Självständigt arbete på grundnivå

Independent degree project – first cycle

Datateknik
Computer engineering

Undersökning av webbsidors säkerhet vid användning av Facebook Login
- Vidareutveckling och analys av OAuthGuard

Alice Hedmark

Mittuniversitetet
MID SWEDEN UNIVERSITY

MITTUNIVERSITETET

DSV Östersund

Examinator	Dr. Felix Dobslaw	Felix.Dobslaw@miun.se
Handledare	Raja-Khurram Shahzad	Raja-Khurram.Shahzad@miun.se
Författare	Alice Hedmark	alhe1501@student.miun.se
Utbildningsprogram	Programvaruteknik, 180hp	
Kurs	DT133G, Självständigt Arbete	
Huvudområde	Datateknik	
Termin, år	VT, 2019	

Sammanfattning

Single Sign-On (SSO) är en autentiseringsprocess som tillåter en utvecklare att delegera autentiseringsansvaret till en dedikerad tjänst. OAuth 2.0 är ett auktoriseringsramverk som ofta står som grund för ett autentiseringslager som i sin tur möjliggör SSO. En identitetsleverantör är tjänsten som står för hantering av användaruppgifterna och autentiseringen, två vanliga identitetsleverantörer är Google och Facebook som i sin tur implementerar SSO med hjälp utav autentiseringslagren OpenID Connect respektive Facebooks egna autentiseringslager. Det har visat sig att många klienter som ska utnyttja SSO med OAuth 2.0 implementerar det fel så att säkerhetsbrister uppstår, studier har utförts med förslag till lösningar men många bristande implementationer fortsätter produceras och existera. Att skapa diverse verktyg för att främja säkerhet i dessa sammanhang är en metod där OAuthGuard utvecklats med visionen att även kunna skydda användaren, direkt från en webbläsare. OAuthGuard har även tidigare använts för att analysera säkerheten med Google SSO och visat att 50% av undersökta klienter har brister, men motsvarande studie eller verktyg saknas för Facebook SSO. Denna studie gjorde en motsvarande undersökning för Facebook SSO-klienter med en vidareutvecklad version av OAuthGuard och fann att de lider av brister med liknande trend som tidigare studies resultat mot Google-SSO-klienter, men att färre Facebook-SSO-klienter har brister i jämförelse. Vid vidareutvecklingen av OAuthGuard upptäcktes ett antal svårigheter och framtiden för denna typ av verktyg behöver vidare analyseras. Vidare analys behöver även göras för att bedöma om Facebook-SSO kan vara att föredra över Google-SSO ur säkerhetsperspektiv samt vidare utforskande av nya säkerhetsfrämjande metoder behöver utföras.

Nyckelord: Webbläsartillägg, OpenID Connect, OIDC, OAuth 2.0, Single Sign-On, SSO, Säkerhet, Facebook, Social login.

Abstract

Single Sign-On (SSO) is an authentication process that allows a developer to delegate the authentication responsibility to a dedicated service. OAuth 2.0 is an authorization framework that often serves as a base for authentication layers to be built upon that in turn allows for SSO. An identity provider is the service that is responsible for handling user credentials and the authentication, two common identity providers are Google and Facebook that implement SSO with the authentication layers OpenID Connect respectively Facebooks own authentication layer. It has been shown that many clients using OAuth 2.0 as base for SSO make faulty implementations leading to security issues, a number of studies has proposed solutions to these issues but faulty implementations are continually being made. To create various tools to promote security in these contexts is a method where OAuthGuard has been developed with the vision to also directly protect the common website user directly from the browser. OAuthGuard has been used in an earlier study to analyze the security of clients using Google SSO and discovered that 50% of the analyzed clients had flaws, no comparable study has been done for clients using Facebook SSO, which is the second largest third party log in variant. This study made a comparable investigation for Facebook SSO clients with a further developed version of OAuthGuard and found that these clients suffer from flaws with a similar trend as the previous study with Google-SSO clients, although fewer Facebook-SSO clients suffer from these flaws. When further developing OAuthGuard a dumber of difficulties was discovered and the future of these kind of tools needs to be investigated. Further analysis needs to be done to assess if Facebook-SSO should be recommended over Google-SSO from a security perspective and also further exploration of new methods to promote security needs to be done.

Keywords: Web browser extensions, OpenID Connect, OIDC, OAuth 2.0, Single Sign-On, SSO, Security, Facebook, Social login.

Innehållsförteckning

Tabellförteckning

Figurförteckning

Akronym

OIDC	OpenID Connect
SSO	Single Sign-On
CSRF	Cross Site Request Forgery
IdP	Identity Provider
RP	Relying Party
OP	OpenID Provider
SP	Service Provider
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
SDK	Software Development Kit
API	Application Programming Interface

1 Inledning

Det nämns ofta på medier om säkerhetsläckor och system som blivit hackade [1-6]. En av de saker som nämns är autentisering och säkerhetsbrister som leder till att identifieringsuppgifter kan synliggöras och därav illvilligt kunna utnyttjas av andra parter än avsedd användare [1, 3-5]. Den traditionella användarnamn-lösenord autentiseringen (enfaktorautentisering) har flertalet gånger visat sig problematisk, bland annat för att användare tenderar att använda samma lösenord för flertalet tjänster och bristande säkerhet har resulterat i obehörig tillgång till lösenordsdatabasers innehåll [7,8]. Det förutsägs att autentiseringmetoder kommer gå mer och mer från rena lösenordsbaserade metoder till andra metoder, däribland tvåfaktorautentisering och lösenordslösa autentiseringsmetoder för att öka säkerheten [9-11]. Men det kan vara svårt att implementera säkra autentiseringssystem, speciellt utan djup kunskap om området [12, 13] och det är en av anledningarna till att det i medier ofta förekommer rapporter om sårbara system.

Single Sign-On (SSO) är en autentiseringsprocess där flera tjänster delegerar autentiseringsansvar till en dedikerad autentiseringstjänst (identitetsleverantör). En användare kan då istället för att skapa ett nytt konto i varje system logga in och bekräfta sin identitet med exempelvis de mest populära identitetsleverantörerna Google [17] eller Facebook [18] (som 2015 enligt [13] stod för 51% respektive 26% av utförda tredjepartsautentiseringar) vilket gör det möjligt för användare att minimera antalet lösenord de behöver använda. OAuth 2.0 [14] är ett ramverk för auktorisering som bland annat kan användas som bas i SSO autentisering och det har under åren uppdagats brister i bland annat implementationer av klienter som använder OAuth 2.0 baserad SSO autentisering [12, 19, 20]. Ett flertal studier har försökt sammanfatta och/eller belysa om säkerhetsbrister som förekommer samt föreslå lösningar [12, 19, 20, 24]. Det visar sig i tidigare studier [9, 19, 20, 21, 22] att befintliga specifikationer och instruktioner inte ännu löser problemet med bristfälliga implementationer och att en stor mängd av studiernas undersökta tjänsteleverantörer har brister då problemen kvarstår trots uppmärksammande.

En annan strategi att hantera säkerhetsproblemet är att utveckla hjälpmedel och verktyg för att skydda användarna vilket är en av de metoder som rekommenderas i [10]. Ett begränsat antal verktyg för detta syfte har utvecklats, däribland OAuthGuard [19], WPSE (Web Protocol Security Enforcer) [11], SSOScan [21] och PrOfESSOS [9]. Av dessa verktyg är både WPSE och OAuthGuard tillägg till Google Chrome som kan användas av utvecklare. OAuthGuard är av dessa två det verktyg som skyddar mot flest typer av attacker, däribland CSRF (Cross Site Request Forgery) och diverse felhantering av tokens, och finns lättillgängligt på Chrome Web Store [27]. PrOfESSOS och SSOScan är tester/verktyg utvecklade främst för utvecklare att använda där SSOScan fokuserat på automatik för att scanna en stor mängd hemsidor [9, 21]. Det föreslås i [19] att vidare utveckling och analys av OAuthGuard verktyget behövs då det i nuläget endast är en prototyp som inte är i tillräckligt bra skick för att användas av vanliga användare, exempelvis genom att utöka så fler identitetsleverantörer stöds då endast Google stöds i nuläget. OAuthGuard [19] användes för att undersöka en mängd tjänsteleverantörer och hittade allvarliga säkerhetsbrister i 69 av 137 undersökta klienter som använde sig av Google som identitetsleverantör samt att de mer populära hemsidorna oftare hade säkerhetsbrister än

de mindre populära. Om OAuthGuard skulle anpassas för att även stödja Facebook som identitetsleverantör skulle OAuthGuard kunna behandla och skydda vid en betydligt större del av alla tredjepartsautentiseringar (över 76% av alla tredjepartsautentiseringar jämfört med tidigare 51% [13]). I en tidigare undersökning utförd 2013 på populära tjänsteleverantörer som stödjer Facebook som identitetsleverantör visade det sig att 345 av de 1660 undersökta med verktyget SSOScan hade säkerhetsbrister relaterat till autentisering/auktorisering [21]. Det har visat sig finnas ett intresse för att hitta säkerhetsfrämjande lösningar i form utav diverse verktyg, men de få verktyg som finns fokuserar endast på en identitetsleverantör och behandlar olika säkerhetsbrister. Om det visar sig att en utökning av antalet identitetsleverantörer OAuthGuard kan hantera leder till att en större mängd säkerhetsbrister kan upptäckas kan detta motivera ytterligare utökning av OAuthGuard eller liknande verktyg för att få en mer mogen produkt som lätt kan användas av allmänheten för att ge skydd.

1.1 Problemformulering

Tidigare studie utförd med verktyget OAuthGuard [19] upptäckte att en övervägande del av de undersökta Google-SSO-klienterna hade säkerhetsbrister, något som kan utsätta en användares resurser och identitet för fara. Ingen motsvarande studie eller verktyg existerar till författarens vetskap för Facebook-SSO-klienter. Denna studie ska vidareutveckla OAuthGuard så även Facebook som identitetsleverantör stöds, dels för att ge ett bredare skydd åt användare, men även för att göra en motsvarande studie för Facebook-SSO-klienter. För att utföra studien används den utökade funktionaliteten i OAuthGuard på flertalet webbsidor med Facebook som autentiseringsalternativ och upptäckta brister dokumenterades för att samla information inför utvärdering av säkerhetsläget vid autentisering med Facebook-SSO.

1.2 Syfte

Denna studie är ämnad att vidareutveckla OAuthGuard till att även stödja Facebook-SSO så att verktyget kan ge ett bredare skydd, samt att med hjälp utav det vidareutvecklade verktyget OAuthGuard undersöka hur säkerhetsläget vid SSO-autentisering ser ut för hemsidor som stödjer Facebook-SSO. Detta för att i huvudsak kunna utvärdera om tjänsteleverantörer som använt Facebook som identitetsleverantör följer samma mönster med säkerhetsbrister vid SSO-autentisering som de tjänsteleverantörer som använt Google enligt tidigare studie med OAuthGuard [19]. Beroende på vad denna utvärdering visar kan det motivera undersökning av ytterligare säkerhetsfrämjande metoder. Eventuellt att någon av de undersökta SSO-alternativen, Google och Facebook, kan vara att föredra ur säkerhetsperspektiv. Denna utvärdering görs med hjälp utav att svara på följande forskningsfrågor, som används som grund för jämförelserna:

- Hur ser säkerhetsläget ut bland populära webbsidor som stödjer Facebook som identitetsleverantör vid användning av OAuthGuard och hur skiljer sig detta mot tidigare studie [19] utförd mot Google som identitetsleverantör?
    - Hur skiljer sig förhållandet mellan antal som har säkerhetsbrister?
    - Hur skiljer sig förhållandet mellan popularitet och antal som använder Facebook SSO?

- Hur skiljer sig förhållandet mellan popularitet och antalet säkerhetsbrister?
- Hur skiljer sig frekvensen av de olika typer av säkerhetsbrister som undersöks med OAuthGuard?

1.3 Omfattning

Utökande av OAuthGuards tillägg till Facebook som identitetsleverantör kommer göras samt bidra med information över hur tillägget kan användas, detta för att utöka OAuthGuards skyddspotential samt för att bli lättare att förstå och därmed potentiellt användas för vanliga användare. Detta även för att underlätta för utvecklare som vill vidareutveckla verktyget. Utökningen av OAuthGuard kommer endast fokusera på Authorization Code flow med response_type code. De säkerhetsbrister som kommer stödjas av vidareutvecklade verktyget OAuthGuard mot Facebok-SSO-klienter samt jämföras mot tidigare studie är CSRF, Third party token leakage, Referer token leakage samt Unsafe token transfer/osäker anslutning.

En empirisk undersökning av säkerhetsläget av Facebook SSO bland de 500 populäraste hemsidor ska utföras. Denna undersökning utförs för att bidra med statistik över nuvarande säkerhetsläget men även för att bedöma hur användbar utökningen av tillägget OAuthGuard var genom att se hur ofta brister förekommer som OAuthGuard kan upptäcka och eventuellt även skydda mot.

Studien kommer i huvudsak fokusera på tjänsteleverantörer och dess brister, detta beror i huvudsak på att tillägget från grunden är utvecklad för just detta. Testande av säkerhetsbrister som är relaterad till identitetsleverantören kommer vara begränsad då ingen implementation av identitetsleverantör/tredjepartskod kommer göras. En testklient kommer utvecklas för att testa så att tillägget upptäcker brister som den ska. Endast brister som inte kräver kontroll över identitetsleverantör eller som kräver flera tjänsteleverantörer/tredjepartskod kommer tas hänsyn till för att inte utvecklingen av denna testklient ska ta för lång tid.

1.4 Översikt

Kapitel 2 beskriver hur området ser ut idag, under 2.1 beskrivs vanlig terminologi och dess innebörd, delkapitel 2.2 – 2.6 redovisar bakgrundsinformation om OAuth 2.0, SSO, OAuthGuard och webbläsartillägg och under 2.7 listas relaterade arbeten och vad de handlar om. Kapitel 3 beskriver vilken metod som använts för att nå resultatet, under 3.1 beskrivs det hur den framställning av den teoretiska delen av arbetet utförts, vidare under 3.2 beskrivs vilka verktyg som använts (3.2.1) och vilka krav (3.2.2, 3.2.3) som ställts. Kapitel 4 beskriver hur konstruktionen (utökning av OAuthGuard samt implementation av testklient) utförts i detalj, både vad gäller design och testning. Kapitel 5 listar resultat av konstruktion och undersökning. Kapitel 6 innehåller en diskussion av arbetet där bland annat svar på forskningsfrågorna listas. Kapitel 7 redovisar slutsatserna som framställts från arbetet.

2 Bakgrund

Kapitlet beskriver hur området ser ut idag, under 2.1 beskrivs vanlig terminologi och dess innebörd. Rubrik 2.2 – 2.6 presenterar information om OAuth 2.0, SSO, OAuthGuard och de säkerhetshot som OAuthGuard behandlar samt information om webbläsartillägg i allmänhet. Under 2.7 listas relaterade arbeten och hur denna studie förhåller sig till dem.

2.1 Terminologi

Integritet: Integritet innebär hur pålitlig en resurs är (ett mått på resursens ”äkthet”). Om en angripare kan modifiera en resurs utan tillgång förstörs resursens integritet [15].

Tillgänglighet: Tillgänglighet refererar till huruvida det går att komma åt en resurs. Om en angripare på något sätt gör det svårt eller omöjligt för avsedd användare att komma åt sin resurs är resursens tillgänglighet försämrad [15].

Konfidentialitet: En resurs konfidentialitet går ut på att begränsa åtkomst till resursen så endast behöriga får tillgång till resursen. Om en angripare får tillgång till en icke-publik resurs som denne inte ska ha tillgång till är denna resurs konfidentialitet bristande [15].

Nonce: Ett ”nonce” är ett godtyckligt, randomiserat tal som används endast en gång i kryptografisk kommunikation och används ofta i autentiseringsprotokoll för att säkerställa att delar av kommunikationen (meddelanden) inte kan återanvändas[1].

Autentisering: Autentisering går ut på bekräfta något, exempelvis sin identitet [2]. Det finns tre faktorer en användare kan använda för att bekräfta sin identitet:

1. Kunskap, något användaren vet – Lösenord
2. Ägarskap, något användaren har – Legitimation, mobiltelefon
3. Biologiskt, något användaren är eller gör/producerar – fingeravtryck, DNA

Auktorisering: Auktorisering innebär att delegera åtkomst till resurser baserat på identitet [3].

Cookie: Begrepp som används för en HTTP cookie och är en liten mängd data som skickas från en server till en webbläsare som sedan kan lagra den och använda den eller skicka vidare den [4]. Cookies används i huvudsak för att hålla koll på information som annars går förlorat vid ett nytt HTTP anrop.

1 https://en.wikipedia.org/wiki/Cryptographic_nonce

2 https://en.wikipedia.org/wiki/Authentication

3 https://www.owasp.org/index.php/Category:Access_Control

4 https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

Session: En session är en temporär lagring av information mellan parter som kommunicerar som vid något tillfälle initieras och vid något senare tillfälle kommer förstöras [5].

Referer header: Ett header-fält i ett HTTP-meddelande som innehåller adressen till hemsidan som förfrågat den webbsida som är aktuell [6].

Endpoints: I sammanhanget OAuth2 är en endpoint en URL som klienten kan skicka förfrågan till efter tokens/codes [7].

Backend/Frontend: Begrepp som beskriver lager i en mjukvaruarkitektur som är baserad på två lager [8]. Frontend beskriver den delen som användaren agerar med (grafik/gränssnittet) medan backend beskriver server-delen (den huvudsakliga bearbetningen och manipulationen av information).

SSO: Single Sign-On är en autentiseringsprocess som gör det möjligt för en användare att logga in till ett flertal tjänster med samma identitet och lösenord/autentiseringmetod [9].

OAuth 2.0: OAuth2 är ett auktoriseringsramverk som gör det möjligt för tjänster att fråga om åtkomst till begränsad information och resurser från andra tjänster [16]. OAuth2 används i många SSO lösningar och kan ses som auktoriseringsdelen av SSO.

OIDC: OpenID Connect är ett identitetslager/autentiseringslager som är byggt på OAuth2 som gör det möjligt för en tjänst att bekräfta en identitet på liknande sätt som OAuth2 frågar om åtkomst till resurser [10]. OIDC tillsammans med andra OAuth2 baserade autentiseringslager används i samband med SSO lösningar och kan ses som autentiseingsdelen av SSO.

IdP/OP: En Identity provider eller OpenID provider tillhandahåller identiteter och autentiseringstjänst. Identity provider är då den mer generella termen för dessa tjänster medan OpenID provider är en IdP för just tjänster som stödjer OIDC.

RP/SP: *Ett Relying part eller En Service provider är en tjänsteleverantör som kan delegera autentiseringsansvaret till en utomstående tjänst (IdP/OP). Denna tjänsteleverantör kan även vilja komma åt en annan tjänsts resurser via något API.*

2.2 OAuth 2.0

OAuth 2.0 är ett auktoriseringsramverk [14] (även tidigare benämnt som protokoll) som gör det möjligt för tredjepartsapplikationer att genom en auktoriseringsprocess/överenskommelse få begränsad tillgång till HTTP tjänster.

5 https://en.wikipedia.org/wiki/Session_(computer_science)

6 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer

7 https://docs.apigee.com/api-platform/security/oauth/configuring-oauth-endpoints-and-policies

8 https://sv.wikipedia.org/wiki/Front-end_och_back-end

9 https://en.wikipedia.org/wiki/Single_sign-on

10 https://openid.net/connect/

Innan en klient kan använda sig utav OAuth 2.0 måste klienten registreras hos tjänsten som ska användas. Vid denna registrering fastställs information om en redirect_uri, som visar vart användaren ska dirigeras om med eventuella codes/tokens efter användaren auktoriserat sig hos tjänsten, klienten ges även ett client_id och en client_secret som används för att identifiera tjänsten. Denna information är kritisk för säkerheten vid auktorisering/autentisering med OAuth 2.0 och måste hanteras på ett korrekt sätt.

OAuth 2.0 ramverken bygger på fyra roller, **resursägaren** (slutanvändaren), **klienten** (applikationen/RP/SP), **resursservern**(API) och **auktoriseringsservern** (API) [16]. Slutanvändaren är den roll som kan ge rättigheter åt en applikation för att komma åt ett konto/information som slutanvändaren har tillgång till. Applikationen frågar slutanvändaren om en begränsad mängd information (exempelvis skriv/läsrättigheter på utvald information) som slutanvändaren sedan får välja att bekräfta eller neka. Resursservern tillhandahåller de skyddade resurserna som skyddas med ett användarkonto. Auktoriseringsservern verifierar identiteten av användaren och tilldelar en access_token till applikationen som applikationen sedan kan använda för att få tillgång till de resurser som användaren tillåtit applikationen att komma åt. Auktoriseringsservern och resursservern kan ses som ett enda API då de ofta används tillsammans och ur tjänsteleverantörens/applikationens implementationsaspekt är densamma.

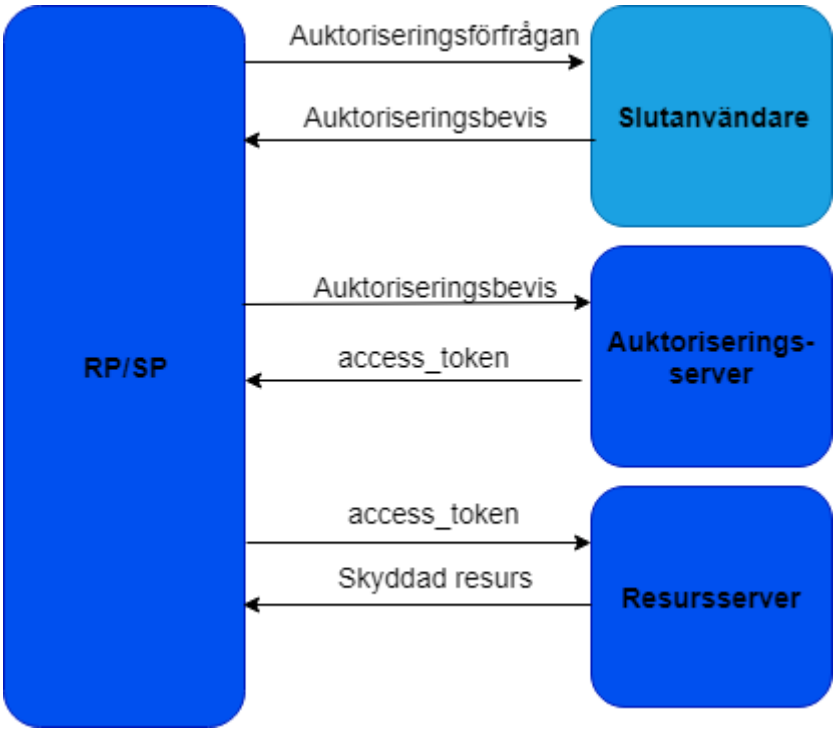


Bild 1: OAuth 2 protokollöversikt

Auktoriseringsflödet mellan rollerna sker på en översiktlig nivå i sex steg (se Bild 1) [16].

1. Applikationen skickar en auktoriseringsförfrågan för att få tillgång till resurser till användaren
2. Användaren auktoriserar förfrågan och applikationen får en auktoriseringsbekräftelse/ett auktoriseringsbevis
3. Applikationen skickar en förfrågan om en access_token till auktoriseringsservern och skickar med auktoriseringsbeviset samt sin egen identitet (client_id, client_secret).
4. Om applikationens identitet kan autentiseras och auktoriseringsbeviset är äkta får applikationen en access_token och auktoriseringen är komplett.
5. Applikationen kan nu med hjälp utav denna access_token begära resurser från resursservern mot uppvisande av token.
6. Om den token som uppvisats är giltig skickar resursservern resurser tillbaka till applikationen.

2.2.1 Flöden, förfrågningar och responser

I Bild 1 visas en generalisering av ett grant flow/code flow. OAuth 2.0 har fyra huvudsakliga grant types [16]:

1. Authorization Code: används hos applikationer som sköter auktorisering via server (backend)
2. Implicit: används i mobila applikationer eller webbapplikationer som endast körs på användarens enhet/system
3. Resource Owner Password Credentials: används för pålitliga applikationer, exempelvis sådana som tjänsten själv äger
4. Client credentials: används för åtkomst med applikations API

Dessa grant flows skiljer sig i vilka parametrar som finns i respons och förfrågan men även i hur många förfrågningar som görs och mellan vilka roller. Tabellen nedan visar dessa flöden och motsvarande response_type eller grant_type[11][16].

Flow	**Response type**	**Grant type**
Resource owner password credentials	-	password
Client credentials	-	client_credentials
Authorization code flow	code	-
Implicit flow	token	-
Device flow token request*	-	devide_code
Refreshing access tokens*	-	refresh_token
Tabell 1: Visar vilka flöden som har vilken response_type/grant_type		

*räknas ej in till de huvudsakliga grant flows definierade i RFC6749 [14].

11 https://oauth.net/2/grant-types/

Response modes:
Nedan beskrivs tre olika sätt att hantera responsen på[12].

Query: Auktoriseringsresponsparametrarna är inkodade i parameterdelen av redirect_uri vid omdirigering till klienten. (Standard för Response-type: code)

Fragment: Auktoriseringsresponsparametrarna är inkodade i fragmentet som läggs itll redirect_uri vid omdirigering till klienten. (Standard för Response-type: token)

Form Post: Auktoriseringsresponsparametrarna är inkodade i ett HTML formulär som värden som sedan automatiskt skickas till webbläsaren (överförs med HTTP POST som metod till klienten).

2.2.2 Tokens och codes

En mängd olika tokens (ett objekt som representerar en rätt till något [13]) och codes används vid auktorisering och autentisering. I OAuth 2.0 och OAuth 2.0 baserade autentiseringslager används olika typer av JSON Web Tokens (JWT)[14]. Nedan listas några vanliga tokens och koder:

- Code: en kod som kan bytas ut mot en access_token
- Id token: en token som innehåller information om en identitet.
- Access_token: en token som används för att få tillgång till resurser.
- Client_id: identifiering av en klient
- Client_secret: en nyckel till en klient, kombineras ibland med client_id för att bekräfta identitet

2.3 Autentiseringslager på OAuth 2.0

2.3.1 Single Sign-On

Single Sign-On (SSO) är en autentiseringsprocess där flera tjänster delegerar autentiseringsansvar till en dedikerad autentiseringstjänst (identitetsleverantör) [15]. En användare kan då istället för att skapa ett nytt konto i varje system logga in och bekräfta sin identitet med exempelvis Google [17] eller Facebook [18]. Att använda sig av en gemensam dedikerad inloggningstjänst i stället för att varje system har en egen autentiseringslösning minimerar antalet lösenord en användare behöver komma ihåg, vilket kan leda till att färre lösenord återanvänds. Färre lösenord att komma ihåg underlättar också för användaren att använda längre och mer komplexa lösenord. SSO-lösningar innebär också att ansvaret för säkerheten vad gäller lagring och hantering av identitetsuppgifter delegeras från tjänsteleverantören till den dedikerade autentiseringstjänsten.

12 https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html#ResponseModes
13 https://en.wikipedia.org/wiki/Token
14 https://tools.ietf.org/html/rfc7519
15 https://en.wikipedia.org/wiki/Single_sign-on

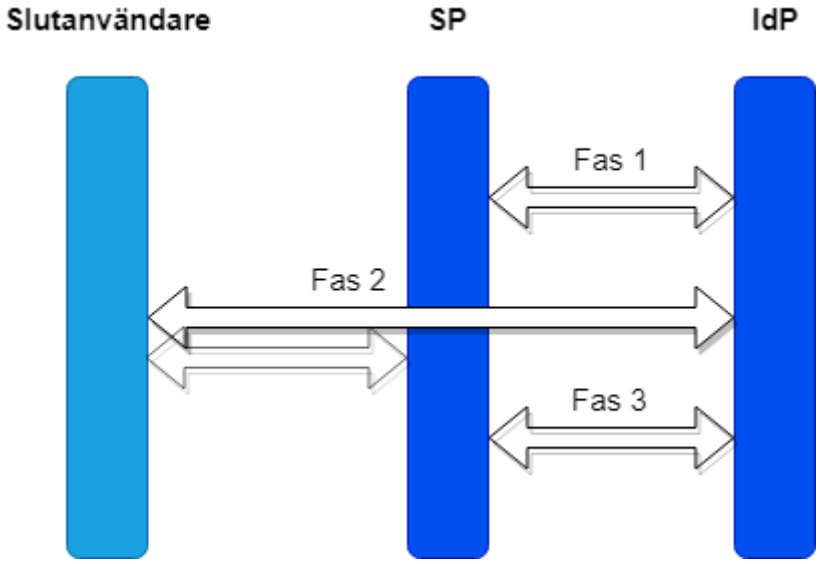


Bild 2: SSO faser

För att använda Single Sign-On autentisering existerar tre faser som används [9]. Se bild 2.

Fas 1 – Etablering av förtroende:

Det finns två varianter av förtroendeetablering, en manuell version när administratören till applikationen specifikt väljer vilka IdP:er som skall användas och manuellt registrerar sig på dessa, den andra versionen är dynamisk och tillåter att en *användare* väljer vilken IdP som ska användas (oavsett om denna IdP tidigare registrerats manuellt eller inte). Denna dynamiska process arbetar övergripande i tre steg:

1. Användaren bidrar med sin email (identifierare)
2. SP extraherar domänen från denna email för att identifiera en IdP
3. SP kan dynamiskt registrera sig på denna IdP genom att skicka en POST förfrågan till en specifik URL, SP får efter lyckad förfrågan/registrering bland annat en client_id och client_secret.

Fas 2 – Skapande av Token:

SP vidaredirigerar (vanligtvis) användaren till vald IdP. Användaren loggar in hos denna IdP som sedan genererar en token som skickas tillbaka till SP.

Fas 3 – Inlämnande av Token:

SP tar emot den token som skickats från IdP och som är tänkt att användas för att identifiera/autentisera användaren. Denna process är känslig och kräver att en mängd information i denna token verifieras. Denna token innehåller information om en identitet; användaren, en token innehåller även information om vilken SP som är tänkt att få tillgång till denna token (en URL eller ett ID på SP), en token kan även innehålla information om hur gammal en token är samt om den är tänkt att återanvändas eller inte (timestamps/nonces) och slutligen kan en token innehålla information om en signatur så en token kan vara signerad. All denna information kräver validering för att undvika säkerhetsbrister.

Claims:
Extra information kan läggas till olika tokens. Denna information kan variera beroende på vilken typ utav token det är och vilken typ utav scope som använts vid en förfrågan. När en SP får en ID Token kan följande claims[16] [17] [18] följa med (som följer OIDC standarden):

- ”iss” - URL, vem som skapat token
- ”sub” - identifiering av användaren (unik)
- ”aud” - vem token är skapad för, client_id
- ”exp” - när denna tokens livslängd tar slut
- ”iat” - när denna token skapades

I andra standarder kan denna information finnas tillgänglig under andra omständigheter där namn som identifierar dessa kan variera. Denna information finns tillgänglig delvis för att kunna verifiera så att rätt information kommit fram till rätt mottagare och sänts från rätt ställe.

Scopes:

Scopes [14][19] är identifierare i form utav strängar som beskriver vad som begärs tillgång till. Dessa scopes skickas i normala fall med då en auktoriseringsförfrågan utförs för att en access_token ska kunna ha de begränsningar som dessa scope anger. Exempel på inbyggda scopes för specifikationen OIDC är[20]:

- profile: efterfrågar åtkomst till standard claims för profil
- email: efterfrågar åtkomst till claims för email och email_verified
- address: efterfrågar åtkomst till claims för adress
- phone: efterfrågar åtkomst till claims för phone_number och phone_number_verified

Dessa scopes är kopplad till en mängd specifika claims som vid användning av en access_token behörig till motsvarande scope kan retureras.

2.3.2 OpenID Connect

OpenID Connect [21] är ett autentiseringslager som bygger på OAuth 2.0 och har därav många likheter med OAuth 2.0. En tydlig skillnad är att openid är ett scope som måste anges för att använda OpenID Connect, detta visar att den struktur som väntas följas följer OpenID Connect specifikationen. En ytterligare skillnad är att OpenID Connect använder sig utav en id_token som innehåller claims om autentiseringen av användaren. Bild 3 visar OpenID Connects protokollflödesprincip och skiljer sig från OAuth 2.0 bland annat genom att resursservern och auktoriseringsservern nu slagits ihop och benämns nu som en identitetsleverantör/IdP.

16 https://auth0.com/docs/api-auth/tutorials/adoption/scope-custom-claims
17 https://developer.okta.com/blog/2017/07/25/oidc-primer-part-1
18 https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml
19 https://auth0.com/docs/api-auth/tutorials/adoption/scope-custom-claims
20 https://developer.okta.com/blog/2017/07/25/oidc-primer-part-1
21 https://openid.net/specs/openid-connect-core-1_0.html

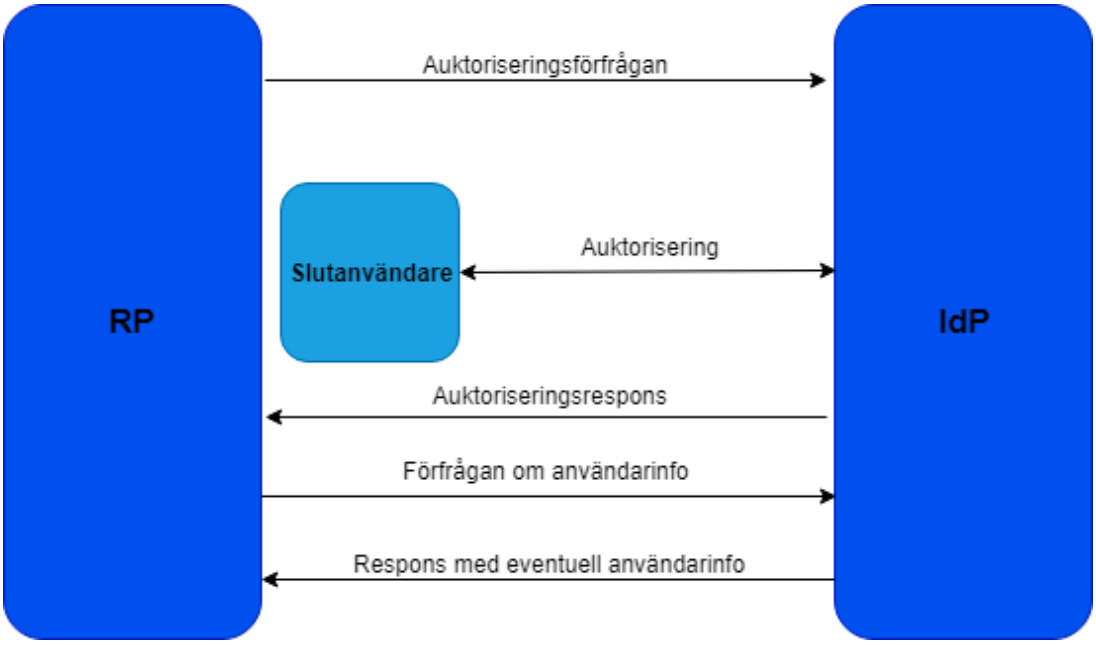


Bild 3: OpenID Connect protokollöversikt

OpenID Connect har tre huvudsakliga flöden (flows[22]), liknande OAuth 2.0 och dess grant types/flows. Beroende på vilket flöde som används skickas olika information mellan RP och IdP på olika sätt, nedan beskrivs detta i större detalj. Authorization Code och Implicit Flow återfinns i OAuth 2.0 och är mycket lika mellan OpenID Connect och OAuth 2.0.

1. Authorization Code Flow: returnerar en auktoriseringskod som kan utbytas mot en id_token och/eller en access_token. Detta flöde kräver ett client_id och client_secret för att hämta tokens och detta ska skötas från backend så att tokens döljs för användaren/webbläsaren. Detta flöde möjliggör åtkomst över längre tid genom refresh_tokens. Får auktoriseringskoden från en auktoriseringsendpoint och alla tokens från en token endpoint.
2. Implicit Flow: gör förfrågan av tokens utan explicit autentisering av klienten (utan client_secret/client_id), istället används endast redirect_uri för att verifiera klientens identitet. Refresh-tokens är inte tillåten och åtkomst över en längre tid med samma access_token är inte rekommenderat. Simplaste flödet att implementera då all kommunikation sker över en enskild request och response. Detta flöde får alla tokens från auktoriserings endpointen.
3. Hybrid Flow: kan kombinera aspekter ur tidigare två nämnda flöden. Kan användas för lång åtkomst med refresh_tokens och klienten ska ha kapacitet att bevara en client_secret.

response_type	**flow**
code	Authorization code flow

22 https://www.scottbrady91.com/OpenID-Connect/OpenID-Connect-Flows

id_token	Implicit flow
id_token token	Implicit flow
code id_token	Hybrid flow
code token	Hybrid flow
code id_token token	Hybrid flow
Tabell 2: Visar vilka response_types som hör ihop med vilka flöden	

Dynamic registration and Discovery

OpenID Connect har även, förutom sin kärnspecifikation[23], stöd för dynamisk registrering[24] av klienter samt stöd för att upptäcka användarens OpenID provider/identitetsleverantör dynamiskt[25]. Detta kräver ytterligare konfiguration vid implementationer och innebär fler aspekter att ta hänsyn till för en att skapa säkra implementationer, bland annat åtgärder för att motverka impersonation attacks.

2.3.3 Facebook Login/Authentication

Facebook login är en funktion/tjänst som Facebook tillhandahåller för att utvecklare ska kunna låta sina användare identifiera sig med ett Facebook-konto [18]. Ett antal SDK:er (Software Development Kit) tillhandahålls för att underlätta implementation av denna inloggning till exempelvis iOS-applikationer, Android-applikationer, webbsidor (JavaScript) med mera. De tillhandahåller även instruktioner för hur man manuellt kan bygga ett inloggningsflöde utan hjälp utav en SDK[26]. Facebook Login utnyttjar OAuth 2.0 som en bas men använder inte OIDC utan ett eget autentiseringslager.

De absolut mest grundläggande stegen för att autentisera med Facebook login är att skicka en GET förfrågan till ”https://www.facebook.com/v3.3/dialog/oauth” med parametrarna client_id, redirect_uri och state. Vidare kan även en response_type parameter definieras för att deklarera vilken typ av respons som önskas (bland annat code eller token) samt parametern scope som anger vilka rättigheter applikationen önskar sig utnyttja. Om inte en response_type definieras kommer code användas och vid lyckad förfrågan/inloggning behöver denna code bytas ut mot en access_token för att få information om användaren (exempelvis email). Om response_type code använts behöver denna på samma sätt som tidigare nämnda flöde ”Authorization code flow” bytas ut mot en access_token, detta görs genom att en förfrågan skickas till ”https://graph.facebook.com/v3.3/oauth/access_token” med parametrarna client_id, redirect_uri, client_secret och code. Responsen om denna förfrågan lyckas innehåller en access_token och information om vilken typ av access_token det är och hur lång tid det är till denna token inte längre är giltig. Dessa tokens bör sedan inspekteras och information om detta samt andra säkerhetsaspekter finns tillgänglig i

23 https://openid.net/specs/openid-connect-core-1_0.html

24 https://openid.net/specs/openid-connect-registration-1_0.html

25 https://openid.net/specs/openid-connect-discovery-1_0.html

26 https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow

Facebooks guider för utvecklare [27] [28]. Förutom de guider som tillhandahålls för att bidra med säkrare och stabilare implementationer så sköter dessa endpoints mycket vad gäller säkerhet själva, exempelvis accepteras inte ej säkra anslutningar längre (sedan 6:e Oktober 2018) och inställningar finns där utvecklaren kan blockera den funktionalitet som inte används/önskas för att minska risk för missbruk i inställningarna för den applikation som används.

2.4 Webbläsartillägg - Chrome

Ett webbläsartillägg för Chrome är ett Javascriptbaserat program som arbetar med webbläsaren för att modifiera/lägga till funktionalitet och har tillgång till liknande resurser som webbläsaren har tillgång till. Webbläsartillägg[29] är tänkt att ha en specifik uppgift och ha ett minimalt användargränssnitt.

2.4.1 Uppbyggnad

Ett webbläsartillägg kan bestå av en mängd olika komponenter som kan variera beroende på tilläggets syfte men det varje tillägg behöver är en manifest-fil[30]. En manifest-fil innehåller information om tillägget som namn, version, bakgrundsskript, rättigheter med mera[31]. Ett bakgrundsskript behövs för att ge programmet en inledande instruktion på vad som ska hända, i denna fil kan bland annat ett antal listeners registreras som lyssnar på när en specifik händelse skett och utför något. Rättigheter kan definieras i manifest-filen för att få tillgång till en mängd API:er för att utöka tilläggets tillgång till resurser. Vidare kan filer för att initiera och visa ett användargränssnitt definieras i manifest-filen samt filer för ikoner, inställningar med mera.

2.4.2 Begränsningar

Då webbläsartillägg är baserad på Javascript och arbetar med en webbläsare ärver tilläggen begränsningar från detta språk och denna miljö. Dessa begränsningar inkluderar bristen på förmåga att komma åt eller manipulera information som är ämnad för backend då webbläsartillägg hör till frontend. Vidare får inte tillägg skriva eller läsa filer från användarens system då de är begränsad till den lagring och miljö som existerar i webbläsaren (med undantag för cookies som är små filer som är starkt knutna till webbläsaren)[32]. Vidare är Javascript begränsad vad gäller förmåga att avläsa HTTP-trafik då åtkomsten till bland annat POST-trafik är mycket begränsad och inte tillgänglig på något traditionellt vis. Chromes utvecklarsida tillhandahåller en lista av API:er[33] som ger en idé om vad som är möjligt att göra med webbläsartillägg för Chrome.

27 https://developers.facebook.com/docs/facebook-login/security
28 https://developers.facebook.com/docs/graph-api/securing-requests
29 https://developer.chrome.com/extensions
30 https://developer.chrome.com/extensions/overview
31 https://developer.chrome.com/extensions/getstarted
32 https://www.thoughtco.com/what-javascript-cannot-do-2037666
33 https://developer.chrome.com/extensions/api_index

2.5 OAuthGuard

OAuthGuard är ett webbläsartillägg utvecklat för att i första hand skydda användare mot bristfälliga OAuth 2.0 och OpenID Connect implementationer (men fokus på OAuth 2.0 – klienter, utvecklat med Chrome version 63) [19]. OAuthGuard finns tillgänglig i Chrome Web Store [27] och källkoden finns tillgänglig på Github[34].

Strukturen för OAuthGuard är fördelad över fyra JavaScript-filer och tre huvudsakliga komponenter. De tre komponenterna listas nedan.

1. “OAuth 2.0 Detector”: avläser HTTP meddelanden och bedömer vilka som är OAuth 2 meddelanden. Finns både ”oauth” och ”redirect_uri” med i URL:en bedöms det vara en OAuth 2.0 request. Finns ett av orden ”code”, ”access_token” eller ”id_token” anses det vara en OAuth 2.0 respons. Informationen om HTTP meddelandena som bedöms vara en request/response extraheras och lagras i localStroage med tjänsteleverantörens domän som nyckel.

2. “Vulnerability Analyser”: se vad OAuthGuard skyddar mot för attacker.
    - Om en OAuth 2.0 response upptäckts av ”OAuth 2,0 Detector” så analyseras detta meddelande efter potentiella brister och använder motsvarande request som referens. Om ingen request med motsvarande domän som tjänsteleverantören har hittas i localStorage undersöks det om denna domän finns med på en whitelist[35] (exempelvis om de använder proxy för inloggningar), om så inte är fallet anses detta vara en intentional privacy leak.
    - Om en motsvarande request däremot kan hittas i localStorage påbörjas processen för att bedöma säkerhetsbrister.
        - CSRF: Om state parametern inte finns i respons så är det CSRF samt om referer headern till OAuth 2.0 responsen inte pekar till den IdP domän som använts eller RP domänen så räknas även detta som CSRF.
        - Impersonation attack: om bara en access_token finns i responsen är det impersonation attack
        - Authorization Flow misuse: om en kombination av code, access_token och id_token finns i responsen är det authorization flow misuse
        - Unsafe token transfer: om det är HTTP istället för HTTPS för att överföra responsen är det unsafe token transfer threat
        - Privacy leaks: Kollar först om en ”code”, ”access_token” eller ”id_token” finns i referer headern. Om någon av dessa tokens finns extraheras domänen av referer header i responsen och HTTP förfrågan, är dessa olika så är det en unintentional privacy leak

3. “Vulnerability Protector”:
    - Blockerar en HTTP förfrågan om en Privacy Leak är upptäckt
    - Försöker omdirigera en OAuth 2 respons med HTTPS om Unsafe Token Transfer upptäcks

34 https://github.com/oauthguard/OAuthGuard

35 https://sv.wikipedia.org/wiki/Vitlista

- Varnar användaren om RP är sårbar för Impersonation attacks
- Blockerar Oauth2 respons om en CSRF attack upptäcks

OAuthGuard använder strict Referer validation samt kollen efter state för att bedöma om en CSRF attack finns och borde blockeras. Denna metod är beroende av Referer headern, används HTTP går det inte att avläsa denna information och CSRF skyddet fungerar då inte som tänkt, samma gäller om Referer header skulle vara tomt.

OAuthGuards användargränssnitt nås med en ikon i Chrome-webbfönstret och tillhandahåller knappar som omdirigerar en användare till andra sidor med information eller funktionalitet. Tabbarna ”OAuthGuard”, ”Warnings” och ”Tools” finns. Under tabben ”OAuthGuard” visas hur många attacker som upptäckts kategoriserat per vilken typ av attack det är som som upptäckts (CSRF som blockerats, referer token leaks som blockerats, third party token leaks som blockerats samt antalet HTTPS upgrades som gjorts). Under ”Warnings”-tabben listas de hemsidor som lider av impersonation attacks. Under tools finns fem knappar, ”Check Report”, ”Open OAuth 2.0 Tools”, ”Export data”, ”Turn on/off HTTPS upgrade” samt ”Vulnerability Description”. Check report knappen listar alla brister som noterats sorterat på domän. Open OAuth 2.0 Tools öppnar ett fönster där domäner kan sökas på för att få detaljerad information om request/respons som gjorts för denna domän. Export data skapar en fil med den data som lagrats i localStorage som laddas ned. Turn on/off HTTPS upgrade sätter på/stänger av funktionen som försöker omdirigera trafik från HTTPS till HTTP. Vulnerability Description listar information om de säkerhetsbrister OAuthGuard upptäcker/skyddar mot.

2.6 Säkerhetshot OAuthGuard behandlar

OAuthGuard är designat för att upptäcka fem typer av hot samt att skydda mot fyra (listad under rubrik 2.5) [19]. OAuthGuard ska skydda mot: CSRF oavsett om state parametern använts eller inte, detekterar om en RP är misstänkt för impersonation-attacks och varnar användaren om detta, kan upptäcka Authorization Flow misuse, kan upptäcka brist av HTTPS och försöka omdirigera så HTTPS används samt upptäcker privacy leaks och blockerar dem.

2.6.1 CSRF

Cross-Site Request Forgery (CSRF) är en attack som på något sätt får en användare att mot sin vilja utföra förändringar mot en webbapplikation som denna användare är autentiserad mot, denna typ av attack utsätter användarens resurser för fara och om användaren även är en administratör kan mer än bara en individs resurser påverkas. Denna attack kan utföras exempelvis genom att användaren på något sätt dirigeras till en URL där användaren är autentiserad med information som gör att den webbsida som användaren hamnar på utför en händelse som användaren inte villigt utfört (exempelvis information i URL via en GET)[36].

36 https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Förutom att tjänsteleverantörer med OAuth 2.0 kan vara öppen för de traditionella CSRF-attacker kan de även vara öppen för missbruk och manipulering av dess redirect_uri som kan leda till att en obehörig person får obehörig tillgång till en användares resurser. Angriparen startar attacken genom att själv skaffa tillgång till en code eller access_token för sina egna resurser. Sedan avbryter angriparen omdirigeringsflödet för sin autentisering tillbaka till RP och får istället användaren/offret att utföra denna omdirigering till RP. Detta leder till att RP associerar offrets RP-session med angriparens resurser som går att komma åt med hjälp utav de tokens som genererats. Detta leder till att en användare exempelvis kan ladda upp känslig data till angriparens resurslagring som sedan en angriparen kan komma åt utan att veta användarens lösenord eller användarnamn [20]. Skydd mot CSRF i samband med OAuth 2.0 innebär att tillföra en state parameter som är unik för en inloggningssession och som ska valideras att den förblir densamma genom alla omdirigeringar. Ett antal andra lösningar på CSRF finns men då användning av OAuth 2.0 innebär omdirigeringar som beter sig som CSRF fungerar inte alltid dessa [20].
Vissa endpoints behöver skydd utöver det skydd som state parametern ger mot CSRF attacker, exempelvis genom att även kontrollera origin header[25][19].

2.6.2 Impersonation attack

En attack som rotar sig i förvirring mellan autentisering och auktorisering (relaterad till OAuth2). Viktigt att komma ihåg att access_token (en bearer token) är tänkt för auktorisering och är inte bunden till någon specifik RP (inte lämplig för autentisering), detta gör det möjligt för vilken RP som helst som får tillgång till denna token att använda den för att få tillgång till resurser som denna access_token ger tillgång till samt att autentisera sig på de ställen som inte implementerat skydd för impersonation attack och som accepterar access_token som autentiseringstoken [19]. Attacken går ut på att token använts som respons-typ, då en access_token/token returneras direkt via autentisering istället för en code som sedan byts ut mot en access_token. Denna token kan sedan användas av användaren på en skadlig sida/klient som fångar upp denna token och sedan försöker autentisera sig på andra sidor med denna token (se sida 60 i [14]). Om dessa sidor inte vidtagit åtgärder för att inspektera token/kräver ytterligare information för autentisering kan angriparen logga in som offret och få tillgång till offrets resurser. Inspektionen av tokens kollar vilken app token var genererad för och användandet av tokens som inte var skapad för den app som token hamnat hos kan då stoppas.

2.6.3 Authorization Flow Misuse

Endast en code ska skickas från RP till inloggnings-endpointen som bevis att användaren autentiserat sig, inte en kombination av code, access_token och id_token[19]. Denna typ av attack är endast tillgänglig när hybrid-flow används så det är detta flöde som gör det möjligt att få tillgång till alla dessa tokens/codes [22].

2.6.4 Unsafe Token Transfer

OpenID Connects säkerhet är beroende av konfidentialiteten och integriteten av transportlagret för att kunna skydda de tokens/information som överförs. SP's och IdP:er borde använda HTTPS vid kommunikation. Vidare kan "HTTPS strict transport security" och "public key pinning" användas för att ytterligare utöka säkerheten[25].

2.6.5 Privacy Leakage

En token ska inte visas eller göras tillgänglig på något sätt för någon annan part än tänkt RP, men detta kan hända frivilligt eller ofrivilligt. En ofrivillig läcka kan bero på att en RP har tredjepartskod på samma sida som inloggningsflödet (där tokens exponeras) medan en frivillig läcka innebär att en RP medveten skickar tokens till en tredje part [19]. Försiktighet ska vidtas så att värdet på state eller auktoriseringskoden inte skickas till en opålitlig tredjepart via referer header. För att förhindra att sådan information skickas och/eller används av en okänd tredjepart ska de endpoints som involverar denna URL:information hållas fri från, exempelvis, länkar till externa hemsidor. Som extra försäkring, ifall informationen ändå skulle hamna på fel ställe, ska värdet i en state-parameter endast gå att använda för ett inloggningsförsök och den code som skickas med ska enbart gå att använda för att lösa ut en token en gång [25].

2.7 Relaterade arbeten

Ett flertal studier har försökt sammanfatta och/eller belysa om säkerhetsbrister som förekommer samt föreslå lösningar [12, 19, 20, 24]. Exempel på attacker som tjänsteleverantörerna misslyckas att skydda sig mot innefattar Cross Site Request Forgery (CSRF), som kan leda till att en angripare kan kontrollera en användares konto [24]. Enligt [23] saknades de vanligaste skydden mot CSRF helt i över 60% av studiens undersökta hemsidor samt att färre än hälften av de resterande klienter som försökte implementera detta skydd lyckades, vidare fann [19] allvarliga säkerhetsbrister i 69 av 137 undersökta klienter som använde sig av Googles OIDC-tjänst samt att de mer populära hemsidorna oftare hade säkerhetsbrister. I slutet av 2013 hade 345 av 1660 undersökta tjänsteleverantörer med Facebook som identitetsleverantör säkerhetsbrister relaterat till autentisering/auktorisering som gick att upptäcka med SSOScan, vidare visade det sig att populära webbsidor oftare hade stöd för Facebook som identitetsleverantör jämfört med mindre populära webbsidor [21]. Studierna [19, 21] har försökt uppmärksamma de bristande tjänsteleverantörerna via email eller webbformulär om deras säkerhetsbrister i syfte att tjänsteleverantörerna ska åtgärda dem, men utan större framgång då flertalet ej svarade och/eller ej åtgärdade bristerna. Det spekuleras att detta kan bero på att de bristande tjänsteleverantörerna ej har dedikerade säkerhetsavdelningar eller sätt att adressera säkerhetsbrister [19, 20]. I [22] nämns det att en stor risk finns att bristfälliga implementationer kommer fortsätta produceras då bland annat auktoriseringskodflödet kan lämna utvecklaren med valmöjligheter som vid fel val kan leda till säkerhetsbrister. Vidare skrivs det i [9] att många tjänsteleverantörer inte följer de instruktioner som finns i specifikationen för att utveckla säkra implementationer, vilket märks då de brister implementationerna har behandlas i protokollets specifikation. Anledningen till dessa brister kan vara ofullständig förståelse av konsekvenser vid uteblivna säkerhetskontroller [9]. De bibliotek undersökta av [9] hade alla brister relaterat till attacker som inte nämns i

specifikationen vilket enligt [9] är ett förväntat resultat då implementationer som till fullo följer specifikationen ändå löper risk att vara mottaglig för dessa attacker. Det finns alltså en mängd anledningar till att implementationer kan ha säkerhetsbrister och tidigare studier [9, 19, 20, 21, 22] visar att befintliga specifikationer och instruktioner inte ännu löser problemet med bristfälliga implementationer.

En annan strategi att hantera säkerhetsproblemet är att utveckla hjälpmedel och verktyg för att skydda användarna, antingen för att användaren själv ska kunna skydda sig med ett verktyg eller för att utvecklaren ska kunna använda verktyget så att säkerhetsbrister upptäcks innan publicering av en tjänst utförs. En del olika verktyg för detta syfte har utvecklats, däribland OAuthGuard [19], WPSE [11], SSOScan [21] och PrOfESSOS [9] med tillhörande studier. OAuthGuard har även utvecklats med den mer unika visionen att direkt skydda användaren istället för att vara ett verktyg i huvudsak för utvecklare.

Då studien med SSOScan [21] är den (till författarens kunskap) enda studien som utförts för Facebook SSO är denna studie intressant ur både utvecklarsynpunkt och ur jämförelse-synpunkt. SSOScan skiljer sig däremot ganska mycket från OAuthGuard både vad gäller syfte och struktur av verktyget men även vad gäller vilka säkerhetsbrister som på något sätt behandlas. De fem säkerhetsbrister som SSOScan behandlar listas nedan[37], ingen av dessa behandlar CSRF, som är ett stort fokus i OAuthGuard:

- Access token Misuse: risken med att access_tokens kan användas av andra, föreslår att man använder code eller signed_request. Säger inte explicit vilken access_token som används, finns i alla fall tre stycken för Facebook. Antagit app access-token.
- Signed_request misuse: validering av signed_request är viktig, används även en access_token ska dessa två valideras mot varandra också (så de hör ihop). Länken till detaljer fungerar inte längre.
- Credential leakage via referrer header: tredjepartskod som kan skada om access_token skickas via URL, går då att få tag på via referrer headern. Denna information bör konsumeras via backend.
- Client secret leakage: Att inte lagra client_secret i frontend så detta kan leda till att andra kan använda den för att skapa egna requests.
- Credential leakage via page content: att inkludera credentials som access_tokens eller dylikt på registreringssidan efter att en användare loggat in kan vara farligt. För att lösa detta bör inte access_tokens lagras i innehållet av sidan utan bör hämtas från servern med AJAX eller dylikt.

SSOScan studien [21] kom fram till att populärare sidor oftare hade stöd för Facebook SSO. Vidare hittade studien att det var vanligare att populära sidor missbrukade tokens/viktig information oftare men läckte den mer sällan än mindre populära sidor. Av de 17913 sidor som undersöktes med SSOScan hade 1660 stöd för Facebook varav totalt 345 (20.3%) hade någon form utav säkerhetsbrist. OAuthGuard-studien [19] undersökte 1000

37 http://ssoscan.org/vulnerabilities/

sidor och hittade att 137 klienter hade stöd för Google SSO och att totalt 69 (50%) hade någon form utav brist, vidare hade 53 stycken (39%) CSRF, 21 stycken (15%) missbrukade authorization flow varav 13 av dessa var öppen för impersonation-attacks, 9 stycken klienter läckte tokens via referer header varav två av dessa 9 gjorde detta explicit och slutligen använde 13 stycken klienter HTTP istället för HTTPS. Studien [19] visade även att populärare sidor oftare hade stöd för Google SSO men även oftare hade brister där bristerna och Google SSO möjligheten är vanligast bland de sidor som har en rank mellan 100 och 200. En av de sidor [19] nämner som har stora problem är issuu.com med risk för impersonation attack med mera.

Författarna till OAuthGuard-studien [19] W Li och CJ Mitchell har tidigare publicerat flertalet artiklar inom området SSO och OpenID Connect som blivit väl citerade och har även bidragit med ett antal undersökningar om hur säkerhetsläget ser ut för OpenID Connect/OAuth 2.0 SSO klienter. W Li har bidragit med [24] och [22] som har citerats 37 respektive 32 gånger enligt Google Scholar, jämförbart med OpenID Connect specifikationen[38] som blivit citerad 112 gånger eller OAuth 2.0 RFC [14] som citerats 850 gånger. CJ Mitchell har bidragit i en stor mängd artiklar inom säkerhetsområdet/kryptografi och har totalt citerats över 6000 gånger. Jämfört med WPSE är OAuthGuard lättare tillgänglig, studierna i sig skiljer sig inte så mycket vad gäller popularitet, OAuthGuard har däremot vinklat sitt syfte mer till användare snarare än utvecklare och har därav ett syfte som till författarens vetskap inte undersökts tidigare.

Detta arbete bygger på det tidigare arbetet som utförts på OAuthGuard [19] och fortsätter undersöka potential med verktyg som riktas mot att skydda användaren då många andra metoder visat sig bristfällig för att lösa säkerhetsproblemen med SSO. Studien kommer utföra en liknande studie där de populäraste hemsidor som använder Facebook som identitetsleverantör kommer undersökas med verktyget OAuthGuard och jämföras mot den ursprungliga studiens resultat för hemsidor som använder Google som identitetsleverantör [19]. Detta för att undersöka möjligheter med verktyg som direkt skyddar användaren samt för att bidra med ny information och en jämförelse mellan Google SSO och Facebook SSO.

38 https://openid.net/specs/openid-connect-core-1_0.html

3 Metod

Kapitlet beskriver vilken metod som använts för att nå resultatet, under 3.1 beskrivs det hur den framställning av den teoretiska delen av arbetet utförts, vidare under 3.2 beskrivs vilka verktyg som använts (3.2.1) och vilka krav (3.2.2, 3.2.3) som ställts.

3.1 Angreppssätt

Då den huvudsakliga målsättningen med studien baseras på en vidareutvecklad version av OAuthGuard som även stödjer Facebook som identitetsleverantör behövde denna vidareutveckling utföras först. Till en början utfördes en undersökning över befintlig implementation och identifiering över vad som saknades för att motsvarande funktionalitet som fungerar för Google-SSO skulle fungera för Facebook-SSO och vilka moduler som behövde ändras. När detta gjorts började den nya funktionaliteten implementeras, precis som tidigare implementerad funktionalitet med hjälp utav att analysera och hantera HTTP-meddelanden. För att testa så att den utökade funktionaliteten fungerar som den ska utvecklades en testklient. Denna klient utvecklades med minimalt gränssnitt för att utföra en inloggning med Facebook-SSO samt för att kunna efterlikna säkerhetsbrister som OAuthGuard är tänkt att behandla. De brister som inte kunde implementeras i testklienten testades sedan med verktyget manuellt med hjälp utav att köra tillägget/verktyget vid inloggning på befintliga klienter/webbsidor och analysera HTTP-meddelanden och meddelanden från tillägget i Google Chrome konsolen, både de som visat brister samt de som inte gjort det. Baserat på dessa tester bedömdes det om verktyget kunde upptäcka och behandla brister där den skulle det (enligt kriterier nämnda i föregående studie [19] och som listats under 2.5). När detta var klart samlades information in inför undersökningen om säkerhetsläget vid autentisering med Facebook-SSO. De 500 populäraste webbsidorna/domänerna undersöktes manuellt och de klienter som hade stöd för Facebook-SSO valdes ut. Dessa utvalda klienter delades in i fem grupper rankad efter popularitet som sedan undersöktes med OAuthGuard genom att i tur och ordning göra en inloggning på dessa sidor medan tillägget är aktivt för att sedan dokumentera vilka eventuella brister tillägget OAuthGuard och Google Chrome konsolen anmärkt på.

3.1.1 OAuthGuard

Strukturen för hur första versionen av OAuthGuard är uppbyggd undersöktes noggrant för att bedöma vart och hur ny funktionalitet ska implementeras. Enligt OAuthGuard-studien [19] är tillägget uppbyggd på tre huvudsakliga moduler: “OAuth 2.0 Detector”, “Vulnerability Analyser” och “Vulnerability Protector”. Till en början analyserades dessa moduler teoretiskt utifrån vad studien [19] skrivit, därefter analyserades koden för att identifiera dessa moduler och hur de är fördelad över de fyra script som OAuthGuard består utav. Då studien beskriver att tillägget i huvudsak analyserar och bearbetar HTTP-meddelanden lokaliserades vart detta utfördes och vilka parametrar och headers som undersöktes. Då en del av verktyget är att presentera vilka brister som hittats, undersöktes även vilken struktur som använts för att lagra information om de tjänsteleverantörer som har upptäckta brister. Baserat på information om vad modulerna har ansvar för och vart i koden de är placerade

gjordes en bedömning över vilka moduler som behövde modifieras och vad som behövde tillföras för att Facebook ska kunna användas som identitetsleverantörsalternativ under skydd av OAuthGuard. Denna bedömning tog hänsyn till hur stora skillnader Google och Facebook som identitetsleverantör har vad gäller autentiseringsflödet och vilka parametrar och headers som används. Om skillnaderna är stora, så flertalet obligatoriska parametrar som skiljer sig eller om flödet skiljer sig, behöver nya submoduler skapas men är de tillräckligt små kan funktionalitet integreras direkt till befintlig struktur. Vidare testades verktyget så som det finns tillgängligt utan modifikation både för Google och mot Facebook för att se vad tillägget klarar av i nuläget och vad det inte klarar av, detta kontrollerades genom analys av vilken data som sparas i localStorage[39] (som är tilläggets val av lagring) samt vidare analys av koden. Vidare analyserades koden för den befintliga implementationen för att se hur dessa säkerhetsbrister upptäcks/åtgärdas för Google och om det är möjligt att upptäcka samma brister mot Facebook SSO. Baserat på resultaten av detta gjordes en plan för vilken minimal modifikation som skulle krävas för att få tillägget att fungera för både Facebook och Google, i första hand fokuserades det endast på funktionalitet som gör det möjligt att göra den empiriska undersökningen med tillägget och inget speciellt fokus på användarvänlighet eller stabilitet vid olika användarscenarios. Utvecklingen av denna utökade funktionalitet skedde iterativt. Första steget var att utföra någon form utav modifikation, sedan testades denna modifikation på en mängd hemsidor som tillhör det urval som den empiriska studien ska utföras på (se rubrik 3.1.3) för att analysera säkerhetsläget hos klienter med Facebook SSO, samt test på bland annat testklienten (se rubrik 3.1.2 samt rubrik 3.1.4) för att bekräfta att rätt säkerhetsbrist uppstår av rätt anledning. HTTP-trafiken och den data som lagrades av tillägget analyserades sedan tillsammans med Chromes utvecklarkonsol[40] och koden för tillägget för att bedöma vilka fler förändringar som behöver göras. Denna process upprepades några gånger och resultaten av testning på de hemsidor som ingår i den empiriska undersökningen noterades tillsammans med förfining/nya strategier för hur testningen med tillägget på klienterna skall utföras.

3.1.2 Tjänsteleverantör

Skapandet av tjänsteleverantören som stödjer Facebook som identitetsleverantör utfördes med hjälp utav Springboot och Angular, här hade vilket verktyg som helst kunna användas som klarar av att arbeta med HTTP förfrågningar/responser samt som ger stöd för ett backend så Authorization Code Flow kan tillämpas. Springboot och Angular valdes framför något annat då utveckling med dessa skulle gå snabbast under författarens förutsättningar. Denna tjänsteleverantör skapades med huvudsyftet att dess kod ska kunna modifieras så att den får säkerhetsbrister för att sedan kunna användas för att testa verktyget OAuthGuard. För att göra denna funktionalitet möjlig behövdes en webbsida där det går att initiera och utföra en inloggning mot Facebook på, detta kräver ett minimalt gränssnitt. Vidare behövdes en hantering av autentiseringsflödet som till så hög grad som möjligt går att modifiera, framförallt vad gäller vilka parametrar som används och hur de används, därav begränsades antalet bibliotek och SDK:er som gick att utnyttja då många bibliotek medför en för hög abstraktionsnivå och därav inte går att modifiera på parameternivå. En

39 https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
40 https://developers.google.com/web/tools/chrome-devtools/console/

SDK som inkluderades var Nimbus OAuth 2.0 SDK[41] för Java (oauth2-oidc-sdk från com.nimbus), denna SDK bidrar med en mängd värdefulla funktioner, bland annat generering av ett state.

Facebooks och Googles specifikationer för deras SSO funktionalitet studerades tillsammans med de säkerhetsbrister som OAuthGuard upptäcker och/eller skyddar emot för att bedöma vilka av säkerhetsbristerna en tjänsteleverantör kan efterlikna med Facebook SSO för att sedan kunna implementeras. Endast brister som inte kräver kontroll över identitetsleverantör eller kräver flera tjänsteleverantörer/tredjepartskod kommer tas hänsyn till för att inte utvecklingen av denna testklient ska ta för lång tid.

3.1.3 Empirisk undersökning

Studien som utvecklade OAuthGuard använde majestic million[42] som källa och denna källa användes även för denna studie, majestic million kollar statistik för hela världen genom att ranka hur många gånger en sida refererats till (backlinks[43]). Den liknande studien med SSOScan använde däremot statistik från www.quantcast.com som behandlar trafik endast från USA (hur ofta en sida besöks på en månad), men för att kunna jämföra data med så lika premisser som möjligt valdes samma källa som den ursprungliga studien [19].

Den empiriska undersökningen påbörjades med att extrahera en mängd (de 500 populäraste) sidor från majestic-million rapporten. Dessa sidor söktes sedan manuellt igenom för Facebook SSO samt Google SSO och detta noterades. Av de hemsidor som visade sig ha SSO med Facebook räknades domäner som dirigeras om till samma sida bort och endast en av dessa räknades. Tidigare studie hade inte nämnt någon exkludering av några webbsidor men om dubbletter av samma webbsida förekommer i studien bedömdes resultaten bli mindre pålitliga då samma webbsida med eventuella brister hade räknats som två eller fler webbsidor. Fördelning av dessa exkluderade dubbletter av domäner bland de 500 undersökta webbsidorna noterades.

De 500 domäner som totalt undersökts för Facebook SSO delades upp i fem grupper om 100 baserad på rank på samma sätt som tidigare studie gjort [19] för att senare kunna bedöma hur data förändras med popularitet. Dessa grupper undersöktes sedan manuellt i omgångar med uppdaterade tillägget OAuthGuard (med utökad funktionalitet för Facebook) och data samlades in. Denna data består utav:

1. Totala antalet klienter inom specifik grupp som har någon form utav brist.
2. Totala antal klienter som lider av en specifik brist inom specifik grupp.
3. Totala antal klienter som har en ej fungerande implementation av Facebook SSO.
4. Totala antal klienter som stödjer Facebook SSO inom specifik grupp.
5. Totala antal klienter som stödjer Google SSO inom specifik grupp.

41 https://connect2id.com/products/nimbus-oauth-openid-connect-sdk
42 https://majestic.com/reports/majestic-million
43 https://en.wikipedia.org/wiki/Backlink

En mängd undersökningar utfördes med verktyget under olika utvecklingsstadier och med olika metoder för att analysera eventuella variationer och för att slutligen bestämma en metod för den slutgiltiga undersökningen med slutgiltiga versionen av OAuthGuard för denna studie. Tidigare studie [19] angav ingen specifik metod annat än att manuellt gå igenom utvalda hemsidor med verktyget OAuthGuard, vilket hade varit önskvärt för att generera data på så lika villkor som möjligt för bäst jämförbarhet, därav testades ett antal metoder för undersökningar för att undersöka hur resultat kan påverkas beroende på metod. Den manuella körningen med OAuthGuard utfördes först två gånger på alla Facebook SSO klienter ingående i undersökningen i testsammanhang vid vidareutveckling av tillägget. Den generella metoden för undersökningar (se bild 4) är att använda ett temporärt Facebook-konto (som inte kopplats till viktig information) och Google Chrome webbläsaren i inkognito för att surfa in på webbsidorna och utföra en inloggning med detta Facebook-konto. En ny browser-session startas för varje grupp och tillägget återställs (data som lagrats av tillägget tas bort och Chrome konsolen töms på meddelanden).

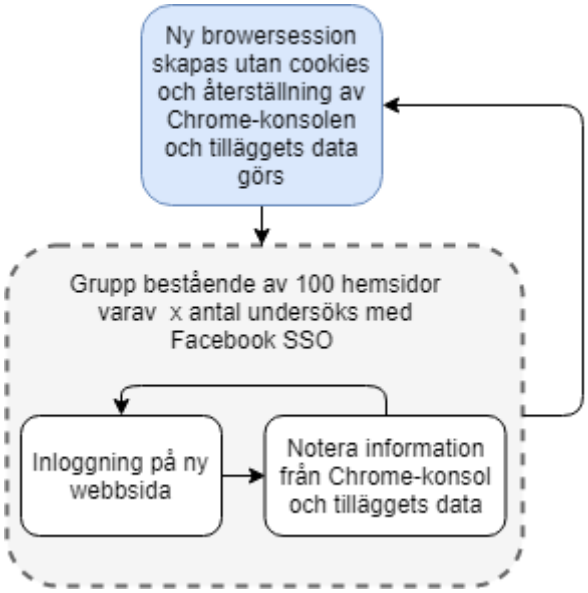


Bild 4: Generell metod för undersökningar (ej den slutgiltiga undersökningen)

Vidare utfördes även ett antal mindre undersökningar på den grupp som visade sig både ha flest brister och flest klienter som stödjer SSO. En av dessa mindre undersökningar gick igenom de klienter som hade Google SSO, tre andra bestod av att undersöka denna grupp för Facebook SSO och se om resultaten varierade trots att samma metod använts för samma klienter. Slutligen utfördes den sista av de mindre undersökningarna på denna grupp med klienter som stödjer Facebook SSO där användning av cookies och spridning av information för riktad reklam i så stor grad som möjligt nekades. De sidor där inställningar gick att ändra eller de sidor där man var tvungen att acceptera cookies med mera noterades för vidare analys.

Efter dessa implementerades en grundläggande koll för referer-token leakage, dock infördes ingen whitelist för denna så alla genomgångna klienter med denna brist har manuellt granskats för att exkludera facebook-relaterade URL:er på samma sätt som

tidigare version av tillägget hade whitelist för Google-relaterade URL:er. Den slutgiltiga undersökningen bestod av en hel genomsökning av alla de klienter med Facebook SSO (av de 500 genomgångna) där till skillnad från tidigare *varje* klient testades med en helt ny browser-session och med en rensning av cookies. Tilläggets lagrade data rensades mellan varje inloggningsförsök på samma sätt som tidigare beskrivet vid den generella metoden, nu bara efter varje klient och inte var hundrade. Viktig detalj för denna undersökning är att alla sidor fick ladda klart helt innan inloggningsförsöket utfördes då en del tredjepartskod/resurser tar tid att laddas in. Dubbletter av CSRF varningar för en sida kommer fortfarande bara räknas som en CSRF (då detta uppstår när tillägget av någon anledning registrerat två identiska responser till en förfrågan och därav räknar det som två brister), till skillnad från token leaks där alla räknas. En hemsida kan vara öppen mot CSRF-attacker och registrering av flera CSRF hot för identiska responser av samma anledning gör ingen direkt skillnad i hur stort hotet är, däremot indikerar flera token leaks på att flera tredjepartssidor får tillgång till känslig information och hotet ökar då om flera sådana förekommer. Föregående studie [19] nämnde inget om hur de hanterat detta. Se bild 5 för detaljerad redovisning av metod.

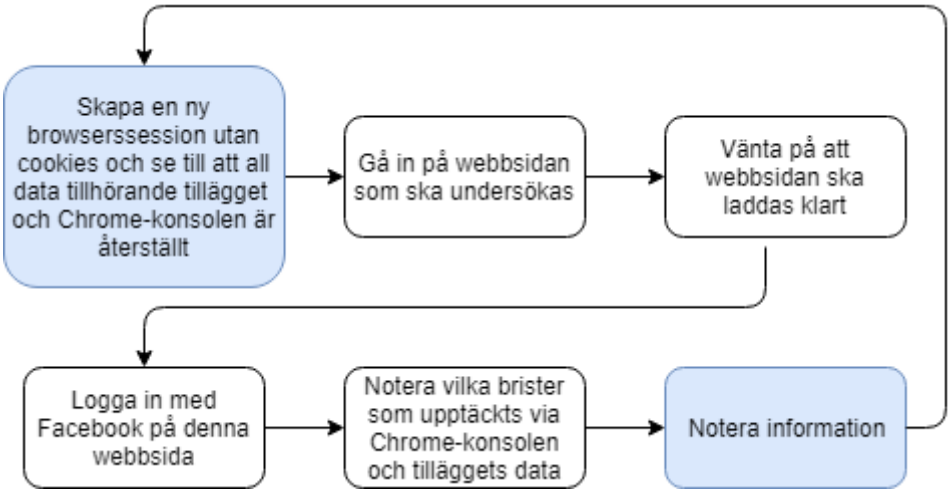


Bild 5: Metod för slutgiltig undersökning

Några stickprov av Google SSO kommer även göras, bland annat för hemsidan issuu.com som enligt tidigare studie hade stora problem. Detta beror på att dubbletterna av CSRF för utvald undersökningsgrupp uppstår då mer än en identisk respons registreras. Alla dessa undersökningar samlar data enligt de punkter som nämnts ovan.

Den data som samlats används analyserades/jämfördes sedan, både jämfört med varandra och jämfört med den tidigare studien [19] enligt de punkter/frågor som beskrivs under syfte (rubrik 1.2).

3.1.4 Testning

Till en början undersöktes det om den tidigare versionen av OAuthGuard hade testats med någon speciell strategi eller metod (enligt studien [19]), men detta saknades och OAuthGuard är ännu bara informellt testad. För att testa så att den utökade funktionaliteten av OAuthGuard fungerade som tänkt användes den testklient/tjänsteleverantör som

utvecklats under studiens gång tillsammans med ett temporärt Facebook-konto som inte är knuten till någon viktig identitet/resurs (för att undvika att utsätta viktiga resurser för fara vid testning på klienter med brister). OAuthGuard testades med denna leverantör (Facebook) på den utvecklade tjänsteleverantören som implementerats att ha brister som OAuthGuard är tänkt att upptäcka. Detta skedde genom att låta tillägget vara aktivt medan en inloggning utfördes på testklienten, både med och utan brist, för att avläsa om tillägget behandlar och analyserar HTTP-meddelandena korrekt och lyckas identifiera rätt hot vid rätt tillfälle och av rätt anledning.

Den utökade versionen av OAuthGuard som utvecklades testades i två omgångar på de klienter som valts ut för undersökning av säkerhetsläge vid Facebook-SSO. Detta för att både undersöka hur tillägget måste utvecklas vidare men även för att undersöka vilken metod som lämpar sig att använda för att göra den slutgiltiga kontrollen. Testerna utfördes som en variant utav Funktionell testning[44] där vissa kriterier testades för uppfyllelse (se kapitel 4.2.2). Dessa testomgångar dokumenterades och faktorer som förändring i resultat och beteende av tillägget noterades och användes som underlag för vidare modifikation av kod och testmetod. Under dessa testkörningar uppmärksammades och analyserades även tilläggets förmåga att upptäcka brister baserat på HTTP trafiken och vilken trafik tillägget valt att anmärka på (och vad tillägget anmärkt) i förhållande till vad tillägget är tänkt att anmärka på. Denna undersökning gjordes i huvudsak för de brister som testklienten inte klarar av att testa. Detta beskrivs mer i detalj under kapitel 4.2.

3.2 Implementation

3.2.1 Verktyg

Microsoft Visual Code version 1.33.1

NodeJs version 10.2.0

NPM version 6.4.1

Angular version 7

Spring Tool Suite version 3.9.8

Git / Bitbucket

Windows 8.1 Pro / Windows 10 Home

Google Chrome version 73.0.3683.103

3.2.2 Funktionella krav

OAuthGuard ska vidareutvecklas som ett Google Chrome-tillägg och därav stödja Google Chrome som webbläsare.

44 https://en.wikipedia.org/wiki/Functional_testing

Google Chrome tillägget OAuthGuard ska behålla sig funktionalitet för Google som identitetsleverantör med följande krav:

- Upptäcka säkerhetsbrister:
  - Upptäcka CSRF genom att kolla efter state-parametern i HTTP responsen.
  - Upptäcka ”Impersonation attacks” genom att kolla om endast en access_token finns i HTTP responsen
  - Upptäcka ”Authorization flow misuse” genom att kolla om en kombination av code, access_token och id_token finns i HTTP responsen.
  - Upptäcka ”Unsafe token transfer threat” genom att kolla om HTTP används istället för HTTPS för att överföra responsen.
  - Upptäcka ”Privacy leak” (oavsiktlig) genom att kolla om referensfältet i HTTP förfrågan är olika
- Motverka säkerhetsbrister:
  - Blockera CSRF
  - Varna om ”Impersonation attacks”
  - Försöker omdirigera responsen med HTTPS istället för HTTP om HTTP används
  - Varnar om en ofrivillig eller frivillig ”Privacy leak” hänt
- Verktyg:
  - Kolla säkerhetsrapport
  - Analyseringsverktyg
  - Exportera data
  - HTTPS uppgraderingsverktyg

Google Chrome tillägget OAuthGuard ska utöka sin funktionalitet till att även stödja Facebook som identitetsleverantör. Denna utökning borde, i så stor grad som möjligt, tillföra (utöver funktionaliteten som finns för Google som identitetsleverantör) följande funktionalitet:

- Upptäcka säkerhetsbrister:
  - Upptäcka CSRF genom att kolla efter state-parametern i HTTP responsen.
  - Upptäcka ”Impersonation attacks” genom att kolla om endast en access_token finns i HTTP responsen
  - Upptäcka ”Authorization flow misuse” genom att kolla om en kombination av code, access_token och id_token finns i HTTP responsen.
  - Upptäcka ”Unsafe token transfer threat” genom att kolla om HTTP används istället för HTTPS för att överföra responsen.

- Upptäcka ”Privacy leak” (oavsiktlig) genom att kolla om referensfältet i HTTP förfrågan är olika

- Motverka säkerhetsbrister:
  - Blockera CSRF
  - Varna om ”Impersonation attacks”
  - Försöker omdirigera responsen med HTTPS istället för HTTP om HTTP används
  - Varnar om en ofrivillig eller frivillig ”Privacy leak” hänt

3.2.3 Icke-funktionella krav

Behålla tidigare struktur på utseende med pop-up-fönster för tillägget när tilläggsikonen tryckts på. Pop-up-fönstret ska i alla fall ha tre tabbar:

- OAuthGuard
  - Visar statistik över vad tillägget hjälpt till med i form utav hur många attacker/säkerhetsbrister som blivit blockerade samt hur många HTTPS uppgraderingar som utförts ska finnas.
- Warnings
  - En lista av hemsidor som det inte rekommenderas att använda Google sign-in på ska finnas
  - En lista av hemsidor som det inte rekommenderas att använda Facebook sign-in på bör finnas
- Tools
  - Ska ha knappar för att visa:
    - Kolla säkerhetsrapport
    - Analyseringsverktyg
    - Exportera data
    - HTTPS uppgraderingsverktyg
    - Säkerhetsbrist-beskrivning
    - Guide/Användarinstruktioner
      - Ska tillhandahålla instruktioner för hur tillägget används

Utökningen av OAuthGuard kommer publiceras på GitHub och vara fritt tillgänglig för allmänheten, vidare kommer den kod som produceras hållas strukturerad för att underlätta underhåll och vidare utökning.

4 Konstruktion

Detta kapitel beskriver hur konstruktionen (utökning av OAuthGuard samt implementation av testklient) utförts i detalj, både vad gäller design och testning.

4.1 OAuthGuard

Som tidigare nämnt består strukturen för OAuthGuard av fyra JavaScript-filer och tre huvudsakliga komponenter, “OAuth 2.0 Detector”, “Vulnerability Analyser” och “Vulnerability Protector”. Dessa komponenter och koden för dem analyserades genom att bland annat testa verktyget, läsa vad som står om verktyget i studien [19] samt att förstå koden genom granskning samt kontroll av flöden med konsol-loggar. De fyra JavaScript-filerna och dess innehåll samt upptäckta brister listas nedan:

1. analyse.js:

   Innehåller funktioner för att hämta data om respons och request från en angiven RP. Skapar tabeller som visar denna data samt har knapp för att rensa data som visats från tabellerna. Denna knapp heter ”Clear”, den tar inte bort data från localStorage utan bara från de tabeller som genererats, tabellen tas däremot inte bort så när en ny tabell genereras skapas flera. Inte heller rensas tabeller från data om flera sökningar görs, så tabellerna fylls bara på med mer data.

2. popup.js:

   Kollar vilka tjänsteleverantörer som lider av impersonation attacks och skapar en lista av dessa som tillägget varnar om under sin ”warnings” tabb. Exporterar all data i localstorage till en fil om ”Tools → Export Data” tryckts på. Kör funktionen som upptäcker vilka säkerhetsbrister som finns, baserat på den data som lagrats i localStorage, vilka hot som upptäckts skrivs sedan ut till användargränssnittet.

3. background.js:

   Kollar efter brister och initierar eventlisteners för att fånga upp HTTP-trafik och eventuellt även blockera HTTP-trafik. Lagrar alla responser som tas emot men endast den senaste förfrågan. Detta kan leda till att tillägget visar felaktigt antal brister samt att analyseringsverktyget och rapportverktyget inte visar rätt/förväntat resultat.

4. report.js:

   Kollar efter brister och genererar en rapport över vilka säkerhetsbrister som hittats sorterad på domän. Funktionen som upptäcker hot finns även i denna fil då denna funktion körs överallt där hot listas (hot sparas inte, endast responser och förfrågningar), denna funktion var inte densamma som i exempelvis background.js. Vidare saknades användning av funktionen som upptäcker referer token leakage i denna fil och kan därav inte ge en komplett säkerhetsrapport.

Testkörning av verktyget utan modifikation visade att responser inte uppfattades för facebook med nuvarande implementation. Vidare fanns problem med att extrahera referer ur headern vilket orsakar CSRF, detta var i huvudsak beroende på att denna funktionalitet numera kräver extra konfiguration med tillägget ”extraHeaders” (från Chrome version 72)[45].

Efter att ”extraHeaders” definierats upptäcktes inga CSRF felaktigt av denna anledning vid användning av Google som IdP. En egen version av funktionen som upptäcker om det är en OAuth2 respons lades till som fungerar för mer generella OAuth2-baserade autentiseringslager, däremot är identitetsleverantören hårdkodad till Facebook för denna. Denna funktion kördes parallellt med koll om det är Google som används som IdP och träder i kraft om det visar sig att det inte var Google som var IdP (se bild 6).

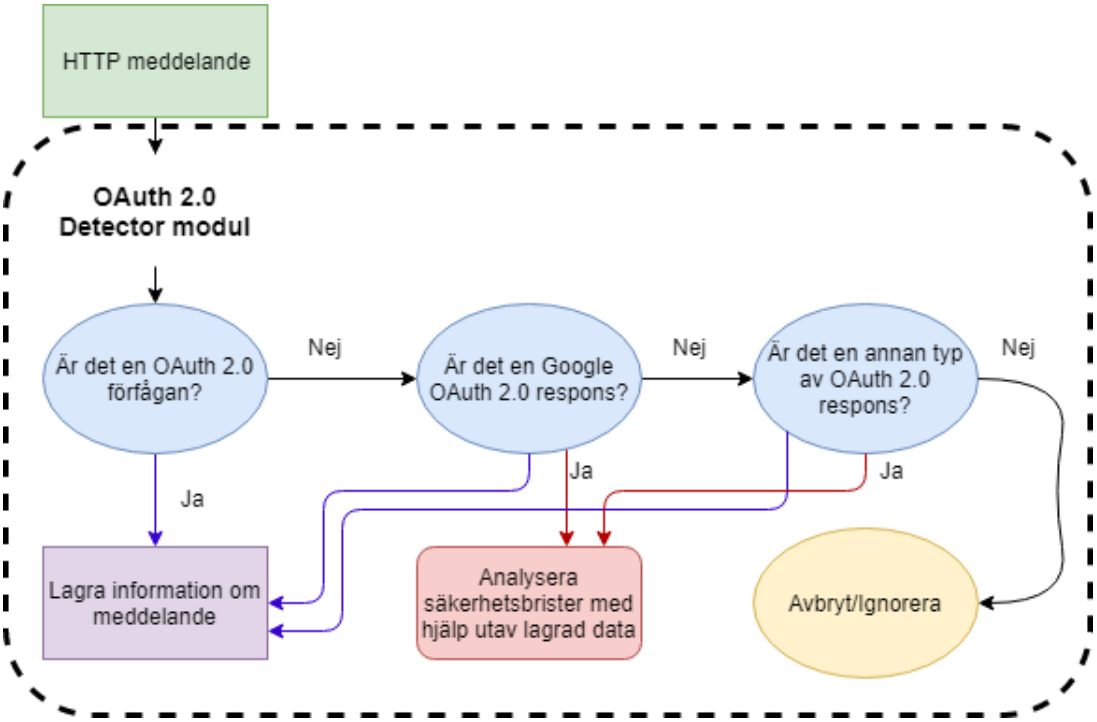


Bild 6: OAuth 2.0 Detector modulens flöde efter implementation av igenkännande av mer generella OAuth 2.0 responser

Analyseringsverktyget är mycket oförutsägbart vad gäller användargränssnitt och tar inte bort tabeller som önskat, vid sökande av flera domäner i rad blir denna information mycket uppblandad då tabellerna fylls på successivt och inte heller försvinner helt vid tryck på ”clear” (se Bild 7).

45 https://developer.chrome.com/extensions/webRequest

Bild 7: Exempel på tabeller som inte rensas bort

Vidare paras inte rätt request ihop med rätt response om flera inloggningsförsök gjorts för samma klient, den första responsen som gjordes mot klienten kommer alltid visas mot den senaste requesten som gjorts (äldsta med nyaste) vilket visar felaktiga resultat, tillägget fungerar som förväntat om endast ett inloggningsförsök görs per klientsida. Vidare blockeras inte CSRF hot som beror på avsaknaden av state parametern då denna variabel endast används vid utskrift i konsol och vid användning av rapporteringsverktyget, detta kan vara så som det är tänkt men enligt studien framgår det att det räknas som en CSRF attack och att alla sådana attacker ska blockeras. Ett antal funktioner finns i ett flertal filer där bland annat funktionen ”detectOAuth2Threats” existerar i två versioner, i detta fall leder detta till att report-verktyget inte upptäcker samma hot som från början upptäckts. Detta fixades genom att ta den senaste versionen (den i background.js) och ersätta den utdaterade versionen i report.js. Ett problem som hittades vid vidare undersökning av kod är att Authorization Flow misuse och Impersonation attacks inte kommer gå att implementera för Facebook på grund utav avsaknad av metod att extrahera Facebook-tokens från POST data (inget regex[46]) samt att Facebook SSO helt saknar en id_token.

Efter detta gjordes en första testkörning med de minimala förändringarna på de 100 utvalda klienterna med Facebook som inloggningsval. Detta visade att brister finns men även att tillägget stöter på flera problem som ger felmeddelanden, detta kan leda till/betyda att tillägget inte fungerar som tänkt och behöver åtgärdas för att säkerställa att tillägget fungerar så bra som möjligt.

Första felet som åtgärdades var att kolla om en variabel/objekt ”data” samt ”rawData” var definierad innan dessa variabler användes i funktionen detectOAuth2response i background.js. Sedan fanns ett problem i funktionen detectOAuth2 där host inte kunde avläsas från objektet RPUrl, detta beror delvis på att funktionen som används för att kolla om en sträng är en URL inte alltid returnerar förväntat svar. Detta är svårt att få till så det fungerar som förväntat för alla URL:er, valet av lösning som fanns implementerat är att

46 https://en.wikipedia.org/wiki/Regular_expression

använda regex. En uppdatering av denna funktion har utförts så att den accepterar alla strängar som går igenom konstruktion av klassen URL[47].

Vid andra testkörningen uppkom då inga felmeddelanden. Noterat är däremot att third party token leakage varierar i antal och kan behövas väntas in någon sekund från inloggningsförsök för att de ska registreras. En annan sak som påverkar resultat av hur många token leakage som finns på sidorna är huruvida man accepterar användning av riktad reklam och så vidare. En annan notis är att Facebook sparar information i cookies och att det kan vara värt att ta bort denna cookie mellan inloggningsförsök för att alla inloggningsförsök ska gå igenom samma steg. Vidare kan inte data/brister som genererats efter att mer än ett försök av inloggning utförts anses vara pålitlig, då tillägget har odefinierat beteende för detta. En ny browser-session användes vid varje 100 genomsökta och då nollställdes även cookies.

De sidor rankade 101-200 testades för klienter med Google-SSO för att se om tillägget kunde upptäcka det den tidigare kunde för Google. De andra mindre undersökningarna utfördes även här. Av dessa upptäcktes att spenderad tid på hemsidorna gjorde att antalet token leaks varierade.

Ny funktionalitet att upptäcka referer-token leakage för Facebook har implementerats genom att funktionen som upptäcker/identifierar generella OAuth 2.0 responser kördes parallellt med koll om det är Google som användes som IdP och träder i kraft om det visar sig att det inte var Google som var IdP. Efter testkörningarna bedömdes det även att för sista och slutgiltiga undersökningen ska varje sida ha en helt ny browser-session och vara utan cookies. Dubbletter av CSRF-attacker räknades bort för varje domän (om flera CSRF hot upptäckts för samma klient räknades endast detta som en).

En användarguide lades till från Tools-tabben i tillägget och kan nås vid tryck på knappen ”User Guide” (se Bild 8).

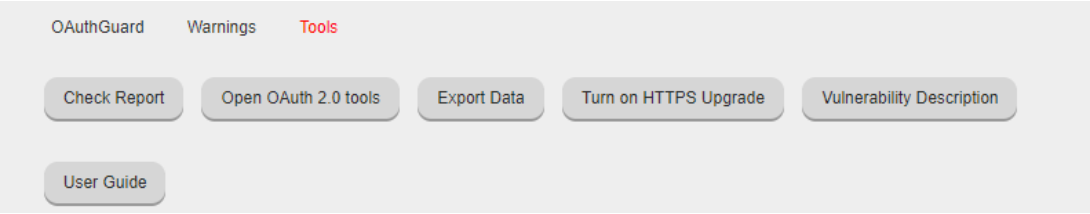


Bild 8: Användargränssnitt (Tools - tabben)

4.2 Funktionalitetstester

En testklient utvecklas för att upptäcka CSRF-brister vid avsaknad av state parameter samt för att kontrollera att tillägget upptäcker responser för Facebook SSO som den ska. De andra bristerna kontrollerades via avläsning av responser och förfrågningar för att bedöma om de stämmer med beskrivningen av hur brister ska upptäckas.

47 https://developer.mozilla.org/en-US/docs/Web/API/URL

4.2.1 Testklient

Testklienten skapades för att i huvudsak bestå av en service[48], en restcontroller[49] och ett frontend. Frontend utvecklades i första hand för att tillhandahålla ett formulär/form[50] (se bild 9).

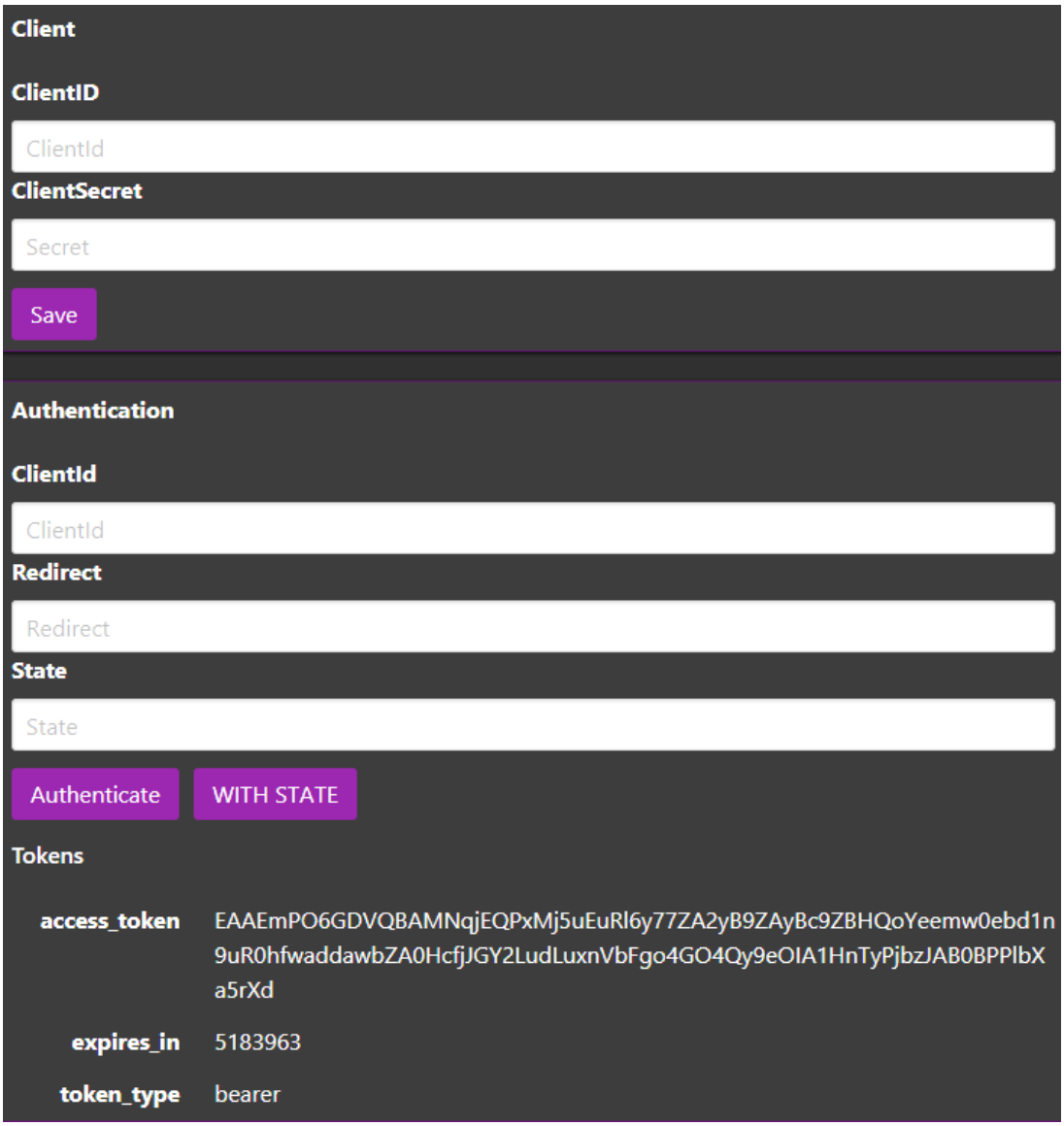


Bild 9: Användargränssnitt med information om access_token efter lyckad auktoriseringsprocess

48 https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm
49 https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html
50 https://www.w3schools.com/html/html_forms.asp

Detta formulär (se bild 9) används för att fylla i en client_secret, ett client_id, en redirect_uri (som även är definierad i Facebooks utvecklarportal [51] så att denna URI kan verifieras) samt ett state om så önskas. Vid tryck på knappen ”Authenticate” skickas denna data, med metoden GET, till Facebooks inloggnings-endpoint så att användaren får acceptera eller neka behörighet. I restcontrollern finns i huvudsak en funktion som är knuten till relativa URL:en /code som hanterar mottagande och konsumerande av code från Facebooks inloggningsendpoint. Denna URL är även definierad i inställningsavdelningen för applikationen hos Facebooks utvecklingsportal som en OAuth Redirect URI. Restcontrollern skickar sedan vidare eventuellt mottagen code till en service eller sätter ett felmeddelande om ingen code mottagits. Denna service är sedan den som skickar vidare code tillsammans med client_secret, client_id och redirect_uri till Facebooks token endpoint för att få en access_token. Om en code skickats med till denna service byggs denna code tillsammans med client_id, client_secret och redirect_uri ihop med hjälp utav en UriComeponentBuilder[52], detta lämnar utrymme att modifiera skickade parametrar. Denna builder har Facebooks token-endpoint som bas-URL. Den information som lagrats i denna builder används sedan för att bygga en sträng innehållande all information som behövs (bas-URL och parametrar) för att göra en förfrågan mot Facebooks token-endpoint. En RestTemplate[53] skickar iväg den uppbyggda URI-strängen och tar emot resultatet. Om den information som skickats till token endpointen är korrekt återfås en access_token som sparas undan och även visas i användargränssnittet (se bild 9) på testklienten. Bild 10 visar detta flöde (fungerar som Authorization Code flow).

51 https://developers.facebook.com

52 https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/util/UriComponentsBuilder.html

53 https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

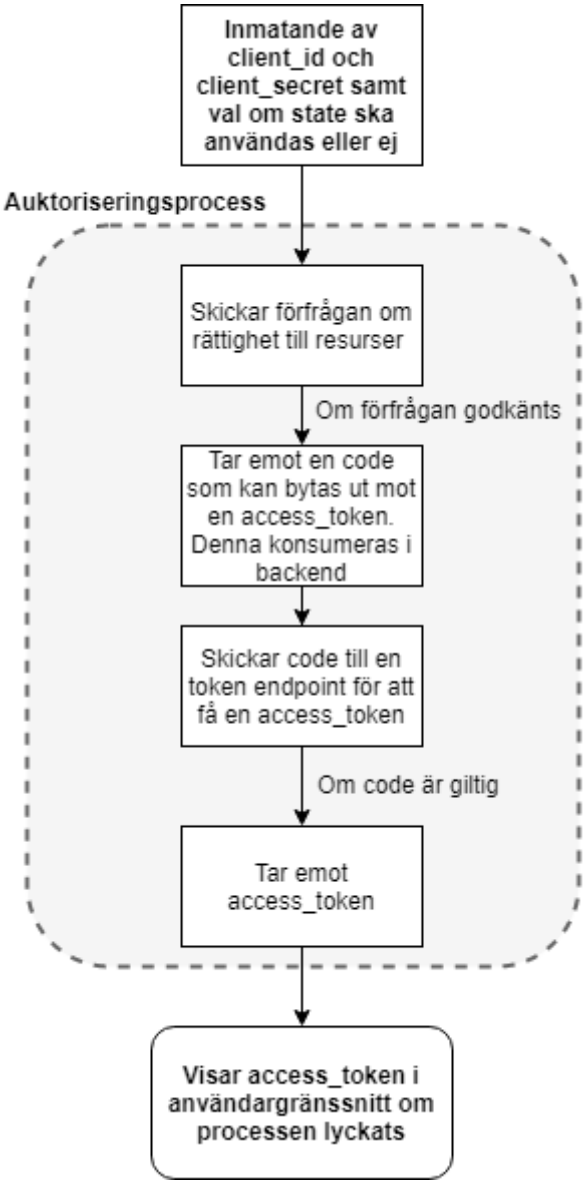


Bild 10: Testklientens flöde för hämtning av access_token

4.2.2 Övriga tester

De övriga bristerna som implementerats i OAuthGuard för Facebook SSO kontrollerades manuellt genom att avläsa lagrad data (se bild 12) från inloggningsförsök. Dessa kontrollerades i huvudsak efter notis om att en påstådd brist upptäckts från Chrome konsolen (se bild 11) men stickprov på hemsidor som inte visat brister utfördes även för att se så att inte brister smiter igenom tilläggets kontroller.

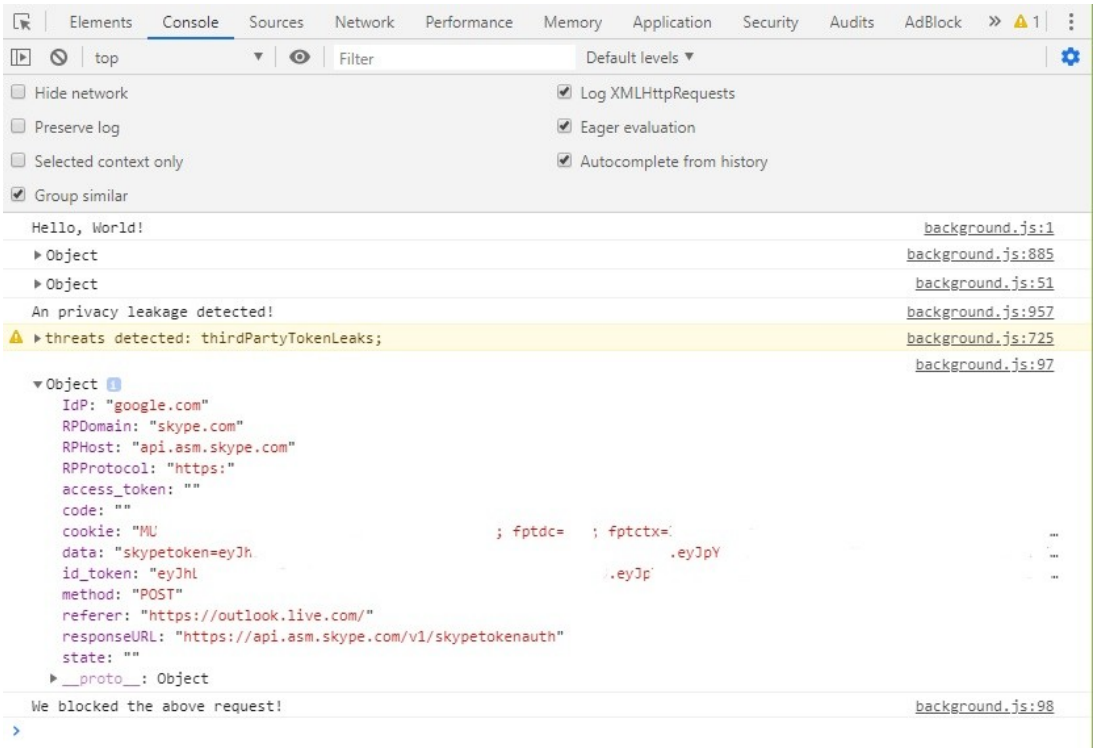


Bild 11: Vy för Chrome konsolen som användes för att identifiera och analysera meddelanden som anses skadlig av OAuthGuard

De brister som kontrollerades var Referer token leaks, CSRF med hjälp utav att avläsa referer header, third party token leaks samt till viss grad unsafe token transfer / osäker anslutning. Dessa brister kontrollerades på samma sätt som Vulnerability Analyser modulen i OAuthGuard är tänkt att upptäcka dem fast manuellt (vidare information finns under kapitel 2.5). Nedan listas testkriterierna som även inkluderar information om hur Referer token leakage identifieras samt om hur Third party token leakage identifieras:

- Om Referer headern till OAuth 2.0 responsen inte pekar till den IdP eller RP domän som använts ska detta räknas som CSRF.
- Om HTTP använts istället för HTTPS vid kommunikation ska detta räknas detta som unsafe token transfer.
- Om ingen motsvarande request hittas för en mottagen response ska detta anses vara en third party token leak (intentional privacy leak).
- Om en ”code” eller ”access_token” finns i referer headern så jämförs domänen i referer headern och tillhörande förfrågan, är domänerna olika ska detta räknas som en referer token leakage (unintentional privacy leak).

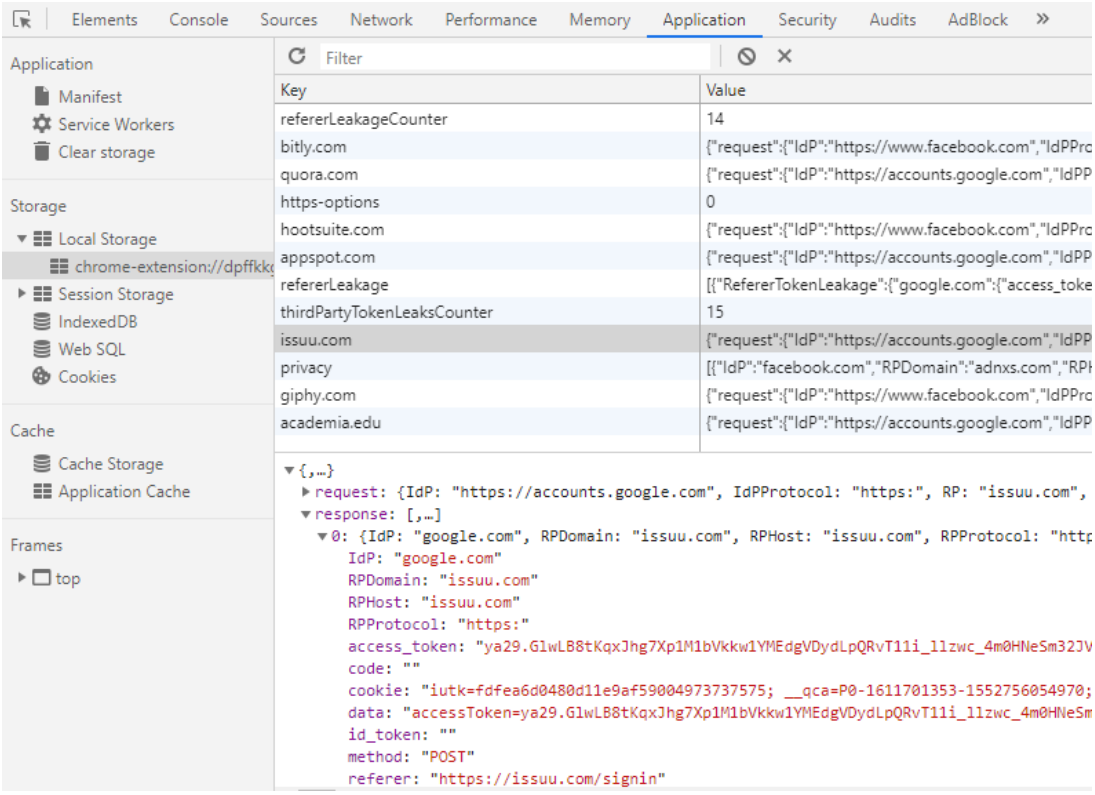


Bild 12: Vy över local storage som använts för att manuellt avläsa sparade responser och förfrågningar

Alla typer av hot som utökningen av OAuthGuard behandlade klarade tillägget av att, till viss grad, upptäcka och ingen större testning behövdes därav göras. Det finns däremot brister i framförallt hur identitetsleverantörer identifieras och hur referer token leakage identifieras, dessa brister gör att tillägget inte fungerar optimalt för en så stor mängd hot som möjligt. Bristen för referer token leakage är att parametrarna i form utav exempelvis code inte går efter ett regex utan efter parameternamn, skulle klienten läcka en sådan code under ett annat parameternamn vid användning av Facebook SSO skulle tillägget inte märka detta. Vad gäller lagring och identifiering av identitetsleverantör sköts även detta med regex för Google SSO, något som resulterar i att sådana regex måste konstrueras för varje individuell identitetsleverantör för optimal prestation.

5 Resultat

Kapitlet listar resultat av konstruktion och undersökning.Delkapitel 5.1 redovisar resultat av konstruktion, delkapitel 5.2 redogör resultat av de funktionalitetstest som utförts. Delkapitel 5.3 – 5.5 redovisar resultat av undersökningarna som utförts med OAuthGuard.

OAuthGuard har vidareutvecklats för att även stödja Facebook SSO. En testklient med Facebook SSO har även utvecklats för att testa funktionalitet i utökningen av OAuthGuard. För undersökningen har de 500 populäraste webbsidorna enligt majestic[54] undersökts, av dessa hade 106 domäner stöd för Facebook SSO, men av dessa var 8 (ungefär 7.5%) riktad till samma webbsida med olika domännamn så endast 98 stycken webbsidor med Facebook SSO har räknats till undersökningen. Dessa 8 dubbletter av domäner var jämnt fördelade över de 500 webbsidorna som undersöktes och hade ingen större koncentration till någon speciell popularitets-rank.

5.1 OAuthGuard

Den nuvarande versionen av OAuthGuard har bibehållit sin funktion för Google SSO så som den var vid tidigare utgåva med enda undantaget att extraHeaders definierats så att referer headern går att avläsa efter Chrome version 72. Den utökade funktionaliteten för Facebook SSO innebar i huvudsak utökning så att tillägget klarade av att spara och läsa responser som inte följer Googles struktur utan även Facebooks/andra OAuth 2.0 baserade autentiseringar. De brister som kan behandlas för Facebook SSO i OAuthGuard efter utökning är följande:

- CSRF
- Osäker anslutning, Facebook skyddar numera själv mot denna typ av hot och accepterar inte osäkra anslutningar men tillägget behåller ändå detta skydd även för Facebook.
- Third party token leakage
- Referer token leakage

Utseendet av tillägget har ej förändrats förutom tillägget av en guide i form utav en knapp under tabben tools som öppnar en ny sida med instruktion över vad knappar och dylikt gör i användargränssnittet. Tillägget i sitt utökade tillstånd finns tillgänglig på GitHub [55].

5.2 Funktionalitetstest och testklient

En testklient skapades för att testa funktionalitet av utökningen av tillägget. Testklienten utvecklades med Java Springboot för backend och Angular för frontend med manuell hantering av parametrar/flöde (ej SDK). De funktioner som implementerats i denna klient för att testa brister är följande:

54 https://majestic.com/reports/majestic-million
55 https://github.com/Mothlym/OAuthGuard

- CSRF: går att med ett knapptryck välja om state ska skickas med eller inte från klienten.

5.3 Testkörningar

Två stycken testkörningar utfördes i utvecklingsprocessen av tillägget. Dessa testkörningar hjälpte till att forma metoden som användes för slutgiltiga undersökningen samt bidrog med information om hur tillägget fungerade. Undersökningen om de 500 utvalda hemsidorna delades upp i fem grupper rankad efter popularitet. Dessa grupper visas efter X-axeln på graferna.

5.3.1 Testkörning 1

Den första testkörningen utfördes efter tillägget genomgått minimala förändringar, endast funktionalitet för att Facebook SSO responser ska upptäckas samt för att referer headers ska kunna avläsas är tillagd. Denna testkörning använde en version av OAuthGuard som fortfarande kan generera felmeddelanden som fanns innan tillägget började vidareutvecklas.

Rank (grupp)	0-100	101-200	201-300	301-400	401-500
Antal med Facebook SSO	19	23	24	16	16

Tabell 3: Visar hur många klienter som stödjer Facebook SSO per grupp för första testkörningen.

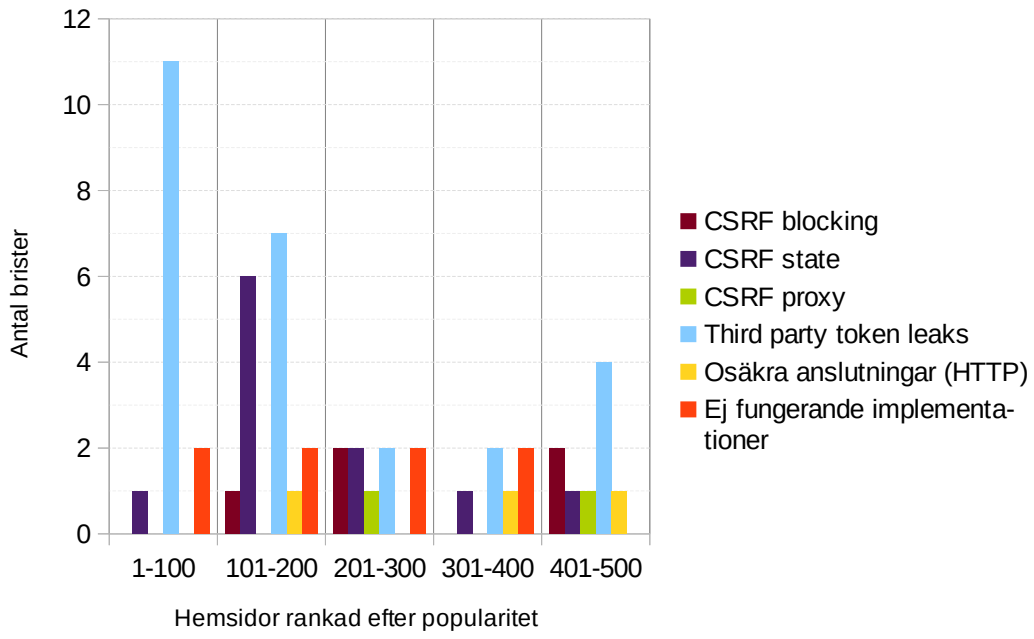


Bild 13: Visar antalet attacker samt attacktyp klienten var sårbar för, uppdelad efter popularitet

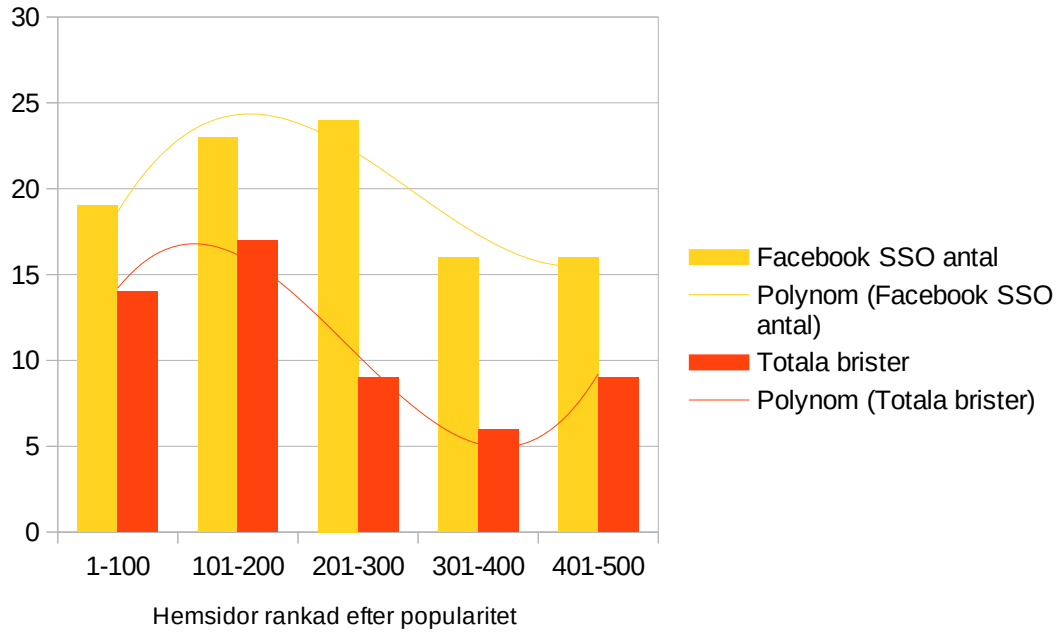


Bild 14: Visar antalet klienter i varje grupp som stödjer Facebook SSO samt hur många totala brister som upptäckts för varje grupp.

Totalt blockerades fem meddelanden för CSRF och ytterligare elva CSRF varningar (mot avsaknad av state parameter) varnades för. Vidare förekom två blockeringar för CSRF som beror på användning av proxy (mot kinja.com) Totalt förekom 26 stycken third party token leaks, tre klienter hade osäkra anslutningar och totalt åtta implementationer fungerade inte Facebook SSO för alls. Se nedanstående tabell.

Brist/Hot	Antal
CSRF	5
CSRF-varning (avsaknad av state-parameter)	11
CSRF på grund utav proxy	2
Third party token leak	26
Osäkra anslutningar	3
Ej fungerande implementationer	8
Tabell 4: Visar vilka hot som upptäckts under första testkörningen samt hur många av dessa som upptäckts	

5.3.2 Testkörning 2

En andra testkörning av tillägget utfördes då alla delar av koden som genererat felmeddelanden ändrats så de inte längre genererar felmeddelanden. Vidare ändrades även undersökningsmetoden så att browser-sessionen och tilläggets lagrade data togs bort var hundrade undersökta hemsida (en gång innan ny grupp undersöks).

Rank (grupp)	0-100	101-200	201-300	301-400	401-500
Antal med Facebook SSO	19	23	24	16	16
Antal klienter med Facebook SSO som hade brister	3 (~15,8%)	8 (~34,8%)	7 (~29,2%)	4 (25%)	3 (18,75%)

Tabell 5: Visar hur många klienter som stödjer Facebook SSO per grupp samt hur många av dessa som hade någon form utav brist för andra testkörningen.

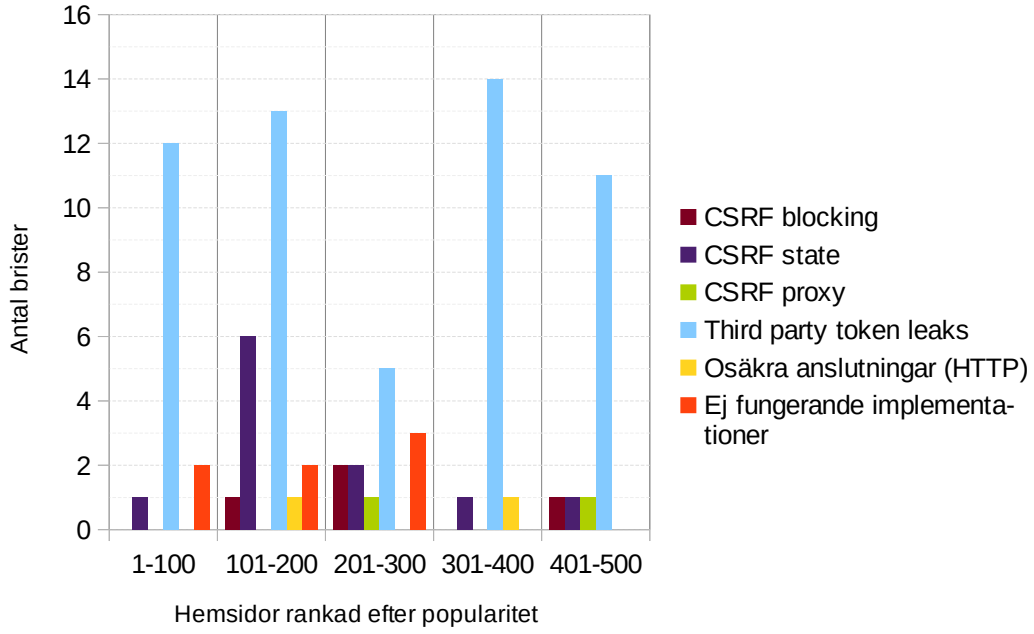


Bild 15: Visar antalet attacker samt attacktyp klienten var sårbar för, uppdelad efter popularitet

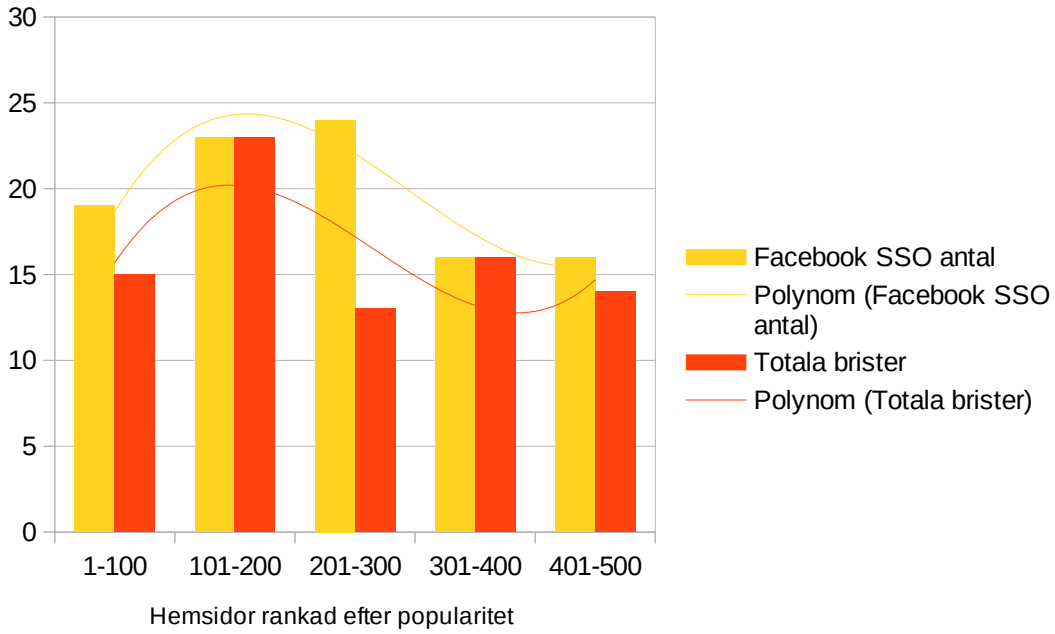


Bild 16: Visar antalet klienter i varje grupp som stödjer Facebook SSO samt hur många totala brister som upptäckts för varje grupp.

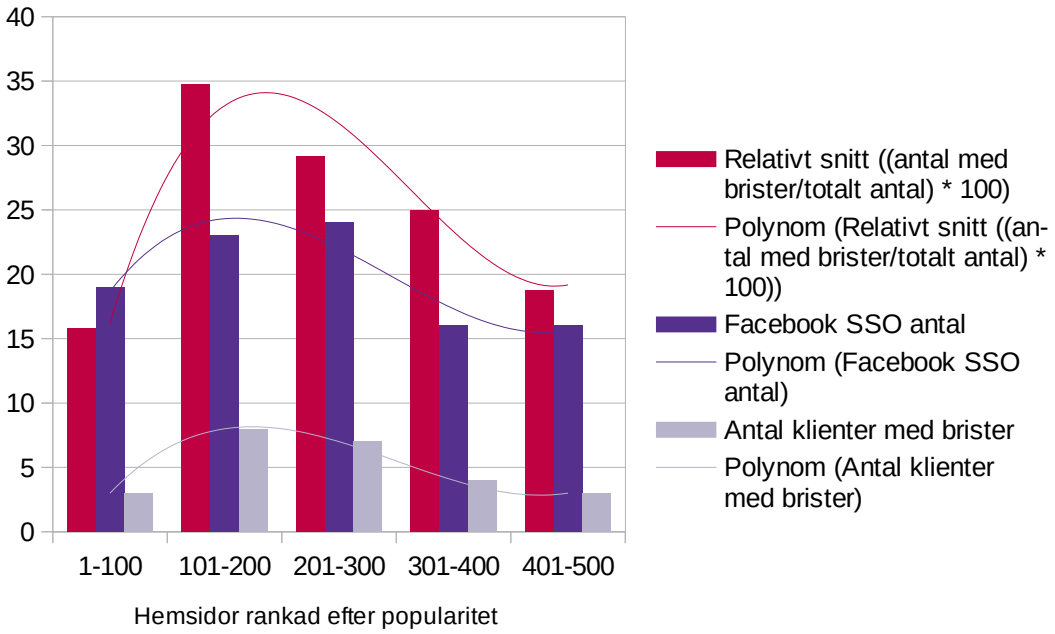


Bild 17: Visar i första hand ett relativt snitt för hur många brister som finns i förhållande till antalet klienter som implementerat Facebook SSO. Den visar även hur många klienter per grupp som stödjer Facebook SSO samt hur många klienter av dessa som har brister.

Totalt hade 25/98 klienter från denna undersökning brister.

Brist/Hot	Antal
CSRF	4
CSRF-varning (avsaknad av state-parameter)	11
CSRF på grund utav proxy	2
Third party token leak	55
Osäkra anslutningar	2
Ej fungerande implementationer	7
Tabell 6: Visar vilka hot som upptäckts under andra testkörningen samt hur många av dessa som upptäckts	

5.4 Mindre undersökningar

Ett antal mindre studier utfördes på gruppen av klienter med rank 101-200 för referensdata. Dessa undersökningars resultat listas nedan. Undersökningen om de 500 utvalda hemsidorna delades upp i fem grupper rankad efter popularitet. Dessa grupper visas efter X-axeln på graferna.

Rank (grupp)	101-200
Antal med Facebook SSO	23
Tabell 7: Visar hur många klienter som stödjer Facebook SSO och vilken rank de tillhör för de mindre testkörningarna.	

5.4.1 Kvantitetsundersökning

Genomsökning av den näst mest populära gruppen, utförd tre gånger i rad med samma metod, med Facebook (rank 101-200) utan dubbletter av CSRF. Tabellen nedan visar resultatet av denna undersökning.

	Första	**Andra**	**Tredje**
CSRF	1	1	1
CSRF (saknar state)	5 (6)	5 (6)	5 (6)
Trasiga implementationer	2	2	2
Osäker anslutning	1	1	1
Third party Token leakage	13	9	7

Totalt antal klienter med brister	8	9	9

Tabell 8: Visar den omgång som undersökts och hur många av respektive attack som hittats per omgång.

5.4.2 Undersökning nekande till reklam/cookies

Genomsökning av den näst mest populära gruppen med Facebook (rank 101-200) utan att acceptera riktad reklam med mera. Totalt undersöktes 23 stycken klienter med Facebook SSO och 10 av dessa hade någon form utav brist (ungefär 43,5%). Vidare gick det inte att göra några inställningar för cookies eller dylikt på 8 av de 23 undersökta sidorna, den enda sidan som hade third party token leaks där det inte går att tacka nej till användning av diverse cookies var ebay.com. Tabellen nedan listar antalet hot som hittats.

Rank (grupp)	CSRF state	Third party token leaks	Osäker anslutning	Trasig implementation	CSRF
101-200	5	8	1	2	0

Tabell 9: Visar den grupp som undersökts och hur många av respektive attack som hittats i denna grupp.

5.4.3 Google undersökning

Genomsökning av den näst mest populära gruppen med Google (rank 101-200), totalt undersöktes 18 stycken Google SSO klienter i denna grupp varav nio av dessa (50%) hade någon form utav brist. Tabellen nedan listar antalet hot som hittats.

Rank (grupp)	CSRF state	Third party token leaks	Referer token leakage	Osäker anslutning	Flow misuse	Impersonation attack	CSRF
101-200	5	14	50	1	1	1	1

Tabell 10: Visar den grupp som undersökts och hur många av respektive attack som hittats i denna grupp.

Värt att notera är att Facebook själv blockerade den osäkra anslutningen (digg) medan den gick igenom för Google. Yelp fungerade även med Google, vilket den inte gjorde för Facebook. För grupp rank 0-100 fanns det 25 stycken med Google SSO, för grupp med rank 101-200 fanns det 18 stycken, för grupp med rank 201-300 fanns det 22 stycken, för grupp med rank 301-400 fanns det 13 stycken och slutligen för grupp 401-500 fanns det 17 stycken med Google SSO. Totalt i de 500 populäraste fanns det 95 klienter med stöd för Google SSO (ej inräknat domäner som är direkt anknuten till Google där det endast går att logga in med Google, med dessa är det några fler).

5.5 Slutgiltig undersökning

Undersökningen om de 500 utvalda hemsidorna delades upp i fem grupper rankad efter popularitet. Dessa grupper visas efter X-axeln på graferna.

Rank (grupp)	0-100	101-200	201-300	301-400	401-500
Antal med Facebook SSO	19	23	24	16	16
Antal klienter med Facebook SSO som hade brister	4 (~21,05%)	11 (~47,8%)	8 (~33,32%)	6 (37,5%)	3 (18,75%)

Tabell 11: Visar hur många klienter som stödjer Facebook SSO per grupp samt hur många av dessa som hade någon form utav brist i den slutgiltiga undersökningen.

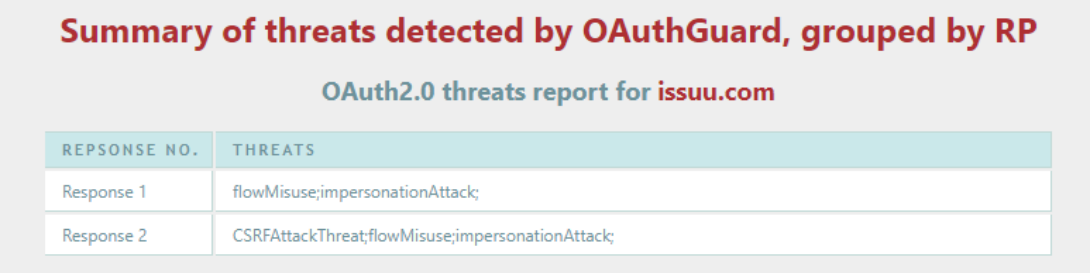
Rank (grupp)	**CSRF state**	**Third party token leaks**	**Referer token leakage**	**Osäker anslutning**	**Trasig implementation**	**Totalt antal brister**
0-100	1	17	0	0	2	18
101-200	5	9	1	1	2	16
201-300	2	5	14	0	3	22
301-400	1	9	0	1	1	11
401-500	2	0	8	0	0	10

Tabell 12: Visar de grupper som undersökts och hur många av respektive attack som hittats i dessa grupper samt hur många totala brister som hittats per grupp.

26/98 (ungefär 26.5%) har någon form utav brist och 11/98 saknar state parametern vid inloggningsflöde och är därav potentiellt sårbar för CSRF. Av de 26 stycken med brister registrerades det att 18 stycken hade third party token leaks samt att fem stycken hade referer token leaks.

Totalt antal med brister inkluderar inte trasiga implementationer. Alla third party token leaks var till domänen adnxs.com. Referer token leakage checken för Facebook använder inte regex och kan endast märka läckage av parametern ”code”, så den upptäcker inte alla. Ett exempel på sida som tidigare visade CSRF med blockering är cbsnews.com, men med ny browser-session är inte detta ett problem, därav inga blockerande CSRF i slutgiltiga undersökningen.

Några stickprov för Google SSO utfördes och visade att hemsidor som bland annat issuu.com hade stora säkerhetsproblem (medan motsvarande Facebook-implementationer var enligt tillägget OAuthGuard säkra) där issuu.com fortfarande visade att den var mottaglig för impersonation attacks (se bild 18).



Summary of threats detected by OAuthGuard, grouped by RP

OAuth2.0 threats report for issuu.com

REPSONSE NO.	THREATS
Response 1	flowMisuse;impersonationAttack;
Response 2	CSRFAttackThreat;flowMisuse;impersonationAttack;

Bild 18: Rapport för issuu.com genererad av OAuthGuard

De sidor som tidigare visat CSRF för användning av proxy vid inloggning ändrades innan den slutgiltiga undersökningen utfördes och registreras inte längre som CSRF hot.

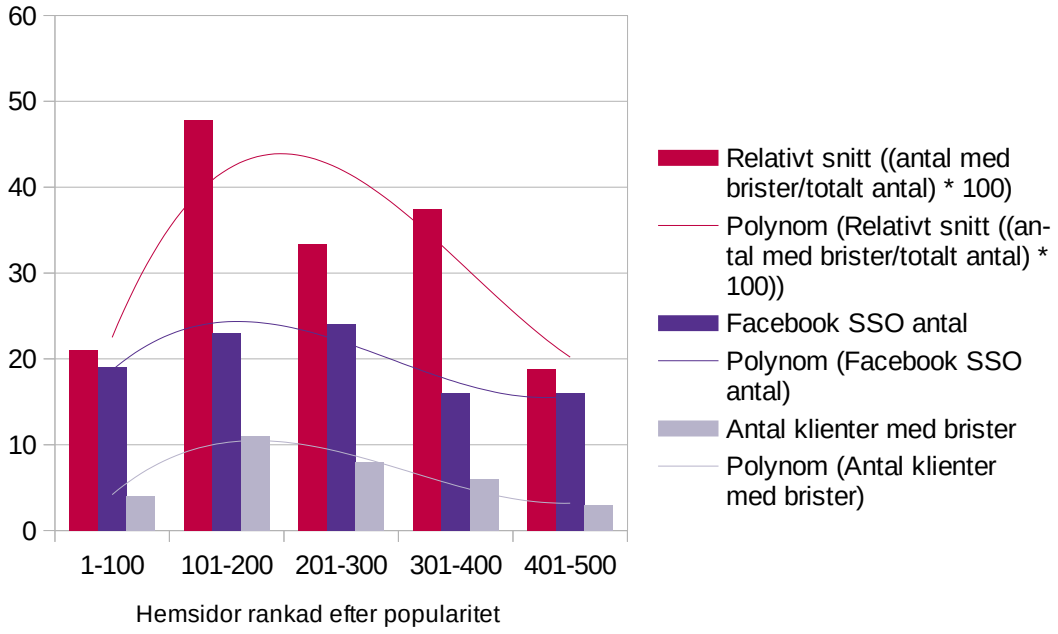


Bild 19: Visar i första hand ett relativt snitt för hur många brister som finns i förhållande till antalet klienter som implementerat Facebook SSO. Den visar även hur många klienter per grupp som stödjer Facebook SSO samt hur många klienter av dessa som har brister.

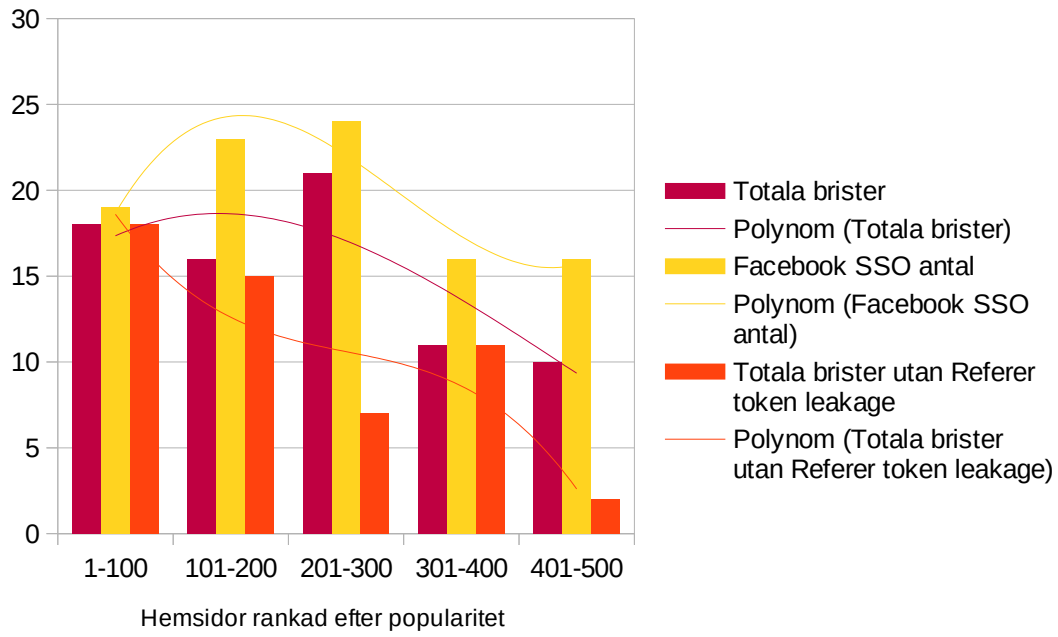


Bild 20: Visar antalet klienter i varje grupp som stödjer Facebook SSO samt hur många totala brister som upptäckts för varje grupp. Det visas staplar både för antalet brister med och utan referer token leakage inräknad.

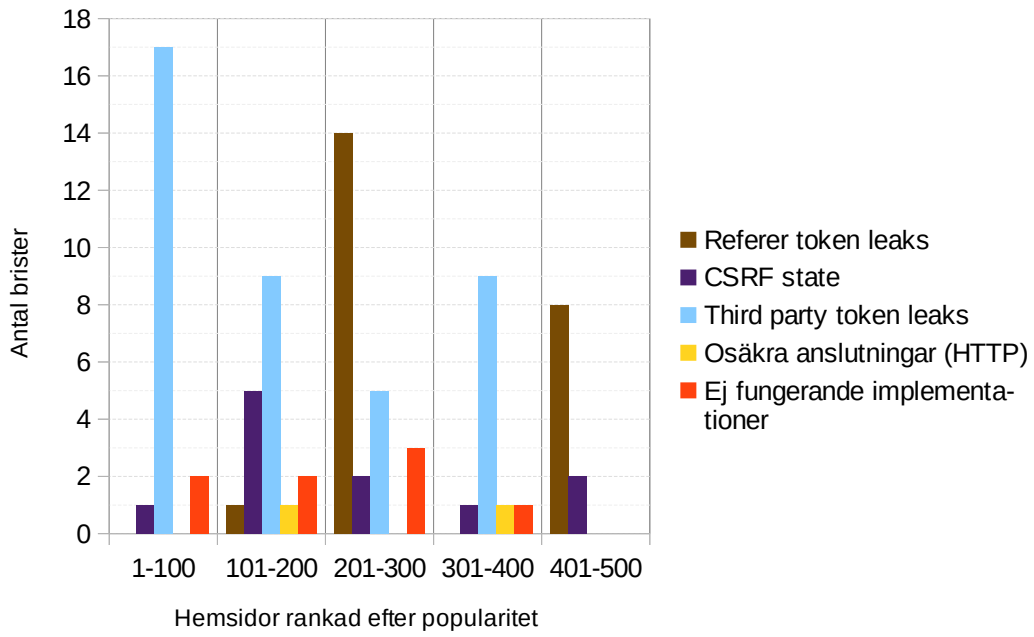


Bild 21: Visar antalet attacker samt attacktyp klienten var sårbar för, uppdelad efter popularitet

6 Diskussion

Under detta kapitel redovisas först en generell diskussion över arbetet där oväntade och väntade begränsningar nämns tillsammans med reflektion över prioriteringar. I delkapitel 6.1 diskuteras implementationen. Delkapitel 6.2 diskuterar funktionalitetstesterna som utförts. Delkapitel 6.3 behandlar och diskuterar resultat av testkörningar och dess eventuella innebörder medan delkapitel 6.4 jämför dessa resultat och presenterar svar på forskningsfrågorna. Slutligen presenterar delkapitel 6.5 de etiska aspekterna.

Arbetet med OAuthGuard har varit komplicerat på ett antal fronter, innan valet att endast göra minimala förändringar till tillägget gjordes hade en utvecklingsprocess påbörjats för att strukturera om verktyget och göra det mer användarvänligt. I denna process upptäcktes det att för många fundamentala förändringar behövdes göras och att den ursprungliga funktionen inte längre kunde garanteras då mindre förändringar i kod kunde ändra hela tilläggets funktion. I brist på tid och den djupa förståelse som behövs för att garantera att tillägget fungerar korrekt bestämdes det att endast minimala förändringar skulle utföras så att undersökningarna garanterat kan producera resultat på samma sätt som tidigare studie [19], något som är nödvändigt för att jämföra data. Dessa minimala förändringar hade då redan identifierats från experimentering och försök till strukturering av OAuthGuard och den nya versionen gick då snabbt att utveckla. Studien fick i och med detta oväntat mindre fokus på att vidareutveckla verktyget. Hade behovet av regex upptäckts tidigare hade en större ansträngning för att hitta en lösning på att hantera de andra säkerhetshoten även för Facebook SSO-klienter kunnat utföras och ett regex eller annan metod för att hantera detta hade kanske kunna hittats. Att detta inte hittades tidigare kan bero på att fokus lades på att påbörja strukturering, experimentering och att åtgärda brister i koden istället för att endast fokusera på att strukturera upp och planera hur brister ska upptäckas och hanteras. Detta borde ha prioriterats annorlunda redan från början och endast fokusera på att tillägget ska fungera för undersökningssyfte/empiriska studien. Detta hade kunna möjliggöra att fler flöden än Authorization Code flow hade kunna tillämpats.

6.1 Implementationsbrister och svårigheter

Till att börja med följer Facebook inte samma standard som Google vid SSO vilket gör att typer av tokens och möjliga parametervärden/flöden varierar vilket i sin tur försvårar processen att översätta hot som existerar för Google SSO till de hot som finns för Facebook SSO. Av denna anledning utelämnades behandling av Authorization flow misuse attack då Facebook SSO inte har id_tokens och då studien i första hand fokuserat på Authorization code flow och därav inte behandlar möjligheten till att så många tokens innehas av klienten. Begränsningen till att endast undersöka Authorization code flow grundas i svårigheten att lokalisera och hantera access_tokens, detta utfördes i tidigare version av OAuthGuard med ett regex men inget bra regex för att extrahera access_tokens vid användning av Facebook SSO hittades inom en rimlig tid. Alternativet hade varit att leta efter ett nytt sätt att utföra detta på, men det hade resulterat i större avvikelser från grundimplementationen. Vidare förlitade sig även verktyget OAuthGuard i sin ursprungliga skick på att kunna identifiera vilken identitetsleverantör som använts genom att använda regex på tokens som skickats med för att sedan använda denna information vid koll av exempelvis CSRF. Svårigheten av att skapa regex som passar alla typer av tokens för en stor mängd identitetsleverantörer gör

att en mer dynamisk lösning är önskvärd och detta är något som bör implementeras i verktyget. En mer dynamisk lösning är även önskvärd vad gäller hantering av funktion som diverse whitelists nu står för. Dessa listor borde i minst kunna utökas på något dynamiskt sätt av användaren så att exempelvis hemsidor som inte uppmärksammats använda proxy vid inloggning kan läggas till i whitelist om användaren litar på dem. Vid undersökning av kod och studie förekommer en del tvetydigheter om vad som är en respons och vad som är en request dessa begrepp blandas, framförallt vad gäller varibelnamn. Att hitta andra metoder att extrahera access_tokens på, att hantera whitelists samt för att identifiera identitetsleverantörer är något som bör undersökas framöver, förhoppningsvis kan en metod som är mer dynamisk och direkt användbar för att stödja flera identitetsleverantörer arbetas fram.

Strukturen för lagring och extraherande av lagrad data skulle behöva struktureras om för att en del brister ska kunna lösas, exempelvis problemet med att om flera förfrågningar och responser gjorts mot samma klient och någon vill analysera senaste respons/request i analyseringsverktyget så kommer senaste request visas tillsammans med första responsen. Vidare analyseras alla sparade responser och requests på nytt när hot ska skrivas ut, detta kan ändras så att de hot som upptäckts (även privacy-hot som nu sparas men separat) sparas i samband med tillhörande request/response. Detta skulle även kunna vara en lösning på problemet med att samma funktioner för att upptäcka hot finns i flera scriptfiler och därav måste uppdateras/exekveras likadant på samtliga ställen för att tillägget ska vara konsistent i vilka hot som upptäcks. I nuvarande implementation och ursprungliga implementationen används inte refererTokenLeakage funktionen eller privacy leakage hantering på samma ställe som där alla andra hot upptäcks, det används alltså inte konsekvent i alla filer och bör kontrolleras/ändras på så att ett så korrekt beteende av tillägget som möjligt kan garanteras. Slutligen behöver en bättre struktur av hela tillägget tillämpas för att öka möjligheter för vidareutveckling och hanterbarhet då det i sitt nuvarande skick är svårt att följa och många dubbletter av kod existerar över flera filer. Hur en bra struktur för tillägg som fungerar för flertalet identitetsleverantörer är upp till framtida studier att undersöka.

6.2 Funktionalitetstest och testklient

All funktionalitet som utökats har testats. På grund utav tidsbrist har inte testklienten utvecklats och använts för att kontrollera alla brister OAuthGuard har utan användes endast för att kolla om state parameter saknas eller inte i inloggningsflödet för CSRF. De andra bristerna, osäker anslutning och privacy-leak genom third party token leakage, har däremot testats genom testundersökningarna och analysering av dess trafik jämfört med vad tillägget rapporterat att den blockerat och varför. Denna metod av testning var inte lika konkret och krävde mer manuell undersökning men gick att utföra under en mindre tidsram vilket hade varit svårt för utvecklandet av en robust testklient. Vidare har denna metod av testande en större felmarginal då det är lättare att förbise brister, för vidare studier och utveckling är det att rekommendera att en fullständig testklient utvecklas och/eller att utvecklarverktyg /tydligare redovisning av data som manuellt kan kontrolleras implementeras i tillägget.

6.3 Undersökningar

Ett antal undersökningar med olika metoder har utförts med verktyget för att undersöka variationer i tilläggets beteende, som delvis använts i utvecklingsprocessen av tillägget för att bedöma vilka ändringar som behöver göras, men även för att förfina undersökningsmetod för den slutgiltiga undersökningen.

6.3.1 Testkörningar

Inför första testkörningen gjordes ett antal minimala förändringar för att tillägget skulle fungera som det gjorde innan samt för att responser som inte tillhör Google skulle kunna registreras. Efter denna ändring fungerade tillägget men fel/felmeddelanden som även genererades innan ändringar kvarstod. Resultatet av denna testkörning jämfört med den andra testkörningen som gjordes när felen där felmeddelanden uppkommit åtgärdats samt då browser-sessionen med tillhörande cookies och lagrad data för tillägget återställs för varje grupp skiljer sig. Färre CSRF blockeringar skedde i andra testkörningen men betydligt fler third party token leaks observerades. Så överlag ökade antalet brister per grupp för andra testkörningen. Det var även i detta skede det upptäcktes att rensande av browser-session, tilläggets data och cookies hade påverkan på hur tillägget uppfattade säkerhetsbrister, som resulterade i den utvalda metoden för slutgiltiga undersökningen.

6.3.2 Kvantitetsundersökning

Skillnader i data (se Tabell 8) mellan dessa försök kan ha ett antal förklaringar, till att börja med kan antalet sidor som har brister variera då tiden spenderad på sidorna kan ha varierat och därmed fångat upp olika många third party token leakage, eller inga alls. Av samma anledning kan antalet token leaks variera då flera requests kan hinna göras om användaren stannar en längre tid på en sida. Av denna anledning lades en detalj till för slutgiltiga undersökningens metod; att vänta till sidan helt laddat klart så tredjepartskod eller stora resurser hinner laddas in. Vidare fångades två responser upp för samma request och skapade därav dubbletter (antal med dubbletter listas inom parentes) av CSRF hot kopplad till avsaknad av state-parametern. Vad detta beror på är ännu inte känt men dessa dubbletter av CSRF-hot bestämdes för att räknas bort inför den slutgiltiga undersökningen.

6.3.3 Undersökning med nekande till reklam/cookies

Jämfört med andra testkörningen (se Tabell 6 och Bild 15) och samma grupp (rank 101-200) visar denna grupp färre brister totalt (se Tabell 8). Däremot har antalet klienter med brister ökat, detta kan bero på variation i hur sidorna använts vid inlogg, då inställningar för cookies tar extra lång tid att utföra jämfört med bara en vanlig inloggning ökar chansen/risken att upptäcka third party token leakage. Då många third party token leakages verkar vara kopplade till tredjeparter i form utav reklam är detta en trolig anledning till varför inte antalet token leaks ökat markant men att antalet sidor med bister ökat.

6.3.4 Google undersökning

Förutom de attacker som inte går att upptäcka för Facebook ser säkerhetsläget ganska lika ut men något sämre för Google än för Facebook (se Tabell 10 för Google SSO data och Tabell 12 för Facebook SSO data).

6.3.5 Slutgiltiga undersökningen

Den ändrade metoden att skapa en ny browser-session och rensa cookies resulterade i betydligt färre CSRF blockeringar så tidigare undersökningar hittat runt 5 medan slutgiltiga inte hittade en enda (se Tabell 12). Ett exempel på sådan sida som i slutändan inte visade sig ha blockerande CSRF var cbsnews.com. Trenden för CSRF beroende på state ändrades däremot inte (se Bild 21), vilket var förväntat då detta inte direkt påverkas av lagrad data. Några stickprov för Google visade att sidor som hade enligt tillägget säkra implementationer av Facebook SSO kunde ha stora problem med Google SSO, bland annat sidor som issuu.com (som nämndes i tidigare studie som problematisk).

6.3.6 Metod

Under arbetes gång genomfördes en mängd undersökningar med verktyget från olika stadier för att undersöka eventuella variationer i resultat så att en slutgiltig metod kunde arbetas fram efter analys. Variationer i resultat, både vid samma metod och version av tillägg samt vid olika val av metod och förändring i tillägget, indikerade att denna utforskande metod var nödvändig för att korrekt kunna positionera den framtagna informationens värde. För att förbättra metoden ytterliga hade fler identiska undersökningar i den slutgiltiga undersökningen för att få ett medelvärde om data fortfarande varierar. Då stor del av undersökningsprocessen innebär manuellt arbete (inloggning på hemsida, avläsning av data och sedan återställning av verktyg med mera) finns en felmarginal som gör att data kan variera. Om ett flertal undersökningar gjorts och ett medelvärde hade kunna dragits hade dessa siffror varit mer pålitliga då eventuella missar vid manuell hantering förmodligen inte kommer förekomma för exakt samma klienter flertalet gånger. Det är även värt att nämna att de sidor som inte fungerat (ej tillgänglig/ej existerar längre) automatiskt exkluderas då ingen SSO kan hittas på dessa sidor, denna information hade kunna dokumenteras för att återge i statistik och jämförelse för att kunna ta med detta i beräkning och jämförese av data. Kanske att en ny indelning av befintliga klienter borde göras vid framtida studier som är baserad på antalet fungerande hemsidor, exempelvis genom att flytta efter popularitets-rank när sidor exkluderas på grund utav att de inte fungerar. Om en undersökning för alla grupper utförts även för Google-SSO hade detta kunna användas för att med större säkerhet bedöma om tilläggets funktion behållits, i brist på tid kunde inte detta utföras.

6.4 Jämförelser och svar på forskningsfrågor

OAuthGuard visade i slutgiltiga undersökningen att ungefär 26.5% hade någon form utav brist. Tidigare utförda studie med OAuthGuard [19] visade att 50% hade någon form utav brist, denna siffra är betydligt högre, men OAuthGuard klarar även av att upptäcka fler hot för Google SSO än Facebook SSO och dessa siffror kan så inte rakt av jämföras. Siffran för säkerhetsbrister med Facebook SSO är närmare den siffra som studien [21] visat, men verktyget OAuthGuard (med den funktionalitet som finns för Facebook SSO) och verktyget SSOScan fokuserar på olika säkerhetsbrister där SSOScan i första hand fokuserar på access_token relaterade säkerhetsbrister. Vad som däremot går att jämföra är antalet som har specifika brister samt hur trender för säkerhetsbrister och användning av Facebook SSO ser ut i förhållande till popularitet. Denna undersökning visar på att det är mer vanligt

att populära hemsidor har stöd för Facebook SSO (se Bild 19 och 20), tidigare studie [19] visade att så även var fallet för Google SSO. Vidare visar denna studie att det är vanligare med säkerhetsbrister bland populära hemsidor som stödjer Facebook SSO än för mindre populära hemsidor (se Bild 19), på samma sätt som föregående studie [19] visade detta även för Google SSO. Denna studie visade att den grupp som var mest utsatt för säkerhetsbrister av de undersökta med Facebook SSO var den grupp som har en rank mellan 101 och 200, detta är samma som tidigare studie [19] visat för Google SSO. Alltså följer Facebook SSO och Google SSO vid undersökning med samma verktyg ett liknande mönster vilket inte är helt oväntat då de båda är baserade på samma ramverk, OAuth 2.0, samt i stor del fokuserar på att hitta liknande brister.

Vid jämförelse av specifika attacker visade studien [19] att 39% av undersökta Google SSO klienter var mottaglig mot CSRF vilket kan jämföras med denna studie som visar att 11/98 är mottaglig för denna brist (ungefär 11,2%), denna studie har samma stöd för att upptäcka detta för Facebook SSO som [19] hade för att upptäcka detta för Google SSO. Detta är en betydligt mindre andel som är sårbara mot denna typ av hot. Den mindre undersökningen av Google SSO klienters säkerhetsläge utförd i denna studie visar att 6/18 undersökta i gruppen med rank 101-200 är mottaglig för CSRF (se Tabell 10), detta motsvarar ungefär 33,3% och är en siffra som är lik tidigare studiens [19] resultat (39%). I denna studie hade 2/98 (ungefär 2%) Facebook SSO klienter en osäker anslutning, i studien [19] hade 13/137 (ungefär 9,5%) Google SSO klienter osäker anslutning. Att denna siffra är lägre för klienter med Facebook SSO kan bero på att Facebook inte tillåter osäkra anslutningar och att en sådan implementation inte accepteras och därmed är lättare att upptäcka och åtgärda. Studien [19] visade att 9/137 Google SSO klienter (ungefär 6,5%) läckte tokens via referer header, i denna slutgiltiga undersökning visade det sig att 5/98 (ungefär 5%) Facebook SSO klienter läckte tokens via referer headern. Dessa siffror är liknande och denna brist beror i huvudsak på hur utvecklaren designat inloggningssidan och detta hot är med stor sannolikhet lika vanlig för alla OAuth 2.0 baserade autentiseringslager.

Antalet klienter som stödjer Facebook SSO och Google SSO bland de 500 utvalda sidorna är i en princip samma för båda och ingen tydlig avvikelse av popularitet av någon av dem går att urskilja på detta urval av klienter. Sammanfattningsvis visade denna studie följande resultat jämfört med tidigare utförda studie med OAuthGuard och klienter med Google SSO:

- Ungefär 11.2% av utvalda Facebook SSO klienter hade CSRF brister jämfört med 39% av Google SSO klienter.
- Ungefär 2% av utvalda Facebook SSO klienter hade osäker anslutning jämfört med ungefär 9.5% av Google SSO klienter.
- Ungefär 5% av utvalda Facebook SSO klienter läckte tokens via referer-header jämfört med ungefär 6.5% av Google SSO klienter.
- Samma trend har visats av den grupp som har flest brister i förhållande till klienter som implementerar SSO med Google eller Facebook.
- Samma trend har visats att populärare hemsidor tenderar att ha fler brister

- Facebook SSO och Google SSO är näst intill lika vanligt förekommande bland de 500 populäraste hemsidorna enligt denna studies urval.

Anledningar till att Facebook SSO visar lägre procentuell andel med brister (i denna studies urval) kan bero på att detta autentiseringslager inte är lika komplext som OIDC samt att inbyggda restriktioner som spärrar mot HTTP istället för HTTPS. Detta väcker frågor om huruvida Facebook SSO kan vara att rekommendera över Google SSO för utvecklare som bara vill utnyttja någon form utav SSO/tredjepartsinloggning. Då OAuthGuard visar sig komplicerad att både utveckla och använda korrekt går det att spekulera i att nya tillvägagångssätt för att främja säkerhet i området borde undersökas. Ett förslag på detta är att eventuellt uppmuntra identitetsleverantörer att ha striktare krav och koll på de tjänsteleverantörer som tillämpar deras tjänst, på liknande sätt som Facebook strikt inte tillåter osäkra anslutningar, samt att undersöka om och hur detta teoretiskt skulle fungera. Denna typ av verktyg kan i framtiden visa sig ha betydelse för användarens säkerhet, men just OAuthGuard har en lång väg kvar att gå innan detta blir verklighet. Vidareutveckling av denna typ av verktyg med fokus på användarvänlighet, att korrekt behandla brister och att ge ett brett skydd redan från början är något som behövs för att denna typ av verktyg så småningom ska kunna användas av allmänheten. En aspekt som måste noteras i samband med verktyg som är riktade mot att skydda användare och som användare själva är ansvarig för att hantera är huruvida denna typ av verktyg överhuvudtaget kommer utnyttjas av allmänheten, något som borde undersökas för att inte utveckling av dessa verktyg ska ske utan att sedan användas.

6.5 Etiska aspekter

Verktyg som OAuthGuard är ämnad att användas för att främja säkerhet genom att notifiera utvecklare och användare om brister, så att utvecklare kan åtgärda dem eller så användare kan undvika sidor som potentiellt utsätter resurser för fara. Verktyget skulle dock kunna användas för att lättare hitta hemsidor med brister och sedan utföra en attack mot dem och användare som använder de mottagliga hemsidornas tjänster. Ett antal verktyg, som OAuthGuard, upptäcker brister på webbsidor existerar och är fritt tillgängliga. Utökningen av OAuthGuard-verktyget som är riktat mot vanliga användare kommer inte vara av stor vikt för personer med förmåga att utföra de attacker som tillägget upptäcker och skyddar mot. För att minska risk för missbruk av information har inte specifika hemsidor med säkerhetsbrister nämnts, med undantag för de som redan nämnts och kontaktats av författarna i ursprungliga studien [19].

7 Slutsatser

Kapitlet sammanfattar arbetet kort samt redovisar slutsatser. Förslag till framtida arbete i förhållande till vad denna studie bidragit med redovisas även här.

Tillägget OAuthGuard har undersökts och vidareutvecklats för att även stödja Facebook SSO där Authorization code flow används, tillägget kan alltså inte upptäcka och skydda mot implicit flow eller där response_type är token. Den utökade versionen av tillägget har sedan använts för att utföra ett antal undersökningar mot framförallt klienter med Facebook-SSO. Studien har bidragit med en analys av OAuthGuard som tillägg samt en studie över säkerhetsläget bland klienter som tillämpar Facebook-SSO. Denna studie gav resultat om Facebook-SSO-klienter som gick att jämföra med tidigare data insamlat för Google-SSO-klienter där de huvudsakliga jämförelserna skedde för antalet som är mottaglig för CSRF, antalet som är mottaglig för referer-token leakage, osäker anslutning samt diverse jämförelser av trender.

En liknande trend för Facebook SSO har hittats som för tidigare studie [19] med Google SSO. Den huvudsakliga trenden genom alla undersökningar är att den näst mest populära gruppen av hemsidor är den grupp som har absolut flest brister och högst frekvens av brister med upp till dryga 47% med brister. Sedan är trenden något avtagande, så hemsidor med lägre popularitet visar sig ha färre brister i förhållande till antal implementationer av Facebook SSO. Totalt visade det sig att 26/98 (ungefär 26.5%) av de undersökta hemsidorna hade brister där 11 av dessa saknade användning av/felaktigt använde state parametern för CSRF-skydd. Trenden för antalet privacy-leaks är också avtagande i förhållande till popularitet och detta kan bero på att populärare hemsidor använder sig mer utav bland annat riktad reklam eller annan tredjepartskod.

Förutom mindre variationer av resultat, framförallt vad gäller antal privacy-leaks, som förekommer vid undersökningar av samma grupp och samma metod (troligen beroende på hur lång tid som spenderats på en hemsida) har metoden för undersökningarna visat sig ha stor betydelse för variation av resultat. Detta framförallt vad gäller registrering av potentiella CSRF hot upptäckt genom referer-header (blockerande CSRF). Att återställa en browser-session och ta bort alla cookies gjorde att tillägget inte upptäckte några CSRF som blockerades. Vidare visar det sig att mindre felaktigheter/förändringar i tillägget har stora förändringar på resultat. Att små förändringar gör stor skillnad är ingen överraskning men det visar på svårighet i att använda och utveckla dessa verktyg för att analysera säkerhet, både ur akademisk synpunkt men även ur tillämpningssynpunkt för användare. Då mindre förändringar i metod kan medföra större resultatskillnader kan detta göra att data framtagna ur studier som denna blir mindre precis vid jämförelser mot andra studier med samma verktyg, framförallt om inte metoden dokumenterats minutiöst och replikeras exakt vid varje studie. Den metod användes för att arbeta fram den slutgiltiga undersökningsmetod som användes för den slutgiltiga undersökningen gjorde det möjligt att upptäcka och ge insikt i vad som påverkade dessa resultatvariationer. Vid vidareutveckling av tillägget OAuthGuard hade större fokus på att prioriterat att klara kraven så bra som möjligt behövts.

Vid jämförelse av data visade det sig att Facebook SSO skyddar mot osäker anslutning medan Google SSO inte gör detta samt att Google SSO i gruppen med rank 101-200 har även något högre frekvens av CSRF och andra brister jämfört med Facebook SSO. Facebook SSO kanske är något säkrare eller lättare att implementera korrekt, som skulle

kunna bero på inbyggda restriktioner som att endast tillåta HTTPS men även då detta autentiseringslager är mindre komplext på många sätt än Google SSO som anpassar OIDC (en standard med betydligt fler alternativ för implementation, vilket ökar komplexitet).

Verktyg i form utav tillägg så som OAuthGuard kan nog hjälpa en användare att hålla sig säker, detta med förutsättning att allmänheten/användare visar sig vilja använda dessa verktyg, något som även det borde undersökas. Men för att dessa verktyg ska fungera effektivt och för mer än en identitetsleverantör behöver försiktighet och noggrann strukturering av kod utföras så att verktyget blir felsäkert och mer dynamiskt anpassningsbar. Detta kan nog enklast göras genom att försöka implementera skydd och hantering av maximalt ett par hot men för flertalet identitetsleverantörer för att hålla verktyget konsekvent och skydda inom ett brett utbud av klienter med SSO. När en stabil version av sådant verktyg utvecklats kan den sedan byggas vidare på för att skydda mot fler hot, på detta sätt kan slutanvändare snabbare få tillgång till ett mer stabilt verktyg som ger ett skydd mot vissa hot. I sitt nuvarande skick är OAuthGuard fortfarande inget bra verktyg för allmänheten utan behöver en omstrukturering av bland annat hur data lagras och hur data visas men även en kontroll på hur brister upptäcks och hur tillägget kan skydda mot dem. En risk som finns med denna typ av verktyg är att de kan ge falsk trygghet, framförallt om det används av en utvecklare som vill kolla om en implementation är, enligt verktyget, ”säkert”. Om det däremot är väldigt tydligt att ett verktyg endast ska användas av en slutanvändare och att det framgår tydligt att verktyget inte garanterar att fånga alla hot så borde den potentiella skadan av denna typ av verktyg vara minimal och att de fördelar som finns med denna typ av verktyg kommer överväga nackdelarna.

För framtida studier bör möjligheten för att utveckla ett mer generellt verktyg för att hantera en mängd olika identitetsleverantörer studeras och eventuellt utvecklas. Problemet med att bristande implementationer fortfarande existerar behöver även det undersökas hur det kan lösas, denna studie har visat tendenser till att Facebook SSO bland annat mer sällan än Google SSO lider av CSRF relaterade problem, något som vidare kan undersökas för att se om enkelheten, samt andra faktorer, i Facebook SSO kan vara att föredra och rekommendera för utvecklare som vill implementera SSO för enklare autentisering. Ett alternativ är att även undersöka fler säkerhetsfrämjande metoder där identitetsleverantörerna får ta större ansvar för säkerhet, hur detta kan göras bör då till en början undersökas teoretiskt för att se om detta kan vara möjligt.

Referenser

[1] Säkerhetsbrist, Gmail two-factor authentication [Online]
Tillgänglig: https://motherboard.vice.com/en_us/article/bje3kw/how-hackers-bypass-gmail-two-factor-authentication-2fa-yahoo
[Åtkomst: 20 Maj 2019]

[2] Säkerhetsbrist, Phony email [Online]
Tillgänglig: https://techxplore.com/news/2018-05-easy-two-factor-authentication-hack-phony.html
[Åtkomst: 20 Maj 2019]

[3] Säkerhetsbrist, ”password-manager” [Online]
Tillgänglig: https://www.esecurityplanet.com/network-security/lastpass-password-manager-hacked.html
[Åtkomst: 20 Maj 2019]

[4] SJ, säkerhetsbrist [Online]
Tillgänglig: https://www.gp.se/nyheter/sverige/sj-system-hackat-samtliga-l%C3%B6senord-byts-1.11772999
[Åtkomst: 20 Maj 2019]

[5] Släktforskningssida, säkerhetsbrist [Online]
Tillgänglig: https://techworld.idg.se/2.2524/1.703646/slaktforskning-hackad
[Åtkomst: 20 Maj 2019]

[6] Single sign-on, säkerhetsbrist Facebook [Online]
Tillgänglig: https://www.bankinfosecurity.com/blogs/facebook-breach-single-sign-on-doom-p-2668
[Åtkomst: 20 Maj 2019]

[7] C. Jacomme och S. Kremer, ”An Extensive Formal Analysis of Multi-factor Authentication Protocols”, i 2018 IEEE 31st Computer Security Foundations Symposium (CSF), Oxford, 2018, s. 1–15.

[8] N. Kaur och M. Devgan, ”A Comparative Analysis of Various Multistep Login Authentication Mechanisms”, International Journal of Computer Applications, vol. 127, nr 9, s. 20–26, okt. 2015.

[9] C. Mainka, V. Mladenov, J. Schwenk, och T. Wich, ”SoK: Single Sign-On Security — An Evaluation of OpenID Connect”, i *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, Paris, France, 2017, s. 251–266.

[10] V. Mladenov, C. Mainka, och J. Schwenk, ”On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect”, *arXiv:1508.04324 [cs]*, aug. 2015.

[11] S. Calzavara, R. Focardi, M. Maffei, C. Schneidewind, M. Squarcina, och M. Tempesta, ”WPSE: Fortifying Web Protocols via Browser-Side Security Monitoring”, *arXiv:1806.09111 [cs]*, juni 2018.

[12] S.-T. Sun och K. Beznosov, ”The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems”, 2012, s. 13.

[13] Analysis of social connection data [Online]
Tillgänglig: https://auth0.com/blog/analysis-of-social-connection-data/
[Åtkomst: 5 April 2019]

[14] OAuth 2.0 Authorization framework [Online]
Tillgänglig: https://tools.ietf.org/html/rfc6749
[Åtkomst: 22 Maj 2019]

[15] Definition of a security Vulnerability [Online]
Tillgänglig:
https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc751383(v=technet.10)
[Åtkomst: 20 Maj 2019]

[16] An introduction to OAuth 2.0 [Online]
Tillgänglig: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2
[Åtkomst: 20 Maj 2019]

[17] Google login [Online]
Tillgänglig: https://developers.google.com/identity/
[Åtkomst: 20 Mars 2019]

[18] Facebook login [Online]
Tillgänglig: https://developers.facebook.com/docs/facebook-login/
[Åtkomst: 20 Mars 2019]

[19] W. Li, C. J. Mitchell, och T. Chen, ”OAuthGuard: Protecting User Security and Privacy with OAuth 2.0 and OpenID Connect”, 2019, s. 20.

[20] W. Li, C. J. Mitchell, och T. Chen, ”Mitigating CSRF attacks on OAuth 2.0 and OpenID Connect”, *arXiv:1801.07983 [cs]*, jan. 2018.

[21] Y. Zhou och D. Evans, ”SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities”, 2014, s. 17.

[22] W. Li och C. J. Mitchell, ”Analysing the Security of Google’s Implementation of OpenID Connect”, i *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 9721, J. Caballero, U. Zurutuza, och R. J. Rodríguez, Red. Cham: Springer International Publishing, 2016, s. 357–376.

[23] R. Yang, G. Li, W. C. Lau, K. Zhang, och P. Hu, ”Model-based Security Testing: An Empirical Study on OAuth 2.0 Implementations”, i *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security - ASIA CCS ’16*, Xi’an, China, 2016, s. 651–662.

[24] W. Li och C. J. Mitchell, ”Security Issues in OAuth 2.0 SSO Implementations”, i *Information Security*, vol. 8783, S. S. M. Chow, J. Camenisch, L. C. K. Hui, och S. M. Yiu, Red. Cham: Springer International Publishing, 2014, s. 529–541.

[25] D. Fett, R. Kusters, och G. Schmitz, ”The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines”, i *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, Santa Barbara, CA, USA, 2017, s. 189–202.

[26] V. Mladenov och C.Mainka, ”OpenID Connect Security Considrations”, 2017

[27] Chrome web store [Online]
Tillgänglig: https://chrome.google.com/webstore/category/extensions?hl=sv
[Åtkomst: 3 Maj 2019]