

# **Modelling and Mitigation of Cross-Origin Request Attacks on Federated Identity Management Using Cross Origin Request Policy**

by

Akash Agrawall, Maheshwari Shubh Jagmohan, Projit Bandyopadhyay, Venkatesh Choppella

in

*13th International Conference on Information System Security*

Report No: IIIT/TR/2017/-1



Centre for Software Engineering Research Lab  
International Institute of Information Technology  
Hyderabad - 500 032, INDIA  
December 2017

# Modelling and Mitigation of Cross-Origin Request Attacks on Federated Identity Management Using Cross Origin Request Policy

Akash Agrawall, Shubh Maheshwari, Projit Bandyopadhyay,  
and Venkatesh Choppella

IIT Hyderabad

**Abstract.** Cross origin request attacks (CORA) such as Cross site request forgery (CSRF), cross site timing, etc. continue to pose a threat on the modern day web. Current browser security policies inadequately mitigate these attacks. Additionally, third party authentication services are now the preferred way to carry out identity management between multiple enterprises and web applications. This scenario, called Federated Identity Management (FIM) separates the problem of identity management from the core functionality of an application. In this paper, we construct formally checkable models and design laboratory simulations to show that FIM is susceptible to cross origin attacks. Further, we employ the Cross Origin Request Policy (CORP) to mitigate such attacks.

**Keywords:** Federated Identity Management (FIM), Cross-Origin Request Attacks, CSRF Attacks, Security, Web Browser, World Wide Web, Alloy, Modelling, Mitigation

## 1 Introduction

This paper addresses browser-based cyber attacks. Cyber crime has been around even before the internet [2]. Forbes online estimates that cyber attacks will cost USD 2 trillion globally by 2019 [45]. Notably, costs quadrupled between 2013 and 2015. Newer attack vectors e.g., the browser and IOT devices are adding to the already large set of vulnerabilities.

The World Wide Web began as a way to share marked up (HTML) documents [49]. WWW was created to support the idea of hyperlinks, which initiated a new web (HTTP) transaction to fetch other web documents when explicitly requested by the user (often via a mouse click). Soon however, tags were introduced into the HTML language to support *automatic content inclusion*. Tags like `img`, introduced in 1993, caused additional, i.e., *cascaded* HTTP transactions without explicit user interaction. The addition of such tags were key to enhancing user web experience. The browser itself, via multiple tabs and windows, allowed multiple, concurrent web sessions. These sessions shared crucial browser data structures, like cookies. The confluence of these factors opened up the web to a new variety of malicious attacks with the browser acting as a vector.

Depending on whether the origin of the request is the same as that of the cascading request, the request is either a same origin or a cross origin request. Attacks exploiting vulnerabilities that fail to distinguish the origins of requests are known as *Cross Origin Request Attacks*. Cross origin attacks like *CSRF* are a very popular as well as serious threat to many web services and even feature in the Owasp top ten [4]. They have the ability to tamper with user data and even cause a denial of service. An example of denial of service, was when Github was affected for three days due to a browser based DDoS attack, a type of infiltration attack [15][47].

Same Origin Policy (SOP) [50], the core security policy driving today’s web platform, was designed at a time when the web had static pages connected by hyperlinks. SOP does not address the attacks generated due to content inclusion. Previously, several policies have been proposed to fix the loopholes in SOP and mitigate malicious web based attacks. These proposals have made a significant addition in addressing cross-origin resource inclusion, script inclusion and data exfiltration attacks.

Cross Origin Request Policy (CORP), a browser security policy, was proposed [47] to cover the limitation due to cross-origin resource inclusion. CORP enables a server to control cross-origin interactions initiated by a browser. A CORP compliant browser intercepts the cross-origin requests and blocks the unwanted requests by the server. Our previous work shows the structure and use of CORP in mitigating a certain classes of attacks [47].

As the web is increasingly being used as a communication platform for applications, we are faced with new protocols of user authentication and authorization. *Federated Identity Management (FIM)* [25] was introduced to remove the current gap present in authentication. FIM is a partnership between enterprises where one enterprise (*Service Provider, SP*) trusts another enterprise (*Identity Provider, IdP*) for authentication, thus preventing the overhead of having *authentication credentials* and ensures authentication from an enterprise having better authentication protocol. FIM is an example of a complex cross-origin transaction [13,43].

Over the years, researchers have used formal modelling to analyse the security of network protocols. Modelling is a method which helps make necessary abstractions with the aim of finding out how a system works on the whole. Telikicherla et al. [46] proposed a model demonstrating cross-origin request attacks and its mitigation using CORP. In a similar vein, we create Alloy models (*Pre-CORP* and *Post-CORP*) to include the user authentication flow in FIM systems and demonstrate cross-origin attacks along with their mitigation using CORP. The *Pre-CORP* model captures the current state of the web platform, where we model user authentication via FIM and cross-origin request attacks. The *Post-CORP* model extends this by adding additional signatures, facts and predicates to describe CORP and the constraints it enforces on cross-origin request attacks and shows that no instance of a malicious cross-origin request transaction can occur in the presence of CORP.

Thereon, we conducted experiments in a lab environment, where we made malicious cross-origin request attacks to websites, targeting the login and logout state of the user. We then demonstrate that CORP can be used to mitigate these attacks successfully. Since CORP has a client side dependency, we used a chromium extension to carry out the experiments.

The paper has two main contributions. First, the paper analyses FIM protocols [25] from a security point of view. FIM systems are still vulnerable to cross origin attacks. The paper provides evidence of this phenomena via controlled experimentation with popular sites. Also we verified that these types of cross origin attacks can be mitigated using CORP, as CORP addresses this limitation of current browsers.

The second main contribution of the paper is the modelling of FIM, without and with the presence of CORP. The formal models allowed greater insight on the workflow of FIM protocols and show that the FIM flow is still susceptible to CORA. Further they prove the correctness of CORP by showing that the vulnerabilities can be mitigated in its presence.

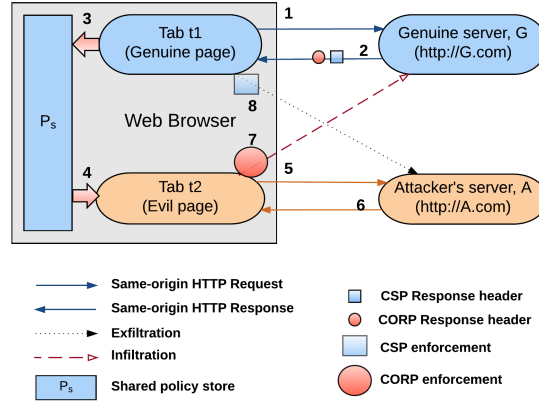
The rest of the sections in this paper are organized as follows: Section 5 describes other mitigation and modelling techniques against this class of attacks and their limitations. Section 2 gives a background overview of FIM and CORP. Section 3 explains the model of FIM protocol SAML and a cross-origin attack on the protocol after the user is authenticated. In this, we also show the mitigation of this attack using CORP. Section 4 describes the experiments we carried out to show cross-site timing and auto-logout attacks as well as their successful mitigation using CORP, and Section 6 concludes with a discussion of future work.

## 2 Background - CORP and FIM

### 2.1 CORP (Cross-Origin Request Policy)

Telikicherla et al. proposed *CORP* (Cross Origin Request Policy) to mitigate cross-origin request attacks such as Cross-Site Request Forgery (CSRF) [1], click-jacking [42], and cross-site timing attacks [23]. *CORP* can be seen as a policy that controls *who*, i.e., which site or origin, can access *what*, i.e., which resources on a cross-origin server (e.g. `/img/*`, `/img/xyz.jpg`, etc), and *how*, i.e., through which browser event (e.g. `<img>`, `<script>` or other tags). *CORP* is declarative, thus it can be added as an HTTP response header of the website. A web browser enforcing *CORP* would receive the policy in the response header of the website. The browser would store the *CORP* in memory accessible to all tabs in the browser such that when any cross-origin request goes to this website, the browser will intercept it and allow it only if it complies with the *CORP* of the website.

Figure 1 shows the model of a browser which supports *CORP*. It shows the difference between exfiltration and infiltration attacks, thereby explaining how *CORP* differs from CSP. The figure shows a genuine server *G*, with origin



**Fig. 1.** Browser model showing exfiltration and infiltration and their mitigation by CSP and CORP respectively.

`http://G.com`, an attacker's server  $A$ , with origin `http://A.com` and a browser with two tabs -  $t1$  and  $t2$ . A general browsing scenario, which is also the sufficient condition for a cross origin attack, where a user logs in at  $G.com$  in  $t1$  and then (unwittingly) opens  $A.com$  in  $t2$  is depicted in the model.

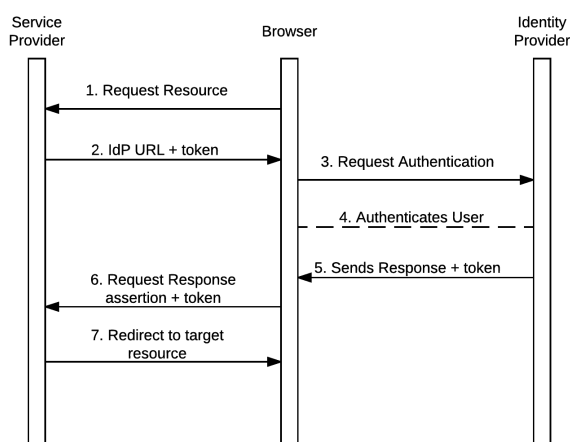
Once a user requests the genuine site  $G.com$  by typing its URL in the address bar of  $t1$ , an HTTP request is sent from  $t1$  to  $G$  and in response, along with content, the declarative security headers  $CSP$  and  $CORP$  are sent by  $G$  (shown by arrows 1 and 2 in the figure). The tab  $t1$  receives these policies, stores  $CSP$  in its local store (as the current CSP-enabled browsers do) and sends  $CORP$  to a shared policy store  $P_s$ .  $P_s$  ensures that  $CORP$  is available to every tab/instance (arrows 3 and 4 in the figure) of the browser. Now, when a user unintentionally opens a malicious page loaded from  $A$  in  $t2$  (arrows 5 and 6 in the figure), every HTTP request initiated by the page in  $t2$  to  $G$  will be scrutinized and restrictions in  $CORP$  will be enforced (location 7 in the figure). Requests from  $t2$  to  $G$  will be allowed only if they comply with the configuration in the policy.  $CORP$  needs to be configured on the server such that when a request is sent to the server, it returns  $CORP$  in the response header. Complete guidelines on the declaration of  $CORP$  can be found in [47].

## 2.2 FIM (Federated Identity Management)

FIM is a partnership between enterprises where trust prevails between the *Service Provider*, ( $SP$ ) and *Identity Provider*, ( $IdP$ ) for authentication. The Service provider provides a service to the user while the Identity provider is a third party service which does authentication for it. FIM prevents the overhead of having *user credentials* and ensures authentication from an enterprise having a better user management system. Technologies used for federated identity include *SAML* (Security Assertion Markup Language) [8], *OAuth* [34], *OpenID* [41], *Se-*

*curity tokens*, etc. Single Sign-On (SSO) is a protocol of access control by which a user logs in with their credentials once and enters a federated environment to gain access to various services without having to maintain separate credentials for each [19]. Google is a prime example of this: logging with Google Accounts allows one to use services like Google Drive, Gmail, etc.

We model the FIM protocol - *SAML* since it is a widely used protocol which is extensible, and has strong authentication and authorization systems, thus removing the necessity of having multiple web application passwords. It also allows for single sign-on and single logout. In this, there are three main interacting agents: the service provider, identity provider and the browser. Figure 2 shows a simple transaction workflow of the *SAML* protocol. We divide the transactions involved into 3 parts:



**Fig. 2.** Transaction Flow - Federated Identity Management (FIM)

- **Request resource from SP:** In this transaction (steps 1 and 2), user types in the *service provider’s* address in *URL address bar*. The service provider sends a response to the user with an *HTML button* to “*Login with IdP*”. The service provider also sends a token in the response. Later, this token is sent to the IdP for identifying the partner server SP.
- **Request authentication from IdP:** When user clicks on “*Login with IdP*” button, a request goes from browser to Identity provider (step 3). The user is then authenticated with the *IdP* (step 4). After the authentication, IdP sends an encrypted response containing user information and the token (step 5).
- **Request assertion from SP:** The authentication is done at IdP’s end till now. The SP has to verify IdP’s response before giving the user access to its resource. A *redirection* request is sent to *service provider* to verify the response sent by IdP (step 6). The SP verifies the response as well as checks if the token matches the token sent by it in step 2. We do not model the verification protocol in this model. If the response is successfully verified, then the user is redirected to the target resource (step 7).

### 3 Alloy Models of FIM, CORA and Mitigation by CORP

In this section, we propose a model for cross-origin request attacks in a transaction involving FIM. We then extend this model and demonstrate that no instance of malicious cross-origin request attacks can occur in the presence of CORP.

#### 3.1 Salient Features of CORP Alloy Model

This section explains the design considerations of the formal model created in Alloy to validate the soundness of CORP. Alloy<sup>1</sup> is a lightweight, declarative modelling language based on first order relational logic which helps describe the structural properties of a model. Borrowing the basics of the CORP model developed by Telikicherla et al. [46], we created a more modular version specifically for user authentication in FIM systems. Below are the key design considerations of our model:

The model comprises a Browser, Service Provider, Identity Provider and Attacker Server. A *User* uses the *Browser* to interact with various components of the model. First, the User tries to access a *Service Provider (BrowserTab1)*. The User is then redirected to the *Identity Provider*. Upon successful authentication, the User is redirected back to Service Provider. The *attack website (BrowserTab0)*, now becomes active (could have opened at any point of time). A malicious cross-origin request is made from *BrowserTab0* to *BrowserTab1*, thereby affecting the user's state. We model this attack (Pre-CORP model) and mitigate it using *CORP* (Post-CORP model). Thereon, it is assumed that the attacker web page is fetched in another tab. To ensure simplicity, we avoid inclusion of response assertion by the service provider in our model. There are four transactions: the first three are concerned with the authentication via FIM and the fourth is a malicious cross-origin transaction from the *attacker server*, which we mitigate in the *Post-CORP* model.

There are a few essential keywords required to define models in alloy. The *sig* keyword helps define abstract objects in the model. *Fact* is used to enforce invariable constraints while *pred* creates instances of the model following certain conditions.

In this model all the relations between the objects are stored in a dynamic signature called *State* by using the built-in *Ordering* library [6]. The four transactions are shown by instances of the *State* (listing 1.1).

#### 3.2 Modelling cross-origin request transaction in FIM (Pre-CORP Model)

In this section, we describe the modelling of cross-origin request transaction along with user authentication via FIM (SAML protocol).

<sup>1</sup> <http://alloy.mit.edu/>

**HTTPTransaction, Browser and Server** To make more realistic model, all the requests and responses are shown using HTTP Transactions (line 7-10 in Listing 1.1). We define multiple objects in Alloy, and their dynamic behaviours are defined in State (explained in Section 3.2). The transaction could be of four types *StartAuth*, *IdPAAuthentication*, *SPAAuthentication* and *CORA*. In the model, *Browser* keyword represents the user's browser. A browser will have multiple *browser tabs*. For the user to interact with multiple servers from the same instance of the Browser at the same time browser tabs are used. There are three different type of servers: *IdentityProvider*, *ServiceProvider* and *AttackerServer*. The *Attacker Server* is the entity which makes a malicious request to the Genuine Servers using the *HTML Elements*. *HTTPEventInitiator* refers to the objects which can trigger an HTML transaction. It can be triggered due to *Redirection* or various other *Elements* (line 5). We classify *Elements* into: *URLAddressBar* and *HTMLElements*. HTML elements can be classified into two types: the elements which cannot trigger HTTP requests are *Passive* HTML elements (e.g. Div, Span, Textbox, etc.) while those which can trigger HTTP requests are *Active* HTML elements (e.g. iframe, script, img, etc.). We do not model passive HTML elements here to keep the model simpler.

**State and Status** Listing 1.1 shows the signature where we define *State*. We have defined *Status* such as to map the relationships: *IDP*, *SP*, *authenticationStatus*. *IDP* (line 12) and *SP* (line 13) keeps track of the authentication status of *IdentityProvider* and *ServiceProvider* respectively. *authenticationStatus* (line 14) keeps track of the authentication state of the user and will be *1*, meaning active state, if and only if the states: *IDP* and *SP* are *1*.

---

```

1      sig State {
2          transType : HTTPTransaction -> one TransType ,
3          token : HTTPTransaction -> one Int ,
4
5          httpEventInit: HTTPEventInitiator one -> lone
              HTTPTransaction ,
6
7          req_to : HTTPTransaction -> one Server ,
8          resp_from : Server -> lone HTTPTransaction ,
9          resp_to : HTTPTransaction -> one BrowserTab ,
10         req_from : BrowserTab -> HTTPTransaction ,
11
12         IDP : Status -> one Int ,
13         SP : Status -> one Int ,
14         authenticationStatus : Status -> one Int
15     }
```

---

**Listing 1.1.** State and Status

*transType* specifies the type of transaction in each transaction. *token* refers to the relay token present in FIM transactions. *httpEventInit* is the *HTTPEventInitiator* which triggers the transaction. A request always originates from *BrowserTab*



and goes to *Server*. Similarly, a response originates from *Server* and goes to *BrowserTab*. Hence, the relations: *req\_to*, *resp\_from*, *resp\_to* and *req\_from* are defined according to this concept (lines 7 - 10). To preserve the ordering of various transactions present in the model, we order the States starting with *s0* to *s3* in the model using the inbuilt library, *ordering*.

### 3.3 Modelling of FIM and CORA without CORP

To show the various transactions present in the model, we use different instances of *State* ordered from *s0* to *s3*.

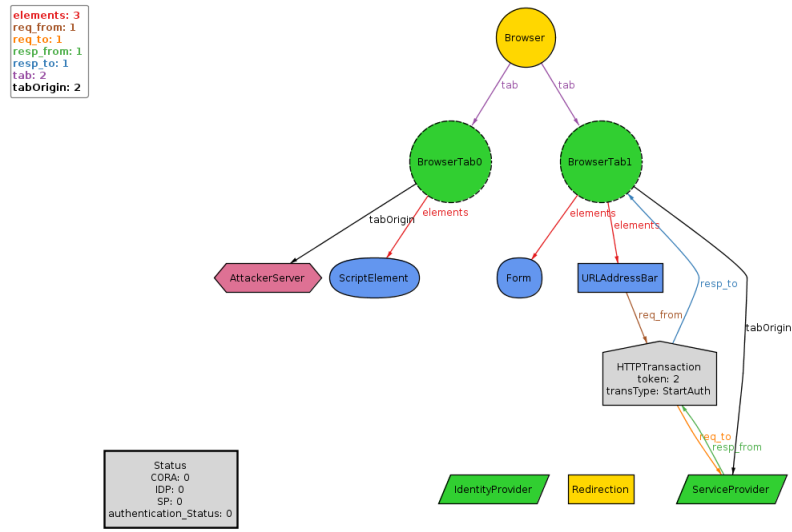


Fig. 3. State - *s0*: User access the Service Provider

***s0 (StartAuth: User accesses SP in BrowserTab1)*** Figure 3 shows the model projected on the state - *StartAuth*. The request originates from *BrowserTab1* and goes to *ServiceProvider*. The response is received from *ServiceProvider* to *BrowserTab1*. Since the user types in the website's address in the *URLAddressBar*, the *httpEventInit* of the transaction is *URLAddressBar*. The *tabOrigin* of *BrowserTab1* is *ServiceProvider* since the content is fetched from this server. The *ServiceProvider* returns a *token*, in the transaction, with value 2. Since, no authentication is completed here, the values of *IDP*, *SP* and *authenticationStatus* are 0.

***s1 (IdP Authentication: User logs in with IdP)*** In this, the user clicks on "Login with IdP" and the browser tab sends a POST request from the *form* element to the *Identity Provider*. The response is then returned from the *Identity Provider* to the browser tab.

**s2 (SP Authentication: IdP sends conformation of the User authentication to SP)** If the *Identity provider* has successfully *authenticated* the user, the browser tab redirects the information in the form of *token* to the *Service Provider*. After authentication and authorization of the user, the *Service Provider* returns the response to the browser tab. Thus, by the end of this state, the user is authenticated by the system.

**s3 (Cross Origin Request Attack: Attack triggered by User interaction in BrowserTab0)** In this state, the user unwittingly visits an *AttackerServer* which initiates a malicious cross-origin request attack from an *HTML Element* to the *Service Provider*, thus affecting the user even after he or she has successfully been authenticated.

### 3.4 Modelling the Mitigation of CORA, with CORP (Post-CORP Model)

We extend the previous model to include attributes and predicates, to demonstrate the mitigation of cross-origin request attack using *CORP*.

**CORP and RequestPath** In this section we define the CORP object in our model.

- **CORP:** The working of CORP has been explained in Section 2.1. In the described model we consider the *Service Provider* to be the *genuine server*. *Browser* stores the *CORP* policy in a shared memory, hence we show CORP as an attribute of the Browser. CORP consists of: *Who* - the source origin of the request, *Where* - the destination path of the request, *How* - the *HTTPEventInitiator* of the request. We have defined the *Who* and *How* in our *Pre-CORP* 3.1 model. In the Post-CORP model *CORP*'s *origin*, *path*, and *how* are added in the *State*, since the *browser* enforces *CORP* only in the cross-origin transaction. In *Post-CORP*, *RequestPath* is also defined.
- **RequestPath:** There can be *two* types of *RequestPath*: *SensitivePath*, the end-points of websites which are vulnerable to cross-origin request attacks (e.g. */transferMoney*, */delete\_user*, etc), and *NonSensitivePath*, the end-points of websites which are not vulnerable to cross-origin request attacks (e.g. */images*, */videos*, etc).

We model the cross-origin transaction with the assumption that the content from *Attacker Server* is already fetched in user's browser tab, to keep the model simpler.

**Pred - CorpCompliantTransaction** Listing 1.2 filters out the cross-origin transaction with destination as *ServiceProvider*. It takes 3 arguments: *origin* (*sv*), *path* (*pt*), and *how* (*ev*). To make sure that all the cross-origin transactions are complaint, it calls predicate *corpCheck*. *corpCheck* asserts whether the *HTTPEventInitiator* (*ev*) of the *Server* (*sv*) is allowed to make a cross-origin request to the *ServiceProvider*

```

1      pred corpCompliantTransaction [sv:Server, ev:
2      HTTPEventInitiator, pt: RequestPath] {
3          all s:State |
4              let sTab= HTTPTransaction.(s.resp_to) | {
5                  sTab.tabOrigin != HTTPTransaction.(s.
6                      req_to) &&
7                  HTTPTransaction.(s.req_to) =
8                      ServiceProvider
9              =>
10                 corpCheck[s, sTab, sv, ev, pt]
11             else
12                 no s.how &&
13                 no s.path &&
14                 no s.http_path &&
15                 no s.origin
16         }
17     }

```

Listing 1.2. Pred - CorpCompliantTransaction

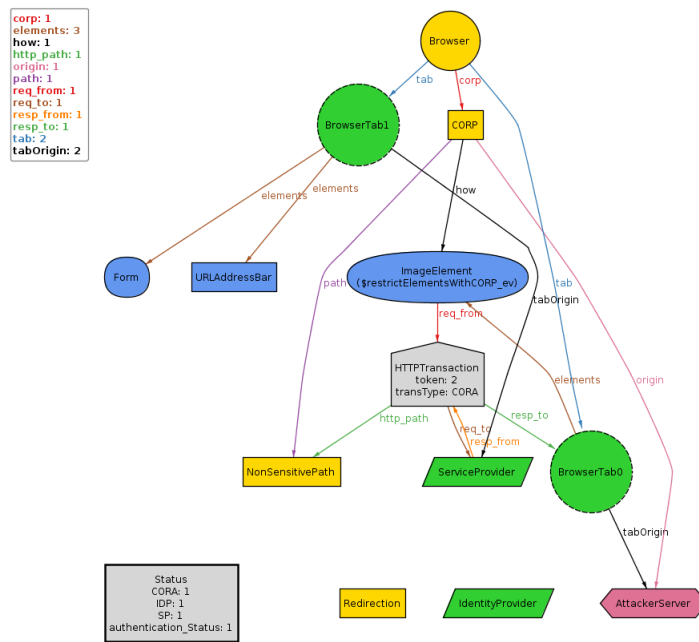


Fig. 4. State - S3 (CORA)

**Instances - Post-CORP Model** We generate an instance of Post-CORP model where we add the predicate *restrictElementsWithCORP* in the code. Figure 4 shows the last transaction of the model where *CORP* is enforced on the *HTTPTransaction*. The request is not a malicious cross-origin request since the destination *RequestPath* is a *NonSensitivePath*. *CORP* has white-listed the *AttackerServer*, the origin of the transaction, and *ImageElement*, the *HTTPEventInitiator* of the transaction.

**Instance - Malicious Cross-Origin Transaction** Listing 1.3 shows the predicate *maliciousCrossOriginAttackWithImage*, which tells alloy to generate an instance where the destination path of *HTTPTransaction* is *SensitivePath*. We run this predicate for at most 20 elements of each object.

---

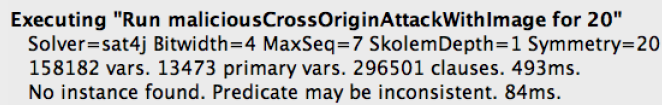
```

1         pred maliciousCrossOriginAttackWithImage {
2             CrossOriginAttack
3             HTTPTransaction.(s3.http_path) =
                SensitivePath
4         }
```

---

**Listing 1.3.** Instance - Malicious Cross-Origin Transaction

Figure 5 shows that there is no such instance generated by alloy, which shows that our model is consistent.



```

Executing "Run maliciousCrossOriginAttackWithImage for 20"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
158182 vars. 13473 primary vars. 296501 clauses. 493ms.
No instance found. Predicate may be inconsistent. 84ms.
```

**Fig. 5.** No instance of malicious CORA generated

## 4 Experimentation

In this section, we describe the experimentation done to show the vulnerabilities present in the authentication endpoints: *Login* and *Logout*, and mitigate the risks using CORP.

### 4.1 Mitigating auto-logout attack - CSRF attack

In this attack, an attacker page makes a cross-origin request to the logout endpoint, without the user's consent. Logout CSRF is a known vulnerability, and attacks have been demonstrated by various attackers <sup>2</sup>.

<sup>2</sup> <http://superlogout.com/>, <http://superlogout.github.io/>

Though Google services (which use SSO <sup>3</sup>) are vulnerable to this attack, Google claims that “it cannot be reliably addressed on the modern web” [10]. However, we argue that this attack can pose substantial risks to modern websites as it can be extended to a denial of service to the user - while the malicious page is open, the user would continuously be logged out of some services. Malicious network administrators or those with knowledge of a user’s activities could conduct the logout attack at crucial points of time which may result in a loss of data. This attack can also be used in conjunction with other techniques to create phishing attacks [40,14] . Some mitigation techniques have been proposed like using POST requests, via a form, to logout users [9], and using csrf cookies [11]. These, however, are application level fixes. The safety of the user’s data will be dependent on security measures that were taken by the service provider.

**Auto-Logout attack** This attack works by sending a cross-origin request, on behalf of the user, to the logout endpoint of a server thereby logging out the user. To achieve this, the attacker injects an *img* attribute into an HTML page using the malicious code mentioned in Listing 1.4.

```

1      var img = document.createElement("img");
2      img.src = https://www.genuine.com/logout;

```

Listing 1.4. CSRF attack - Logout endpoint

The attacker can obtain the logout endpoint of a genuine website by using the network console in browsers like firefox <sup>4</sup> and chrome <sup>5</sup>. Even popular websites like Google do not have a check for a cross-origin request at the logout endpoint [10]. Hence, without the user’s consent, the attacker can log it out of the service.

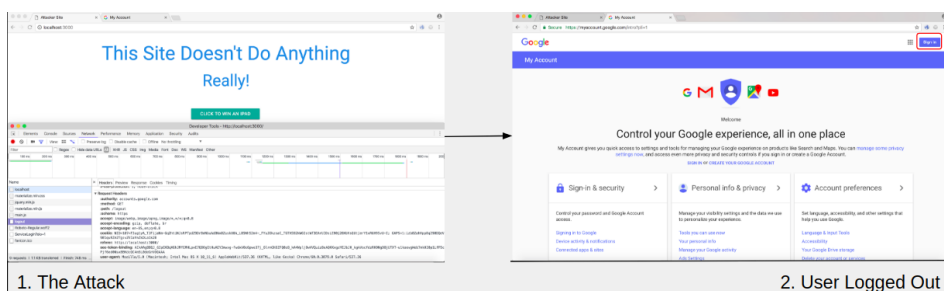


Fig. 6. Logout CSRF Attack simulation on Google

<sup>3</sup> <https://accounts.google.com>

<sup>4</sup> [https://developer.mozilla.org/en-US/docs/Tools/Network\\_Monitor](https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor)

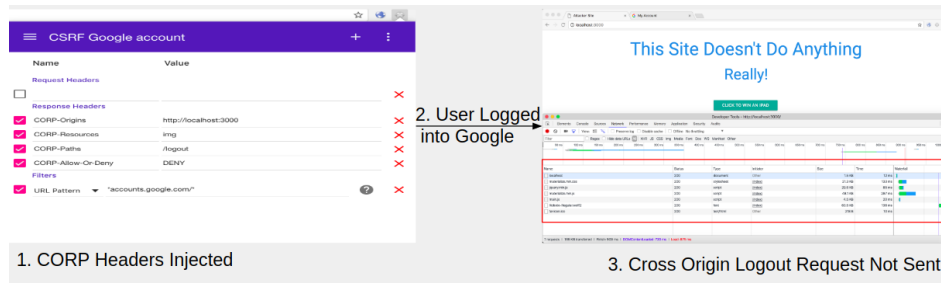
<sup>5</sup> <https://developer.chrome.com/devtools>

**Reproducing the attack** Figure 6 shows the steps we followed to reproduce the attack on *Google*. A user first logs in at `https://accounts.google.com` and then visits an attacker website, containing the code as described in Listing 1.4, in a separate browser tab. The steps involved are:

- **Step 1:** Shows the malicious cross-origin request attack from *Network* tab of Chromium developer tools.
- **Step 2:** Shows that user is logged-out as a result.

As described in Section 2.2, Google uses SSO for user authentication. Hence, if the user is logged-out from `https://accounts.google.com`, it is logged-out of all *Google* websites, e.g. *Google drive*, *Gmail*, etc.

Logout endpoints of a site can be found by observing the *network activity* from the browser console.



**Fig. 7.** Mitigation of Logout CSRF Attack on Google

**Mitigation using CORP** In this, we show that the attack can be successfully mitigated using *CORP*. Figure 7 shows the steps we follow to mitigate the attack. The following points describe the steps mentioned in the figure:

- **Step 1:** We have used the *Modify headers* extension [3] for google chromium to inject response headers in the HTTP response received from *Google*, since we don't have access to its webserver. Listing 1.6 shows the CORP headers used in this example to mitigate the attack. It shows that the *Origin* `http://localhost:3000` cannot send cross-origin request to `https://accounts.google.com` via `img` tag to `/logout` path.

---

```

1      CORP-Origins: http://localhost:3000
2      CORP-Resources: img
3      CORP-Paths: /logout
4      CORP-Allow-Or-Deny: DENY

```

---

**Listing 1.5.** CORP Rule - Mitigating Logout CSRF attacks

- **Step 2:** We reload the url `https://accounts.google.com` for the CORP headers to be injected. The browser receives the response and stores it in a shared memory store.

- **Step 3:** It shows that no malicious cross-origin request is sent to `https://accounts.google.com`.

## 4.2 Mitigating User login detection attack - Cross-site timing attack

Cross-site timing attacks enable an attacker to receive information from a user's view of a victim site [22]. In our attack, we obtain information about a user's current state: logged in or not. Though this vulnerability has been known for quite a while [48] now, most companies neglect it [35]. To execute this attack, resources which are only accessible upon authentication are found and requested for by the attacker site. Some mitigations methods exist like hosting such resources on a separate CDN server [12]. However, vulnerabilities still exist, as shown by the attack.

Knowledge of whether a user is logged in or not is crucial to the success of many other attacks like phishing [32], deanonymization attacks [16], clickjacking [42], and profilejacking [5]. This kind of attack can also be used to gather user data: if a person is logged into sites like *Facebook*, *Twitter*, etc. then they are a socially active person.

**User login detection attack** The vulnerability exploited in this attack is the redirection present in the login endpoint. Redirection information is provided to the login page as a query string, for example:

```
https://www.genuine.com/login.php?next=https:%2F%2Fwww.genuine.com%2Fmyprofile.php
```

This kind of URL directs the user to the profile page if logged in, and to the login page if the user is logged out. Due to Same Origin Policy, cross-origin ajax requests to genuine server would not be entertained. Cross-origin requests for content inclusion (via *img*, *script*, etc. tags) however would be allowed. Most servers host resources (images, scripts, videos, etc.) to be shared on a CDN (Content Delivery Network). One resource that is not hosted on the CDN and is found in most web servers is the *favicon.ico*.

The login url will look like this after setting up with favicon parameter:

```
https://www.genuine.com/login.php?next=https%3A%2F%2Fwww.genuine.com%2Ffavicon.ico
```

The above URL will return *favicon* if user is already logged in, otherwise it will return html of login page. Hence, this URL can be used in `<img>` tag in the website: `<img src=https://www.genuine.com/login.php?next=https%3A%2F%2Fwww.genuine.com%2Ffavicon.ico>` The behaviour of `<img>` tag will be as follows:

- If the user is already logged in, the image will be loaded.
- If the user is logged out, then the request will receive the HTML page of the login screen. This will not be loaded as an image. Hence, it will trigger the *onError* callback

The final exploit will be:  
`<img onload = "alert('logged in to genuine website')"` `onerror = "alert('not logged in to genuine website')"` `src = "https://www.genuine.com/login.php?next=https %2F%2Fwww.genuine.com%2Ffavicon.ico">`

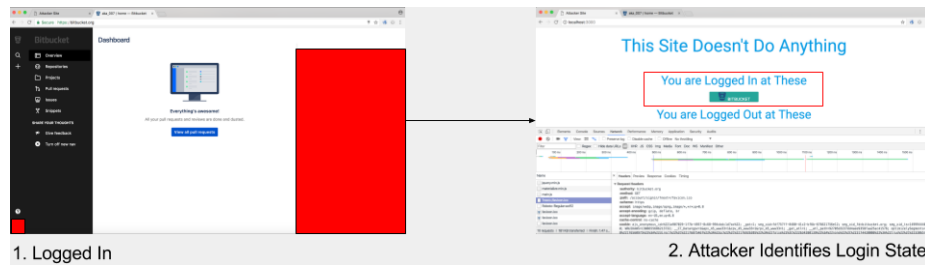


Fig. 8. Login detection Attack - Cross Site timing attack (Simulation on Bitbucket)

**Reproducing the attack** Figure 8 shows the steps we followed to reproduce the attack on *BitBucket*. The following points describe the steps specified in the figure:

- **Step 1:** Shows a logged-in user on <https://bitbucket.org/>.
- **Step 2:** Shows the attacker website which detects that user is logged-in at *BitBucket*. The network console of Chrome developer tools shows the malicious cross-origin request reaching *BitBucket*.

*BitBucket* uses *Google* as its *IdentityProvider* for authentication and is still vulnerable to cross-site timing attack. This shows that even after using FIM, a website can still be vulnerable to cross-origin request attack.

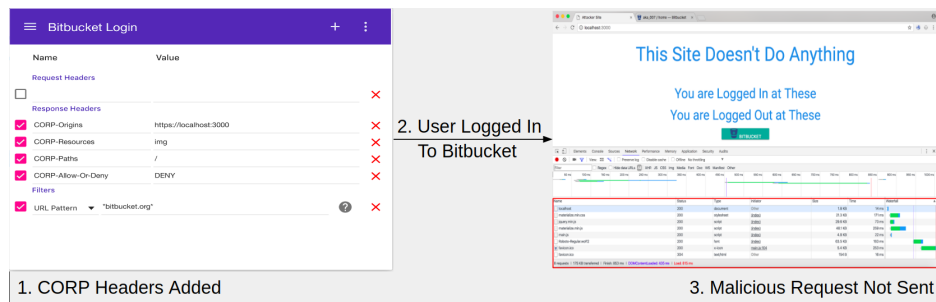


Fig. 9. Login detection Mitigation - Cross Site timing attack (Simulation on Bitbucket)



**Mitigation using CORP** In this, we show that the attack can be successfully mitigated using *CORP*. Figure 9 shows the steps we follow to mitigate the attack. The following points describe the steps mentioned in the figure:

- **Step 1:** We have used *Modify headers* extension [3] for google chromium to inject response header in HTTP response received from *BitBucket*, since we don't have access to it's webserver. Listing 1.6 shows the CORP headers used in this example to mitigate the attack. It means that the *Origin* `http://localhost:3000` cannot send cross-origin request to `https://bitbucket.org/` via *img* tag to */\** path.

---

```

1          CORP-Origin: http://localhost:3000
2          CORP-Resources: img
3          CORP-Paths: /*
4          CORP-Allow-Or-Deny: DENY

```

---

**Listing 1.6.** CORP Rule - Mitigating Logout CSRF attacks

- **Step 2:** We reload the url `https://bitbucket.org/` for the CORP headers to be injected. The browser receives the response and stores it in a shared memory store.
- **Step 3:** It shows that no malicious cross-origin request is sent to `https://bitbucket.org/`.

## 5 Related Work

In this section, we discuss the explorations of various security researchers in modelling web security protocols and in mitigating cross-origin request attacks.

### 5.1 Modelling

Usage of formal verification to analyse the security of network protocols is not a new concept. Akhawe et al. [17] created a formal model of web security based on an abstraction of the web platform which includes essential details about the browser, servers, cookies, HTTP protocol and discussed different scenarios apropos attacks initiated by various types of potential attackers. Similarly, Ryck et al. modelled CSRF attacks to demonstrate how their proposed method can be used to mitigate the attack using Alloy [31]. Chen et al. proposed App isolation in a single browser to increase the security of many browsers [27]. Alloy was used to model a web browser and verify the proposed method. Cao et al. has used Alloy to model their proposed Configurable Origin Policy (COP) and test it [24]. There have been many such initiatives undertaken to validate web security services by modelling [18,21,29,30,33]. Telikicherla et al. [46] proposed an alloy model demonstrating cross-origin request attacks and its mitigation using CORP. We extend this model to include authentication flow of user via FIM and show cross-origin attacks along with their mitigation using CORP.

## 5.2 Existing Browser security policies

Same Origin Policy (SOP) [50] was designed to prevent scripts from accessing the DOM (Document Object Model), network and storage data belonging to other web origins. The earlier problem of cross-origin requests through automatic form submissions or content inclusion was, however, left unanswered by SOP. Content Security Policy (CSP), introduced in 2010 [28] improves on SOP in mitigating the exfiltration attack (as shown in Figure 1) by disabling inline scripts and restricting the sources of external scripts. CSP allows a website to instruct the browser to load resources from specific sources. However, CSP can only prevent exfiltration attacks, such as cross-site scripting (XSS) [44], and not cross-origin request attacks via content inclusion (as shown in Figure 1).

## 5.3 Other Mitigation Measures

Some other mitigation measures are: Same-site cookie, CSRF Guard [26], CSRFx [26], NoForge [20], SOMA [39], RequestRodeo [36], Browser Enforced Authenticity Protection (BEAP) [38], etc. *Same-site cookies* prevent the browser from sending authentication cookies with cross-origin requests [7]. However, it cannot prevent a server from being flooded with malicious cross-origin requests, thereby leading to a browser-based DDoS attack. This is a type of DDoS attack where *users* and *browsers* are used as vectors to send continuous malicious cross-origin request attacks to a genuine server. *CORP* can mitigate this class of attacks as well [15]. *CSRF Guard* and *CSRFx* are techniques which use tokens generated by the server which are sent to the browser tab to prevent malicious requests from being served from a separate tab which has the attacker’s website open. These tokens can, however, be stolen by the attacker via phishing [32] or social engineering techniques [37]. *NoForge* is a server side proxy which implements token validation by adding a secret token to the HTTP response. However, HTML dynamically generated from user’s webpage can still launch a malicious cross-origin attack on the website since no token would have been inserted in it. *BEAP* is a browser based solution, which infers whether a request is malicious or not by analysing real-world application, and removes the authentication cookies. However, it strips the authentication cookies of several genuine cross-origin requests, which are common on the web.

## 6 Conclusion and Future Work

As shown by our experiments, current browser security policies are yet to implement measures to protect against all types of content inclusion cross-origin request attacks. We have investigated a particular scenario that involves FIM (Federated Identity Management) and its susceptibility to cross-origin request attacks. We have built formal models in Alloy of cross-origin request attacks affecting FIM and showed that they can be mitigation using CORP. To validate this hypothesis, we simulated *auto-logout attack* (CSRF attack) and *login*

*detection attack* (cross-site timing attack) on popular sites and mitigated them using CORP in a lab environment. The generic model introduced in this paper subsumes many specific cross-origin attacks, such as CSRF, clickjacking, cross-site timing attack, login detection, and validates the soundness of CORP in mitigating these attacks.

We have implemented CORP in the Chromium browser and are carrying out experiments to test CORP's feasibility. In the future, we plan to perform extensive testing of CORP on other platforms of Chromium (Android, iOS etc.) and propose CORP as a standard to the Chromium community. We invite the research community to help integrate CORP with other open-source browsers (like *Firefox*, *Opera* etc.) and verify its utility. On successful integration, a compelling step would be to propose CORP as a web standard to the W3C community <sup>6</sup>.

## References

1. Cross-Site Request Forgery (CSRF), [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))
2. Cybercrime Time-Line, <http://www.symantec.com/region/sg/homecomputing/library/cybercrime.html>
3. Modify Headers for Google Chrome, <https://chrome.google.com/webstore/detail/modify-headers-for-google/innpjfdalfhpcoinfnehdnbkglpmogdi>
4. OWASP Top 10 Application Security Risks - 2017, [https://www.owasp.org/index.php/Top\\_10\\_2017-Top\\_10](https://www.owasp.org/index.php/Top_10_2017-Top_10)
5. ProfileJacking - legal tricks to detect user profile. Blog, <https://sakurity.com/blog/2015/03/10/Profilejacking.html>
6. Rivery crossing, <http://alloy.mit.edu/alloy/tutorials/online/frame-RC-1.html>
7. Same Site Cookie, <https://www.owasp.org/index.php/SameSite>
8. Security assertion markup language. Article, [https://en.wikipedia.org/wiki/Security\\_Assertion\\_Markup\\_Language](https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language)
9. User logout is vulnerable to csrf, <https://www.drupal.org/node/144538>
10. Xsrf in the logout handler, <https://sites.google.com/site/bughunteruniversity/nonvuln/logout-xsrf>
11. Yii 1.1: Logout csrf protection, <http://www.yiiframework.com/wiki/190/logout-csrf-protection/>
12. Your Social Media Fingerprint, <https://robinlinus.github.io/socialmedia-leak/>
13. Federated sso primer (April 2015), <https://developer.pingidentity.com/en/resources/federated-sso-overview.html>
14. Login/logout csrf: Time to reconsider? Article (Mar 2017), <https://labs.detectify.com/2017/03/15/loginlogout-csrf-time-to-reconsider/>
15. Agrawal, A., Chaitanya, K., Agrawal, A.K., Choppella, V.: Mitigating browser-based ddos attacks using corp. In: Proceedings of the 10th Innovations in Software Engineering Conference. pp. 137–146. ACM (2017)
16. Ahmed Elsobky: Novel Techniques for User Deanonimization Attacks, <https://Oxsobky.github.io/novel-deanonimization-techniques/>

<sup>6</sup> <https://www.w3.org/>

17. Akhawe, D., Barth, A., Lam, P.E., Mitchell, J., Song, D.: Towards a formal foundation of web security. In: *Computer Security Foundations Symposium (CSF)*, 2010 23rd IEEE. pp. 290–304. IEEE (2010)
18. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., et al.: The avispa tool for the automated validation of internet security protocols and applications. In: *International Conference on Computer Aided Verification*. pp. 281–285. Springer (2005)
19. Armando, A., Carbone, R., Compagna, L., Cuellar, J., Tobarra, L.: Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In: *Proceedings of the 6th ACM workshop on Formal methods in security engineering*. pp. 1–10. ACM (2008)
20. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: *Proceedings of the 15th ACM conference on Computer and communications security*. pp. 75–88. ACM (2008)
21. Bhargavan, K., Fournet, C., Gordon, A.D.: Verified reference implementations of ws-security protocols. In: *International Workshop on Web Services and Formal Methods*. pp. 88–106. Springer (2006)
22. Bortz, A., Boneh, D.: Exposing private information by timing web applications. In: *Proceedings of the 16th international conference on World Wide Web*. pp. 621–628. ACM (2007)
23. Bortz, Andrew and Boneh, Dan: Exposing private information by timing web applications. In: *Proceedings of the 16th international conference on World Wide Web*. pp. 621–628. ACM (2007)
24. Cao, Y., Rastogi, V., Li, Z., Chen, Y., Moshchuk, A.: Redefining web browser principals with a configurable origin policy. In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. pp. 1–12. IEEE (2013)
25. Chadwick, D.W.: Federated identity management. In: *Foundations of security analysis and design V*, pp. 96–120. Springer (2009)
26. Chen, B., Zavorsky, P., Ruhl, R., Lindskog, D.: A study of the effectiveness of csrf guard. In: *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*. pp. 1269–1272. IEEE (2011)
27. Chen, E.Y., Bau, J., Reis, C., Barth, A., Jackson, C.: App isolation: get the security of multiple browsers with just one. In: *Proceedings of the 18th ACM conference on Computer and communications security*. pp. 227–238. ACM (2011)
28. Chu, Yang-Hua and Feigenbaum, Joan and LaMacchia, Brian and Resnick, Paul and Strauss, Martin: REFEREE: Trust management for Web applications. *Computer Networks and ISDN systems* 29(8), 953–964 (1997)
29. Clarke, E.M., Jha, S., Marrero, W.: Verifying security protocols with brutus. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 9(4), 443–487 (2000)
30. Cremers, C.J.: The scyther tool: Verification, falsification, and analysis of security protocols. In: *International Conference on Computer Aided Verification*. pp. 414–418. Springer (2008)
31. De Ryck, P., Desmet, L., Joosen, W., Piessens, F.: Automatic and precise client-side protection against csrf attacks. In: *European Symposium on Research in Computer Security*. pp. 100–116. Springer (2011)

32. Dhamija, R., Tygar, J.D., Hearst, M.: Why phishing works. In: Proceedings of the SIGCHI conference on Human Factors in computing systems. pp. 581–590. ACM (2006)
33. Gordon, A.D., Pucella, R.: Validating a web service security abstraction by typing. *Formal Aspects of Computing* 17(3), 277–318 (2005)
34. Hardt, D.: The oauth 2.0 authorization framework (2012)
35. Jeremiah Grossman: Login Detection, whose problem is it? (March 2008), <http://blog.jeremiahgrossman.com/2008/03/login-detection-whose-problem-is-it.html>
36. Johns, M., Winter, J.: Requestrodeo: Client side protection against session riding. In: Proceedings of the OWASP Europe 2006 Conference (2006)
37. Krombholz, K., Hobel, H., Huber, M., Weippl, E.: Advanced social engineering attacks. *Journal of Information Security and applications* 22, 113–122 (2015)
38. Mao, Z., Li, N., Molloy, I.: Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In: *Financial Cryptography*. pp. 238–255. Springer (2009)
39. Oda, T., Wurster, G., van Oorschot, P.C., Somayaji, A.: Soma: Mutual approval for included content in web pages. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 89–98. ACM (2008)
40. PAUL WAGENSEIL: Lastpass can be spoofed in devastating phishing attacks. Article (Jan 2016), <https://www.tomsguide.com/us/lastpass-phishing-attacks,news-22139.html>
41. Recordon, D., Reed, D.: Openid 2.0: a platform for user-centric identity management. In: Proceedings of the second ACM workshop on Digital identity management. pp. 11–16. ACM (2006)
42. Robert Hansen and Jeremiah Grossman: Clickjacking. Blog (Dec 2008), <http://www.sectheory.com/clickjacking.htm>
43. Ruddy, M.: Decision point for federated identity and cross-domain single sign-on. Article (April 2015), <https://www.gartner.com/doc/3029229/decision-point-federated-identity-crossdomain>
44. Spett, Kevin: Cross-site scripting. *SPI Labs* 1, 1–20 (2005)
45. Steve Morgan: Cyber Crime Costs Projected To Reach 2 Trillion by 2019. Article (January 2016), <https://www.forbes.com/sites/stevemorgan/2016/01/17/cyber-crime-costs-projected-to-reach-2-trillion-by-2019/#3c5ecbfe3a91>
46. Telikicherla, K.C., Agrawall, A., Choppella, V.: A formal model of web security showing malicious cross origin requests and its mitigation using CORP. In: Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017. pp. 516–523 (2017), <https://doi.org/10.5220/0006261105160523>
47. Telikicherla, Krishna Chaitanya and Choppella, Venkatesh and Bezawada, Bruhadeshwar: CORP: A Browser Policy to Mitigate Web Infiltration Attacks. In: *International Conference on Information Systems Security*. pp. 277–297. Springer (2014)
48. Tom: Detect if visitors are logged into Twitter, Facebook or Google+ (Feb 2012), <http://www.tomanthony.co.uk/blog/detect-visitor-social-networks/>
49. W3C: History of the World Wide Web. Tech. rep. (1989), <http://www.w3.org/Consortium/facts#history>
50. Zalewski, Michal: Browser Security Handbook. Tech. rep. (2011), [https://code.google.com/p/browsersec/wiki/Part2#Same-origin\\_policy](https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy)