte behöver installera applikationen lokalt på sin maskin, utan applikationen exekverar enbart inom webbläsaren. Detta bidrar till enkel och snabb åtkomst till applikationen, oberoende var i världen användaren befinner sig. En fördel är till exempel att vid uppdateringar av programvaran så kan den nya versionen erbjudas mycket lättare, eftersom användaren inte behöver uppdatera den lokala installationen av applikationen.

Att webbapplikationer är lätt tillgängliga kan även ses som en nackdel, eftersom angripare lättare kan hitta sårbarheter i applikationen. En annan nackdel med webbapplikationer är att de kräver en internetanslutning.

I det här arbetet används begreppet webbapplikation, vilket bör urskiljas från liknande begrepp som ‘webbsida’ eller ‘webbplats’. Begreppet ‘webbapplikation’ skiljer sig från dessa begrepp på grund av dess mer avancerade funktionalitet och dess komplexitet.

2.1.1 HTML, CSS och JavaScript

HTML (Hyper Text Markup Language), CSS (Cascading Style Sheet) och JavaScript är de tre programspråken som är grunden till webbapplikationen i webbläsaren.

2.1.2 Webbläsare

En webbläsare är ett program som ger tillgång till webbsidor och webbapplikationer genom att bearbeta programkoden som fås från webbservern.

2.1.3 Webbserver

2.2 Säkerhet

Säkerhet är ett tillstånd som inte innebär fara eller hot vilket man kan sträva till med hjälp av olika åtgärder för att förminska sannolikheterna för oönskade händelser. Det finns tre huvudsakliga koncept i säkerhet. Sårbarhetskonceptet, vilket syftar på en egenskap som kan potentiellt genom utnyttjande av en sabotör möjliggöra angrepp, vilket bidrar till säkerhetsbrist. Angreppskonceptet syftar på ett direkt anfall och möjligheten av en skadlig inverkan av en sabotör. Det tredje konceptet är mitigering, vilket innebär motåtgärd som förstärker säkerheten till exempel genom att förminska antalet sårbarheter.

Definitionen av säkerhet varierar mycket beroende på kontext. I denna avhandling diskuteras säkerhet i kontext av informationsteknologi med fokus på webbapplikationer.

2.3 Säkerhet i webbapplikationer

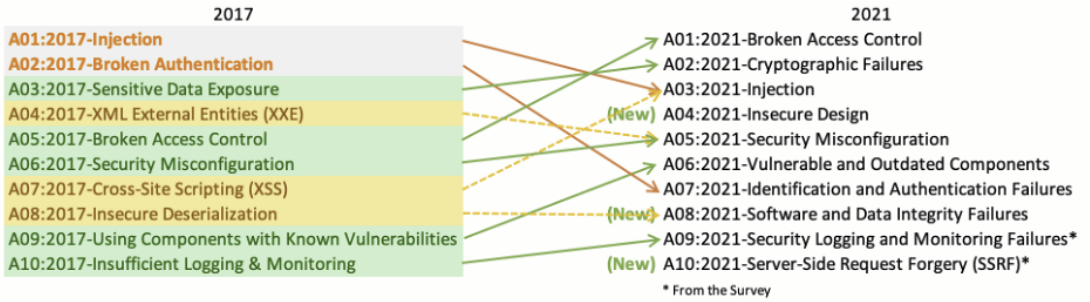
Säkerhet i kontexten av webbapplikationer baserar sig på att säkerställa applikationens önskade funktion och att skydda mot obehörig tillgång, modifiering eller förstörelse av känslig information lagrad i applikationen. De tre koncepten av säkerhet och hur de definieras i kontexten av webbapplikationer, behandlas senare i sina egna kapitel.

2.4 Utveckling

3. Attackvektorer

OWASP (Open Worldwide Application Security Project), är ett icke-vinstdrivande stiftelse som arbetar med att förbättra säkerheten i mjukvarusystem genom att erbjuda olika resurser som skolning och verktyg för utvecklingen av säkra applikationer. OWASP huvudsakliga projekt är ett dokumentation för de tio vanligaste säkerhetsriskerna för webbapplikationen. Dokumentet har benämningen ”OWASP Top 10” och den senaste versionen publicerades i 2021.

Enligt OWASP Top 10 rapporten i 2021, kategoriseras de tre vanligaste attackvektorerna mot webbapplikationer som “fel i åtkomstkontroll", “kryptografiska fel” och “kodinjektion” (eng. Broken Access Control, Cryptographic Failures, Injection). [4] Det här kapitlets syfte är att definiera dessa kategorier, diskutera hur de kan utnyttjas i attack samt mitigeringsstrategierna.



Figur 1. OWASP Top 10 rapporten i 2021, med förändringar från tidigare rapporten i 2017 [4]

3.1 Kodinjektion

Kodinjektion är en attackvektor mot en webbapplikation där skadlig kod skickas med syftet att få applikationen att funktionera på ett oönskat och ofta skadligt sätt. Kodinjektion möjliggörs genom en osäker implementering av insamlingen och lagringen av användarens data samt webbserverns felaktig tolkning av datan. Om data inte valideras och filtreras ordentligt, kan användaren skicka in programkod som i värsta fall exekveras av applikationen. I fall användaren kan exekvera externt kod är möjligheterna för olika attacker vidsträckta. Attacken kan riktas exempelvist mot databaser, för att nå, modifiera och radera innehåll. [5] Angriparen kan också använda sig av kodinjektion för att sprida och exekvera skadliga skript i andra användarnas webbläsare. I det här delkapitlet behandlas några specifika injektionsattacker i detalj.

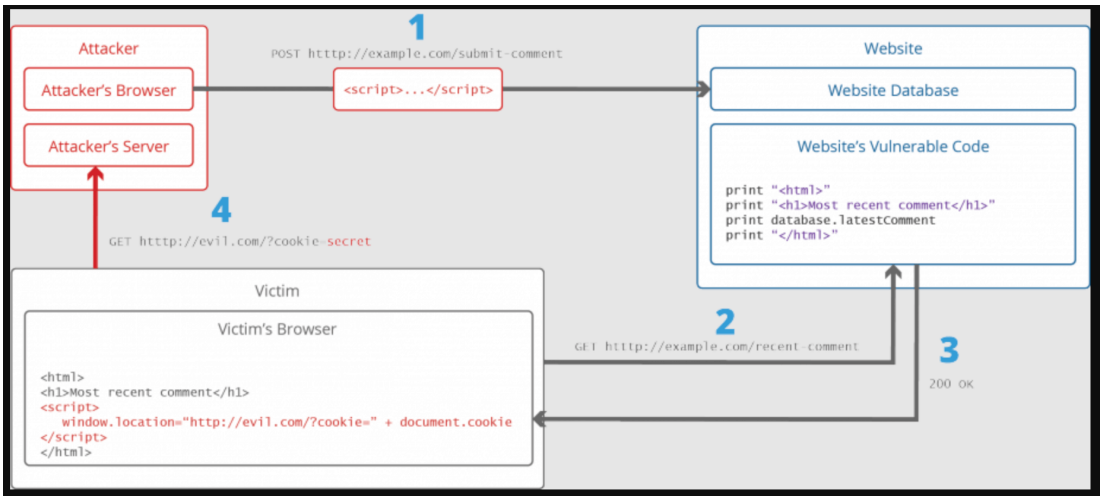
3.1.1 Webbkodinjektion

Webbkodinjektion (eng. Cross-site scripting) (XSS), är en metod för angrepp av webbapplikationer. Angriparen utnyttjar möjligheten att skicka data till webbservern genom att injicera sitt eget programkod i form av HTML och Javascript som webbservern sedan exekverar. Funktionalitet som utnyttjas är oftast sökfält och formulär, där istället för det förväntade informationen skickas det in programkod. Problemet uppstår i webbserverns tolkning av angriparens inskickade data, där JavaScript-koden blir exekverat istället för visats som klartext. [6]

En webbkodinjektionsattack är möjlig när webbapplikationen tar emot opålitlig data från användaren och med hjälp av den dynamiskt genererar innehåll som visas på sidan. Under denna process granskas inte datan för innehåll som är exekverbart för en webbläsare. Exempel på denna typ av innehåll är till exempel JavaScript-kod, HTML-taggar, eller liknande innehåll som tolkas av webbläsaren som exekverbart kod. Detta genererade innehåll kan även lagras på webbservern, vilket leder till att andra webbläsare som laddar sidan, kommer automatiskt att exekvera den skadliga koden som

är injicerad i innehållet. Webbkodinjektionsattacker åtskiljs från varandra oftast genom att se på den skadliga datans beständighet.

När man diskuterar om webbkodinjektion brukar man ofta specificera om det är frågan om reflekterad, lagrad eller DOM-baserad. Webbkodinjektionen kategoriseras som reflekterad om användarens indata sparas i en variabel och visas på webbsidan. Indata lagras inte på serversidan, utan endast renderas på klientsidan. [7]



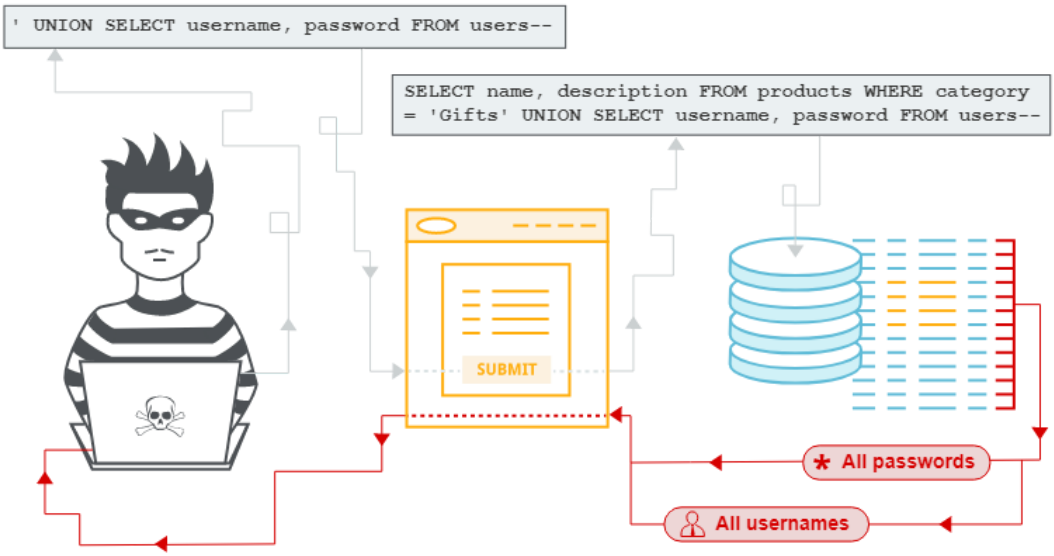
Figur 2. Ett visualisering om en lagrad webbkodinjektionsattack [11]

I figur 2 visualiseras en lagrad webbkodinjektionsattack. Funktionaliteten som uttnyttjas är en kommenteringsfält, genom vilken användare kan lämna kommentarer på sidan. Kommentarerna sparas i webbapplikationens databas därifrån de serveras till andra användare som besöker sidan. Angriparen injicerar sitt eget kod till kommentaren, vilken sparas i databasen. Eftersom webbapplikationen inte processar datan vidare utan serverar den senaste kommentaren, kommer andra användarnas webbläsare att tolka kommentaren som exekverbar kod. Eftersom angriparen har fått sitt kod över till klientsidan kan hen extrahera personlig information från användarnas webbläsare, till exempel sessionskakor.

TODO: DOM-based XSS

3.1.2 SQL-injektion

SQL-injektion (SQLi) är en kodinjektionsmetod som används för att angripa webbapplikationer, genom att rikta attacken mot datalagret. Attacken liknar webbkodinjektion men istället för att injicera HTML och JavaScript, injicerar angriparen SQL-frågor, vilka kan returnera, modifiera eller radera data från tabeller i webbapplikationens databas. [5] En SQL-injektion möjliggörs av webbapplikationens direkta och ofiltrerade användning av användarens indata. Detta gör det möjligt för användaren att skicka in fullständiga SQL-frågor som exekveras av webbservern [9]



Figur 3. En visualisering av en SQL-injektion [12]

Figur 3 visualiserar hur en SQL-injektionsattack fungerar i praktiken. I figuren ovan utnyttjar angriparen ett sökformulär som använder indatan direkt i SQL-frågan. Angriparen kan då injicera sin egen SQL-fråga, genom att avbryta webbserverns fråga med citattecken, och börja en ny fråga för databasens användarnamn och lösenord.

3.1.3 Mitigering

Med injektionsattacker är filtrering den främsta verktyget för mitigering. Filtrering innebär att skrubba data från skadliga och oönskade delar och på så sätt neutralisera den. Filtrering kan göras när data först tas emot av webbservern, eller efter att datan har bearbetats och innan de skickas iväg till webbläsaren. Dessa distinktioner kallas för input och output filtrering. [10]

TODO

3.2 Fel i åtkomstkontroll

3.2.1 CSRF

3.2.2 Forced Browsing

3.2.3 Mitigering

3.3 Kryptografiska fel

3.3.1 Man-i-mitten-attack

3.3.2 Password cracking

3.3.3 Mitigering

4. Sammanfattning och diskussion

TODO

Källförteckning

[1] C. Timberg, “Net of Insecurity: A Flaw in the Design” *Washington Post,* Maj 30, 2015 [Online] Available: https://www.washingtonpost.com/sf/business/2015/05/30/net-of-insecurity-part-1/ *Hämtad 27.02.2023*

[2] Stack Overflow “Stack Overflow Annual Developer Survey” https://insights.stackoverflow.com/survey

[3] OWASP Foundation “Who is the OWASP Foundation?” https://owasp.org/

[4] OWASP Foundation “OWASP Top Ten” https://owasp.org/www-project-top-ten/

[5] F. Q. Kareem, “SQL Injection Attacks Prevention System Technology: Review”, *AJRCoS*, vol. 10, no. 3, pp. 13–32, Jul. 2021.

[6] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides, “Cross-site scripting (XSS) attacks and mitigation: A survey,” *Computer Networks*, vol. 166, p. 106960, 2020.

[7] CWE, “Cwe-79: improper neutralization of input during web page generation (‘cross-site scripting’)”, https://cwe.mitre.org/data/definitions/79.html *Hämtad 11.03.2023*

[8] OWASP Foundation, “Types of XSS”, https://owasp.org/www-community/Types_of_Cross-Site_Scripting

Hämtad 11.03.2023

[9] CWE, “CWE-89: Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)”,
https://cwe.mitre.org/data/definitions/89.html
Hämtad 26.03.2023

[10] A. Rager, J. Grossman, P. D. Petkov, R. Hansen, S. Fogie. “XSS Attacks: Cross Site Scripting Exploits and Defense” (2011)

[11] Okta, “XSS Vulnerability 101: Identify and Stop Cross-Site Scripting”,
https://www.okta.com/identity-101/xss-vulnerability/
Hämtad 28.03.2023

[12] PortSwigger, “SQL injection”,
https://portswigger.net/web-security/sql-injection
Hämtad 28.03.2023