Cross-Site Request Forgery (CSRF) on JavaScript Object Notation (JSON) data

Garo Garabedyan,
TU-Sofia

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:

CSRF възможни пътища за атака

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

- Публикация във форум
- Уеб атака
- Мрежова атака

CSRF 1/3 атаки

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.

CSRF 2/3 атаки

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 удостоверационни header-и. Уеб страници, които разчитат на тези удостоверения може да бъдат объркани от такива заявки.

CSRF 3/3

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 своят регистър с cookie-та.

CSRF browser's server of origin

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.

CSRF browser's server of origin

Следващата таблица показва няколко достатъчни примера как „мисли“ моделът за сигурност на browser-а:

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

CSRF защита

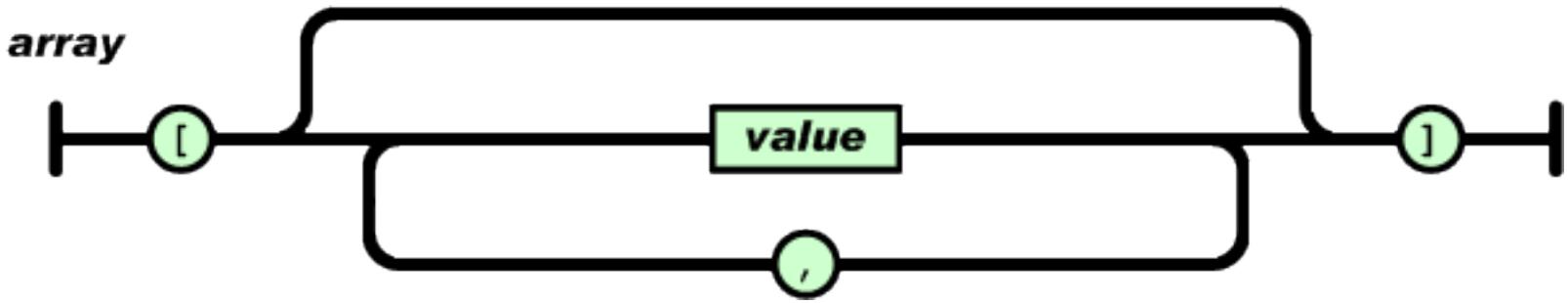
Има разнообразни начини за защита от CSRF, но най-общото е повторно изпращане на удостоверяващата автентичност заглавна част на заявката (в повечето случаи това е cookie) най-малко за всички заявки, които водят до промяна на състояние на потребителската сесия. В CSRF атакуващият няма достъп до оторизационните 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.

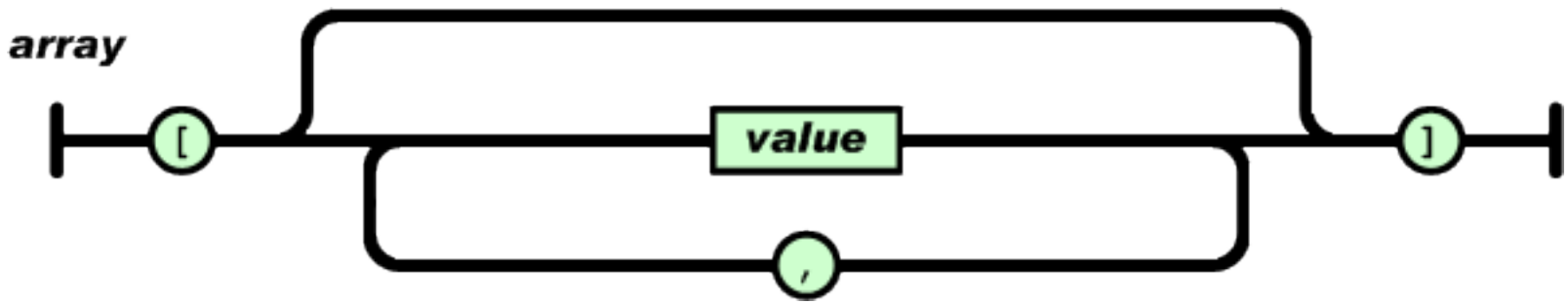
JavaScript

Demo

JSON

JavaScript Object Notation (JSON) е лек текстово базиран езиконезависим формат за обмяна на данни.





JSON vs. XML

```
{
  "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"
        }
      }
    }
  }
}
```

```
<!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>
```

CSRF JSON vulnerability

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.

CSRF JSON vulnerability

Demo

CSRF JSON vulnerability Заключение

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.