Welke beveiligingsproblemen komen voor bij REST-API's en hun authenticatiesystemen, en hoe kunnen deze worden voorkomen?

2025 – 2026
GDC – 3De Graad

Ian-Chains Baute – 6ICWE
Generieke Doorstroom Competentie
Informaticawetenschappen

GDC Mentor: J. Gillis

Woord vooraf

Digitale systemen en softwarepakketten worden steeds groter, uitgebreider en complexer maar de veiligheid en beveiliging mag niet vergeten worden, deze beveiliging wordt ook steeds uitdagender en complexer samen met de groei van de software. Uitgebreide software systemen maken vaak gebruik van API's om data uit te wisselen tussen de gebruiker of andere software systemen en daarbij is de REST-API de meest voorkomende en gebruikte.

Ik heb zelf al meerdere keren REST-API's gebruikt in projecten maar nooit echt stil gestaan bij de beveiliging van de API voor mijn projecten. Daarom wil ik in dit onderzoek dan ook de nadruk leggen op de beveiliging van een REST-API systeem om bij te leren over welke beveiligingsproblemen er vaak aanwezig zijn en hoe ze voorkomen kunnen worden.

Dit onderzoek over de beveiliging van REST-API's heb ik uitgevoerd als voorbereiding op een vervolg project waarbij ik deze informatie over de beveiliging kan gebruiken om op een veilige manier een API systeem op te zetten bij mijn vervolg projecten.

Graag wil ik een aantal mensen vermelden en banken voor hun hulp en assistentie bij dit GDC onderzoek. Ik wil in het bijzonder mijn mentor meneer Gillis bedanken voor de steun en tips tijdens dit onderzoek. Daarnaast wil ik graag ook

Inhoudsopgave

Inleiding

In dit werk wordt er een technisch onderzoek gedaan naar API[1] systemen zowel theoretisch als praktisch, om de algemene beveiliging van API systemen beter te begrijpen en hierop te anticiperen. Dit onderzoek gaat dieper in op 3 veelvoorkomende beveiligingsproblemen bij REST-API's[2]. De 3 beveiligingsproblemen die onderzocht worden zijn: Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) en improper input validation. Het 1ste deel van dit werk is het theoretisch onderzoek naar de beveiligingsproblemen door middel van een literatuur studie om beter te kunnen begrijpen wat deze beveiligingsproblemen nu precies betekenen en inhouden.

Het 2de deel is het praktisch productie gedeelte waarbij er 2 verschillende API systemen zullen opgezet worden. Het 1ste systeem zal voorbeelden met één of meerdere van de onderzochte beveiligingsproblemen bevatten. Het 2de systeem zal dezelfde voorbeelden bevat maar dan met de extra stappen en oplossingen om deze beveiligingsproblemen te voorkomen. Door deze 2 systemen met elkaar te vergelijken, kan er praktisch onderzocht worden welke stappen er kunnen ondernomen worden om deze beveiligingsproblemen te voorkomen.

Het praktische gedeelte wordt volledig gedocumenteerd via een Github repository, de link naar de repository is opgenomen in de bijlagen van dit werk. Bij het praktische gedeelte is er gebruik gemaakt van Python en FastAPI om een API systeem op te zetten, meer informatie over de gebruikte technologiestack kan terug gevonden worden in de repository.

[1] Informatie over wat een API is en hoe het technisch werkt kan gevonden worden in dit artikel van Geeks For Geeks: GeeksforGeeks. (2025, December 15). *What is an API (Application Programming Interface)*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing/what-is-an-api/

[2] Informatie over wat een REST-API is en hoe het technisch werkt kan gevonden worden in dit artikel van Postman: Team, P. (2025, December 16). *What is a REST API? Examples, uses, and challenges*. Postman Blog. https://blog.postman.com/rest-api-examples/

Afkortingen

GDC	Generieke Doorstroom Competentie
i.p.v.	in plaats van
i.v.m.	in verband met
API	Application Programming Interface
REST	Representational State Transfer
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
SOP	Same Origin Policy
CORS	Cross-Origin Resource Sharing
DOM	Document Object Model
DB	Database
Py	Python
JS	JavaScript
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
URL	Uniform Resource Locator
webapp	webapplicatie
----authN	Authentication (Authenticatie)
-----authZ	Authorization (Autorisatie)
UID	Unique Identifier

1 XSS - Cross-Site Scripting

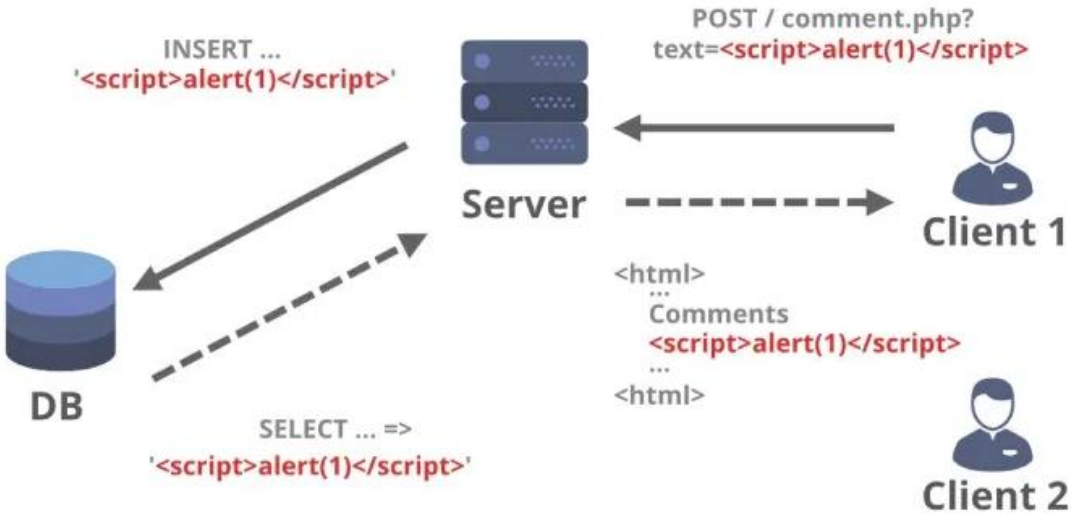
1.1 Wat is XSS?

XSS is een kwetsbaarheid in een webapp dat iemand toelaat om JavaScript te injecteren in de webapp en die te laten uitvoeren in de browser van een andere gebruiker in naam van de vertrouwde maar kwetsbare webapplicatie. Daarbij draait dus de JavaScript code in de browser van het slachtoffer, wat een gewone gebruiker of een beheerder kan zijn, en de browser denkt daarbij dat de code afkomstig is van de vertrouwde webapp. Hierdoor kan de code gevoelige zaken uitvoeren zoals:

- (cookies met) sessie gegevens stelen
- zich voordoen als een andere gebruiker
- formulieren manipuleren
- content van de website aanpassen
- ...

Figuur 1

Visualisatie (Stored) XSS



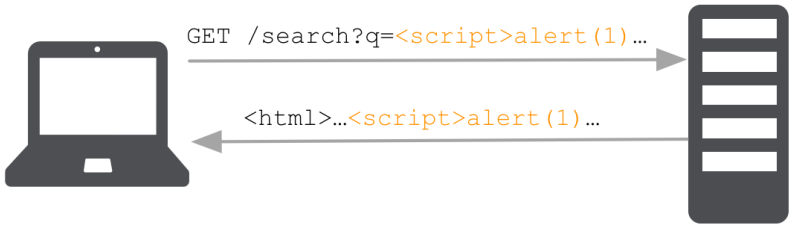
GeeksforGeeks (2025)

Figuur 1 is een visuele weergave van een XSS kwetsbaarheid die wordt getest, waarbij client 1 de aanvaller is en client 2 het slachtoffer is. Er wordt in de figuur een veel gebruikte en simpele JavaScript functie genaamd alert gebruikt om de kwetsbaarheid van XSS aan te tonen. Wanneer het slachtoffer een pop-up in zijn/haar beeld krijgt, dan is er effectief sprake van een XSS kwetsbaarheid. XSS Kan opgedeeld worden in 3 verschillende categorieën: reflected, stored of DOM-based XSS. Deze figuur is een voorbeeld van stored XSS want de code wordt opgeslagen in de database.

1.1.1 Reflected XSS

Figuur 2

Visualisatie reflected XSS bij zoek functie



Hepper (2017)

Bij reflected XSS wordt de code niet opgeslagen maar meteen gereflecteerd en teruggestuurd in de HTTP-response naar de browser van het slachtoffer. Een reflected XSS kwetsbaarheid komt vaak voor bij zoek functies, net zoals bij figuur 2. Er wordt dan JavaScript code meegegeven als zoekterm i.p.v. een realistische zoekterm en dit is exact wat er in figuur 2 gebeurt. De server in figuur 2 verwerkt het verzoek en geeft de opgezochte term terug aan de gebruiker. Want bij de meeste zoek functies wordt de term die je hebt opgezocht nog eens weergegeven aan de gebruiker, daardoor bevind de JavaScript zich in de HTTP-response. Doordat de code zich in de HTTP-response bevind zal deze uitgevoerd worden door de browser van het slachtoffer en kan dit schade veroorzaken.

Een reflected XSS kwetsbaarheid stelt een aanvaller in staat om een kwaadaardige URL te maken met daarin JavaScript en deze door te sturen naar een slachtoffer. Het slachtoffer krijgt deze link en ziet dat deze naar de officiële maar kwetsbare webapp verwijst. Het slachtoffer klikt op de link en de browser verstuurt een HTTP-request naar de server. De server verwerkt de input met de JavaScript en antwoord met een HTTP-response waarin zich de JavaScript bevind. De input en dus de code komt ongewijzigd terug in de pagina van het slachtoffer en de browser voert de JavaScript code uit. Dit is gevaarlijk omdat de gebruiker er vanuit gaat dat alles in orde en veilig is omdat de link naar de officiële webapp gaat maar dat is niet zo door de kwetsbaarheid.

1.1.2 Stored XSS

Bij stored XSS wordt de JavaScript code opgeslagen op de server, vaak in een database. De code wordt dan automatisch uitgevoegd bij elke gebruiker die de opgeslagen content bekijkt of opvraagt. Deze kwetsbaarheid komt vaak voor bij gebruikers gegenereerde content zoals:

- Reacties, opmerkingen & reviews
- Forum & community berichten
- (contact)formulieren & rich text editors
- ...

Figuur 1 uit onderdeel 1.1 toont hoe het plaatsen van een opmerking kan misbruikt worden, als er een XSS kwetsbaarheid aanwezig is. De aanvaller plaats een opmerking met kwetsbare javascript i.p.v. een realistische opmerking. Deze opmerking en code worden opgeslagen in de database van de webapp. Andere gebruikers laden de pagina met opmerkingen en met de kwetsbare javascript, deze javascript wordt vervolgens uitgevoerd in de browser van het slachtoffer. Dit is gevaarlijk omdat de gebruiker zelf niks moet doen, het gebeurt gewoon.

1.1.3 DOM-based XSS

DOM-based XSS komt niet voor bij API's maar in de frontend van een webapp omdat het alleen plaats vindt in de browser van de gebruiker. Er is geen uitwisseling van kwaadaardige code tussen de client en de server. DOM verwijst naar de HTML structuur van een pagina, die dan gemanipuleerd wordt door XSS. DOM-based XSS komt voor doordat er een fout zit in de JavaScript van de frontend.

1.2 Hoe XSS voorkomen?

x

1.3 Praktisch

X

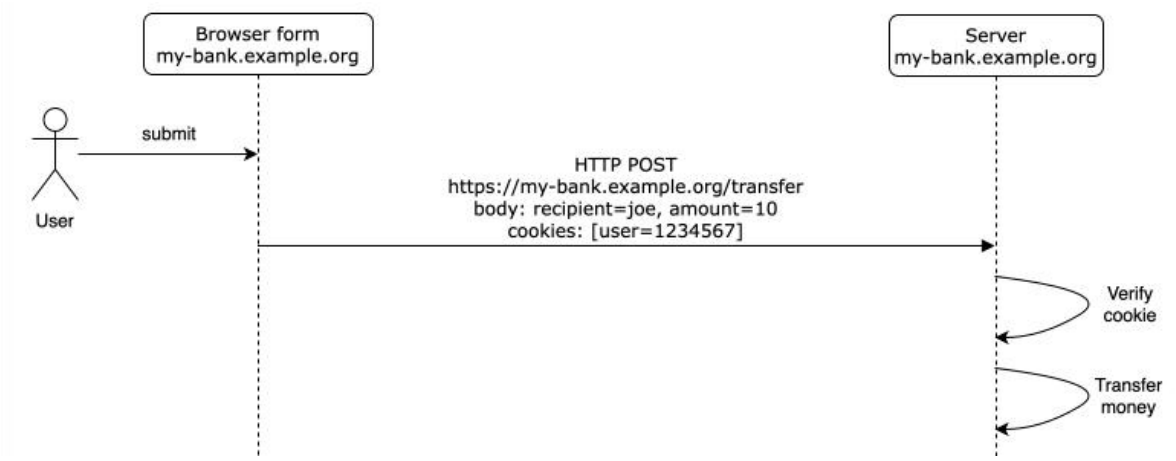
2 CSRF - Cross-Site Request Forgery

2.1 Wat is CSRF?

CSRF is een beveiligingsprobleem waarbij het mogelijk is om ongewenste HTTP requests te versturen vanaf een andere site (cross-site) zonder dat de gebruiker dit wilde en zonder een actie uit te voeren. Door deze requests kan er data worden gedeeld of aangepast worden vanaf een andere site, zonder dat de gebruiker dit wilde en kan een request een kwaadaardig effect hebben.

Figuur 3

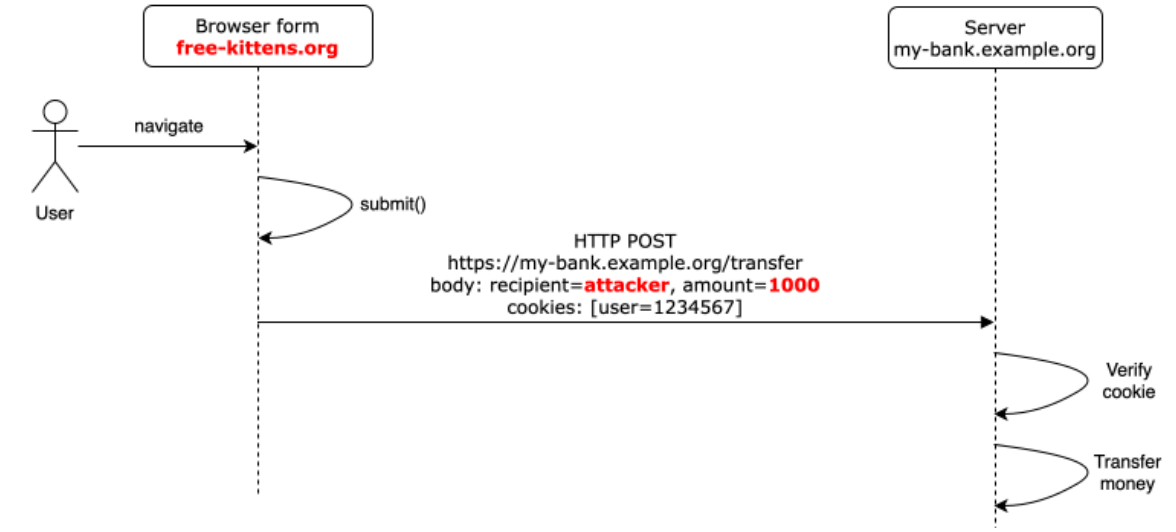
Voorbeeld HTTP POST request, geld overschrijven



Figuur 3 toont een POST request van een API waarbij de gebruiker zich op de webapp *my-bank.example.org* bevind en geld wil overschrijven. De gebruiker voert de actie uit dus wordt er een HTTP POST request verstuurd. De gebruiker is reeds ingelogd en heeft dus een user UID of authenticatie token in de cookies van zijn browser opgeslagen. Deze cookies worden automatisch door de browser altijd meegeven met een HTTP request. De server ontvangt deze request en controleert de user UID en/of de authenticatie token, als deze tokens geldig zijn dan wordt de request verwerkt en het geld overgeschreven.

Figuur 4

Voorbeeld CSRF bij POST request van figuur 3



Figuur 4 toont dezelfde POST request als figuur 3 maar de gebruiker bevindt zich deze keer op een kwaadaardige webapp. Deze webapp zal zelf zonder enige aanleiding of actie een HTTP POST request versturen in naam van de gebruiker om geld over te schrijven. De browser herkent de hostname van de request en linkt deze aan de cookies van de bank webapp en geeft deze cookies en authenticatie tokens automatisch mee in de request. De server ontvangt deze request en zal deze verwerken als de authenticatie tokens nog geldig zijn. Een CSRF-aanval maakt dus gebruik van het feit dat een gebruiker reeds ingelogd is en authenticatie tokens reeds opgeslagen zijn als cookie, die worden dan automatisch meegeven door de browser.

2.2 Hoe CSRF voorkomen?

2.2.1 SOP & CORS

Moderne browsers maken gebruik van de Same-Origin Policy (SOP) beveiligingsmaatregel om CSRF te voorkomen. SOP is een maatregel aan de clientside en zorgt ervoor dat er enkel gevoelige data tussen een frontend en een API kan uitgewisseld worden als deze same-origin[3] zijn. Als er niet wordt voldaan aan de SOP dan wordt de data van de API request geblokkeerd en kan de andere webapp deze data niet gebruiken/lezen. Zodat een andere (kwaadaardige) webapp die cross-origin[4] is, geen gevoelige data kan lezen van een API van een andere webapp.

Cross-Origin Resource Sharing (CORS) is een uitzondering op SOP, met CORS kan je een andere webapp toegang geven om data te gebruiken van een andere API als deze cross-origin zijn.

2.2.2 Server Side Origin Check

2.2.3 CSRF Tokens

2.3 Praktisch

x

3 Improper Input Validation

[3] Meerdere webapps of pagina's zijn **same-origin** aan elkaar als ze dezelfde protocol (HTTPS), hostname en poort hebben.
[4] **Cross-Origin** is het tegenovergestelde van same-origin. Wanneer 1 van de eigenschappen niet volledig overeenkomt, wordt dit aangeduid als cross-origin.

Besluit

- **Nog aanpassen en ook toevoegen van CSRF, dus controleren of de request wel echt is en niet zonder toestemming is gemaakt**

Het is duidelijk dat er één grote lijn en overeenkomst aanwezig is tussen al deze verschillende veelvoorkomende beveiligingsproblemen bij REST-API's. De directe input en gegevens van de gebruiker of client kan nooit vertrouwd worden en moet altijd eerst gecontroleerd en opgekuist worden voordat deze input en gegevens verwerkt worden. Zodat de input geen schadelijke payload of code bevat en geen schade kan aanbrengen aan het digitale systeem, de server, de databank of andere gebruikers. Dus er moet altijd eerst gepoetst worden voordat de input kan vertrouwd worden, eens dat is gebeurd kan de input verwerkt worden.

Deze input gegevens kunnen gepoetst worden op verschillende manieren namelijk:

Bijlagen

Voor het praktische gedeelte van dit onderzoek is er gebruik gemaakt van een Github repository om de bestanden, de wijzigingen en documentatie overzichtelijk te kunnen bijhouden. De link naar de Github repository: https://github.com/GO-atheneum-De-Tandem/GDC-Ian-2025-2026

Alle bijlagen werden samen gebundeld tot een zip bestand. Het zip bestand werd samen met dit werk ingediend. Je kan de bijlagen ook downloaden via volgende link:

Info & Onderwerp:	**Bestandsnaam:**
Referentielijst van dit onderzoek	25-26_GDC_Referentielijst_Ian_6ICWE.pdf

Referentielijst

De referentielijst is opgenomen in een apart bestand om dit onderzoek overzichtelijk te houden. Dit bestand kan je terug vinden in de meegeleverde bijlagen.

het controleren van spelling van dit werkstuk is er gebruik gemaakt van woordenlijst.org en spelling.nu.