

NETRESEC | Products | Resources | Blog | About Netresec |

NETRESEC > Blog

Wednesday, 12 November 2014 21:09:00 (UTC/GMT)

Observing the Havex RAT

It has, so far, been publicly reported that three ICS vendors have spread the Havex Remote-Access-Tool (RAT) as part of their official downloads. We've covered the six pieces of software from these three vendors in our blog post "<u>Full Disclosure of Havex</u> <u>Trojans</u>". In this blog post we proceed by analyzing network traffic generated by Havex.



Indicators of Compromise

Before going into details of our analysis we'd like to recommend a few other resources that can be used to detect the Havex RAT. There are three <u>Havex IDS signatures</u> available via Emerging Threats. There are also <u>Yara rules</u> and <u>OpenIOC signatures</u> available for Havex. Additionally, the following domains are known to be used in the later versions (043 and 044) of Havex according to <u>Kaspersky</u>:

- disney.freesexycomics.com
- electroconf.xe0.ru
- rapidecharge.gigfa.com
- sinfulcelebs.freesexycomics.com
- www.iamnumber.com

HTTP Command-and-Control

The Havex RAT Command-and-Control (C2) protocol is based on HTTP POST requests, which typically look something like this:

POST /blogs/wp-content/plugins/buddypress/bp-settings/bpsettings-src.php? id=84651193834787196090098FD80-c8a7af419640516616c342b13efab&v1=043& v2=170393861&q=45474bca5c3a10c8e94e56543c2bd

As you can see, four variables are sent in the QueryString of this HTTP POST request; namely **id**, **v1**, **v2** and **q**. Let's take a closer look to see what data is actually sent to the C2 server in the QueryString.

Param	Description	Common Values		
id	host identifier	id=[random number][random hex]-c8a7af419640516616c342b13efab id=[random number][random-hex]-003f6dd097e6f392bd1928066eaa3		
vl	Havex version	043 044		
v2	Windows version	170393861 (windows XP) 498073862 (windows 7) 498139398 (windows 7, SP1)		
q	Unknown	q=45474bca5c3a10c8e94e56543c2bd (Havex 043) q=0c6256822b15510ebae07104f3152 (Havex 043) q=214fd4a8895e07611ab2dac9fae46 (Havex 044) q=35a37cab60b51a9ce61411a760075 (Havex 044)		

Analyzing a Havex PCAP

Recent Blog Posts

- » Observing the Havex RAT
- » Full Disclosure of Havex Trojans
- » Chinese MITM Attack on iCloud
- » Verifying Chinese MITM of Yahoo
- » Analysis of Chinese MITM on Google
- » Running NetworkMiner on Mac OS X
- » NetworkMiner 1.6 Released
- » PCAP or it didn't happen

Blog Archive

» 2014 November » 2014 October » 2014 September » 2014 June » 2014 May » 2014 April » 2014 March » 2014 Februarv » 2013 October » 2013 September » 2013 August » 2013 April » 2013 February » 2013 January » 2012 December » 2012 November » 2012 September » 2012 August » 2012 July » 2012 June » 2012 April » 2012 January » 2011 December » 2011 November » 2011 October » 2011 September » 2011 August » 2011 July » 2011 June » 2011 May » 2011 April » 2011 March » 2011 February » 2011 January List all blog posts

Grab our FeedBurner or RSS feed

11/17/2014

Observing the Havex RAT - NETRESEC Blog

I had the pleasure to discuss the Havex Malware with Joel Langill, when we met at the <u>4SICS conference</u> in Stockholm last month. Joel was nice enough to provide me with a 800 MB PCAP file from when he executed the Havex malware in an Internet connected lab environment.



Image: CapLoader transcript of Havex C2 traffic

I used the command line tool <u>NetworkMinerCLI</u> (in Linux) to automatically extract all HTTP downloads from Joel's PCAP file to disk. This way I also got a CSV log file with some useful metadata about the extracted files. Let's have a closer look at what was extracted:

\$ mono NetworkMinerCLI.exe -r new-round-09-setup.pcap Closing file handles 970167 frames parsed in 1337.807 seconds.				
\$ cut -d, -f 1,2,3,4,7,12	new-round-09-se	etup.pcap.Fi	leInfos.csv	head
SourceIP SourcePort Des 185.27.134.100 TCP 80 198.63.208.206 TCP 80 185.27.134.100 TCP 80 185.27.134.100 TCP 80 185.27.134.100 TCP 80 185.27.134.100 TCP 80 185.27.134.100 TCP 80 185.27.134.100 TCP 80 198.63.208.206 TCP 80	stinationIP Des 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121 192.168.1.121	stinationPor TCP 1238 TCP 1261 TCP 1286 TCP 1311 TCP 1329 TCP 1338 TCP 1346 TCP 1353 TCP 1365	t Filesize 244 676 B 150 B 359 508 B 236 648 B 150 B 150 B 150 B 150 B	Frame 14 1640 3079 4855 22953 94678 112417 130108 147902

Files downloaded through Havex C2 communication are typically modules to be executed. However, these modules are downloaded in a somewhat obfuscated format; in order to extract them one need to do the following:

- Base64 decode
- Decompress (bzip2)
- XOR with "1312312"

To be more specific, here's a crude one-liner that I used to calculate MD5 hashes of the downloaded modules:

\$ tail -c +95 C2_download.html | base64 -d | bzcat -d | xortool-xor -s "1312312" -f - -n | tail -c +330 | md5sum





Follow <u>@netresec</u> on twitter: » <u>twitter.com/netresec</u>



Recommended Books

» <u>The Practice of Network</u> <u>Security Monitoring</u>, Richard Bejtlich (2013)

» <u>Applied Network Security</u> <u>Monitoring</u>, Chris Sanders and Jason Smith (2013)

» <u>Network Forensics</u>, Sherri Davidoff and Jonathan Ham (2012)

» <u>The Tao of Network Security</u> <u>Monitoring</u>, Richard Bejtlich (2004)

» <u>Practical Packet Analysis</u>, Chris Sanders (2011)

» <u>Windows Forensic Analysis</u>, Harlan Carvey (2009)

» <u>TCP/IP Illustrated</u>, Volume 1, Kevin Fall and Richard Stevens (2011)

» Industrial Network Security, Eric D. Knapp and Joel Langill (2014) To summarize the output from this one-liner, here's a list of the downloaded modules in Joel's PCAP file:

First frame	Last frame	Downloaded HTML MD5	Extracted module MD5
14	293	7818cb3853eea675414480892ddfe668	7cff1403546eba915f1d7c023f12a0df
3079	1642	9b20948513a1a4ea77dc3fc808a5ebb9	840417d79736471c2f331550be993d79
4855	5117	fb46a96fdd53de1b8c5e9826d85d42d6	ba8da708b8784afd36c44bb5f1f436bc

All three extracted modules are known binaries associated with Havex. The third module is one of the Havex OPC scanner modules, let's have a look at what happens on the network after this module has been downloaded!

Analyzing Havex OPC Traffic

In Joel's PCAP file, the OPC module download finished at frame 5117. Less then a second later we see DCOM/MS RPC traffic. To understand this traffic we need to know how to interpret the UUID's used by MS RPC.

<u>Marion Marschalek</u> has listed <u>10 UUID's used by the Havex OPC module</u> in order to enumerate OPC components. However, we've only observed four of these commands actually being used by the Havex OPC scanner module. These commands are:

MS RPC UUID	OPC-DA Command
9dd0b56c-ad9e-43ee-8305-487f3188bf7a	IOPCServerList2
55c382c8-21c7-4e88-96c1-becfb1e3f483	IOPCEnumGUID
39c13a4d-011e-11d0-9675-0020afd8adb3	IOPCServer
39227004-a18f-4b57-8b0a-5235670f4468	IOPCBrowse

Of these commands the "IOPC Browse" is the ultimate goal for the Havex OPC scanner, since that's the command used to enumerate all OPC tags on an OPC server. Now, let's have a look at the PCAP file to see what OPC commands (i.e. UUID's) that have been issued.

<pre>\$ tshark -r new-round</pre>	d-09-setup.first6000.pcap -n -Y 'dcerpc.cn_bind_to_uuid
!= 99fcfec4-5260-101b	o-bbcb-00aa0021347a' -T fields -e frame.number -e ip.dst
-e dcerpc.cn_bind_to_	_uuid -Eoccurrence=f -Eheader=y
frame.nr ip.dst 5140 192.168.1.97 5145 192.168.1.11 5172 192.168.1.11 5193 192.168.1.11 5193 192.168.1.11 5212 192.168.1.11 5247 192.168.1.11 5257 192.168.1.11 5257 192.168.1.11 5269 192.168.1.11 5280 192.168.1.11 5285 192.168.1.11 5285 192.168.1.11 5285 192.168.1.11 5286 192.168.1.11 5286 192.168.1.11 5286 192.168.1.11	dcerpc.cn_bind_to_uuid 000001a0-0000-0000-c000-00000000046 000001a0-0000-0000-c000-00000000046 9dd0b56c-ad9e-43ee-8305-487f3188bf7a 00001a0-0000-0000-c000-0000000046 55c382c8-21c7-4e88-96c1-becfb1e3f483 00000143-0000-0000-c000-00000000046 00000143-0000-0000-c000-00000000046 00000143-0000-0000-c000-00000000046 00000143-0000-0000-c000-00000000046 00000143-0000-0000-c000-00000000046 00000143-0000-0000-c000-00000000046 39c13a4d-011e-11d0-9675-0020afd8adb3 39c27004-a18f-4b57-8b0a-5235670f4468 39227004-a18f-4b57-8b0a-5235670f4468

We can thereby verify that the IOPCBrowse command was sent to one of Joel's OPC servers in frame 5285 and 5286. However, tshark/Wireshark is not able to parse the list of OPC items (tags) that are returned from this function call. Also, in order to find all IOPCBrowse commands in a more effective way we'd like to search for the binary representation of this command with tools like <u>ngrep</u> or <u>CapLoader</u>. It would even be possible to generate an IDS signature for IOPCBrowse if we'd know what to look for.

The first part of an MSRPC UUID is typically sent in little endian, which means that the IOPCBrowse command is actually sent over the wire as:

04 70 22 39 8f al 57 4b 8b 0a 52 35 67 0f 44 68

Let's search for that value in Joel's PCAP file:

E Find Keyword	×
Find What	
Byte Sequence (hex)	
04 70 22 39 8f a1 57 4b 8b 0a 52 35 67 0f 44 68	
String (text)	
✓ Match <u>C</u> ase	
Text Encodings:	
ASCII CodePage_949	
CodePage_1251 CodePage_1251	
CodePage_1256 V QuotedPrintable CodePage_437 V Unicode	
CodePage_850 VIRL	
CodePage_936	
Find and Select All Matching Flows	Abort Search
Matching flows: 140	

Image: Searching for IOPCBrowse byte sequence with CapLoader

😘 Cap	bLoader 1.2					
Eile Edit Iools Help Input Settings File ID Filename Extracted flows on select Imput Settings Imput Settings Imput Settings Imput Settings Imput Settings File ID Filename Extracted Flows Show empty flows Imput Settings Flows: 169 Imput Settings Flows: 169 Filename: new-round-09-se Imput Settings Show countries Imput Settings Imput Settings Imput Settings Flows: 169 Imput Settings Flows: 169 Filename: new-round-09-se Imput Settings Imput Settings Flows: 169 Imput Se				vs on select :: 169 ime: new-round-09 53ECB00.pcap 810 541 346 B		
Hidde	en flows: 0 Hi	<u>d</u> e Selecteo	l Flows <u>I</u> nvert	Hiding	Show All Flows	
Flow	Client_IP	Client_Por	t Server_IP	Server	Port Protocol	TCP_Flags ^
200	192.168.1.97	137	192.168.1.121	137	NetBIOS Nam	
201	192.168.1.121	1312	192.168.1.97	135	MS RPC	APRS
202	192.168.1.121	1313	192.168.1.11	135	MS RPC	APRS
203	192.168.1.121	1314	192.168.1.97	135	MS RPC	AP SF
204	192.168.1.121	1315	192.168.1.11	135	MS RPC	AP SF
205	192.168.1.121	1317	192.168.1.97	135	MS RPC	AP SF
206	192.168.1.121	1318	192.168.1.11	49162	MS RPC	APRS
207	192.168.1.121	1319	192.168.1.97	135	MS RPC	AP SF
208	192.168.1.121	1320	192.168.1.11	135	MS RPC	AP SF
209	192.168.1.121	1321	192.168.1.11	49163	MS RPC	APRS
210	192.168.1.121	1322	192.168.1.11	49163	MS RPC	APRS 🝷
•						۱. E

Image: CapLoader with 169 extracted flows matching IOPCBrowse UUID

Apparently 169 flows contain one or several packets that match the IOPCBrowse UUID. Let's do a "<u>Flow Transcript</u>" and see if any OPC tags have been sent back to the Havex OPC scanner.

Observing the Havex RAT - NETRESEC Blog

C Transcript: 192.168.1.121:1323 -> 192.168.1.11:49210 TCP MS RPC
Client : 192.168.1.121 TCP 1323 Server : 192.168.1.11 TCP 49210 Statt Time : 2014-08-04 13:51:49.696667 UTC (15:51 GMT+02:00) End Time : 2014-08-04 13:52:01.091783 UTC (15:52 GMT+02:00) Duration : 00:00:11.3951160 Frames : 155 Protocol : MS RPC
Display Frames 100 - Encoding ASCII - Size 8
<pre>B.uc.k.e.t. B.r.i.g.a.d.e** *W.a.t.e.r.f.a.l.l.:.S.i.m.u.l.a.t.i.o.n. I.t.e.m.sB.u.c.k.e.t. B.r.i.g.a.d.e</pre>
.I.t.e.m.sB.u.c.k.e.tB.r.i.g.a.d.e

Image: CapLoader Transcript of OPC-DA session

Oh yes, the Havex OPC scanner sure received OPC tags from what appears to be a Waterfall unidirectional OPC gateway.

Another way to find scanned OPC tags is to search for a unique tag name, like "Bucket Brigade" in this example.

🚹 Share | 🗗 匡 👩 🝸

Short URL: <u>http://netresec.com/?b=14BE342</u>

Posted by Erik Hjelmvik on Wednesday, 12 November 2014 21:09:00 (UTC/GMT)

© 2010-2013 NETRESEC AB | Contact Us | Privacy | RSS