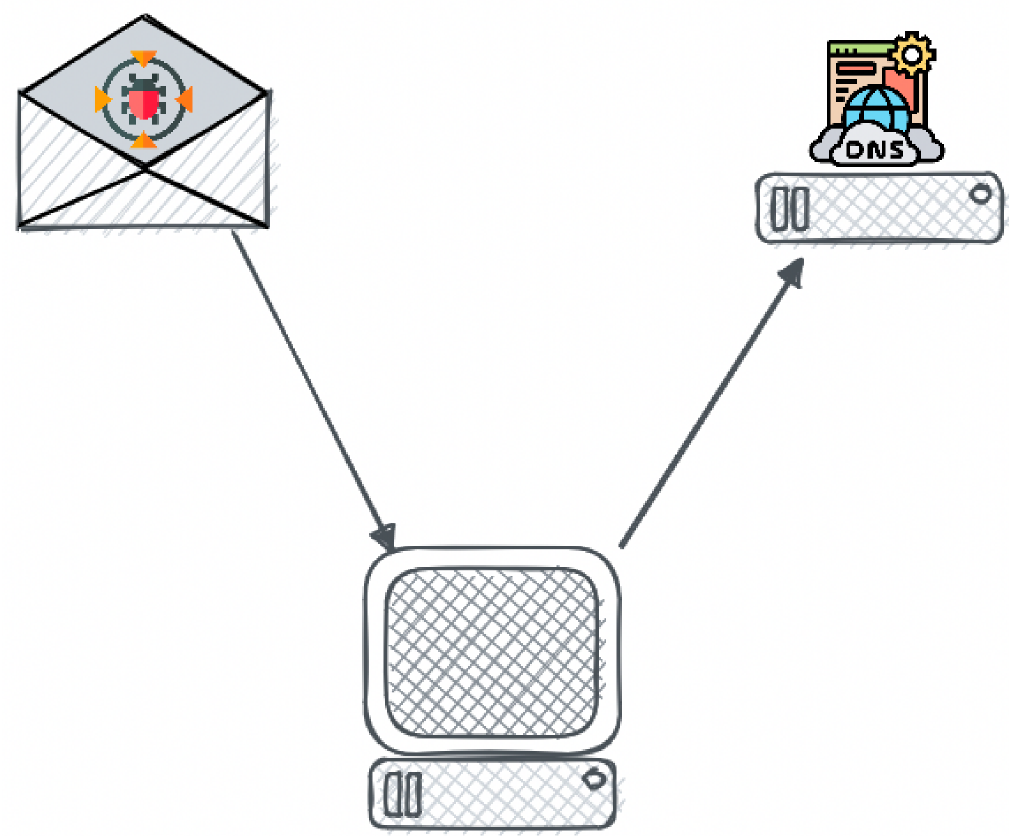
Trickbot

Anchor DNS Variant



Adam V. Link
@linkavych
2022-05-15

Table of Contents

Executive Summary

The Trickbot family of malware is a well-known and established cyber-crime group that has traditionally targeted the financial sector, but has gone well beyond that narrow focus. This campaign specifically uses the Domain Name System in order to communicate from victim systems to the command and control servers.

In the past, Trickbot has been distributed using phishing campaigns to target specific industries, or geographic areas. The campaigns have been widespread in order to continue to grow the botnet and the group's access to more systems. This Linux variant malware has been distributed via Zip files, and acts as an entrypoint for the execution of additional malware on the victim system. Importantly, this variant must be executed as the **root** super-user on a victim server in order to establish persistence.

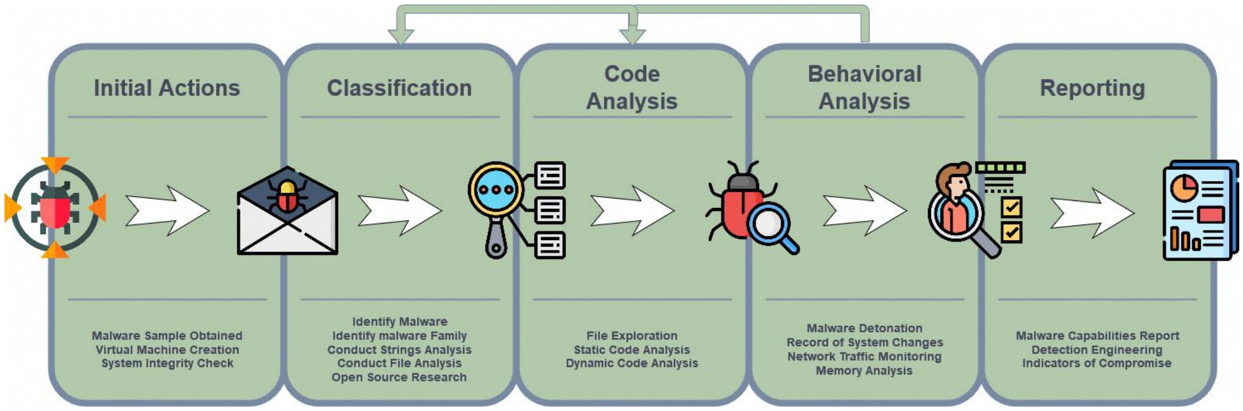
Once established, the malware will gather information about the victim system, including the public IP address, and then begin sending queries to a malicious server with encoded and encrypted data. The attacker can then send additional malware to the victim system for execution. This modularity provides flexibility for the attacker in accomplishing their objectives, but the hardcoded path for downloading and executing new malware provides an opportunity for detection and mitigation.

In targeting Linux systems, the Trickbot group is assuming these systems are less well-defended compared to a Windows Active Directory environment. Another gap exploited is enterprise network monitoring that may not be examining all DNS queries and responses for unusual data. From this position, the attacker can then move laterally within the victim network by downloading additional malware for execution. Anticipating that this may be their foothold within a larger enterprise environment, the attackers embedded a Windows executable within the Linux malware.

This additional executable is intended to be uploaded to exposed network file shares, and executed on Windows systems. This embedded executable is essentially the same as the Linux variant, and serves as a staging point for additional malware functionality to be uploaded into the network.

System owners should be actively monitoring their environments for unusual DNS communication patterns, and additional rules are provided in this report for detection and mitigation.

Methodology



In general, malware is analyzed according to a standard process[1]. Analysis begins with *Initial Actions* upon receipt of a malware sample. A virtual environment is designed and created for the malware sample, and memory snapshots are taken to document a known good state for the testing environment. The memory sample is disabled, compressed, and password protected before beginning the initial examination of the malware.

Next, *Classification* takes place. During this phase, initial analysis examines the file itself for any specific characteristics (e.g. file size), conducts string analysis on the sample, and makes an initial attempt to place the malware within a broader malware family. Virustotal or other open-source malware repositories are also consulted for additional information on the sample, if known. This phase is revisited as new information is revealed throughout the analysis process.

Code Analysis is the process of examining the decompiled (or source) code of the malware sample. Code is viewed within a decompiler, such as Ghidra or Binary Ninja, and examined for relevant structures and capabilities present in the sample. After static analysis, dynamic analysis is conducted within the context of a debugger so the analyst can examine the run time state of the sample.

Behavioral Analysis examines the state of the system as the malware is executed in the environment. Network traffic is captured, logs and registry hives are compared, and a memory snapshot is taken for further analysis.

Finally, the *Reporting* phase includes the development of useful detections for the malware sample, other indicators of compromise, and the drafting of the final capabilities report on the sample.

[1] Bermejo Higuera, J., Abad Aramburu, C., Bermejo Higuera, J., Sicilia Urban, M. and Sicilia Montalvo, J., 2020. Systematic Approach to Malware Analysis (SAMA). *Applied Sciences*, 10(4), p.1360.

File Characteristics

TrickBot - ELF - x86_64

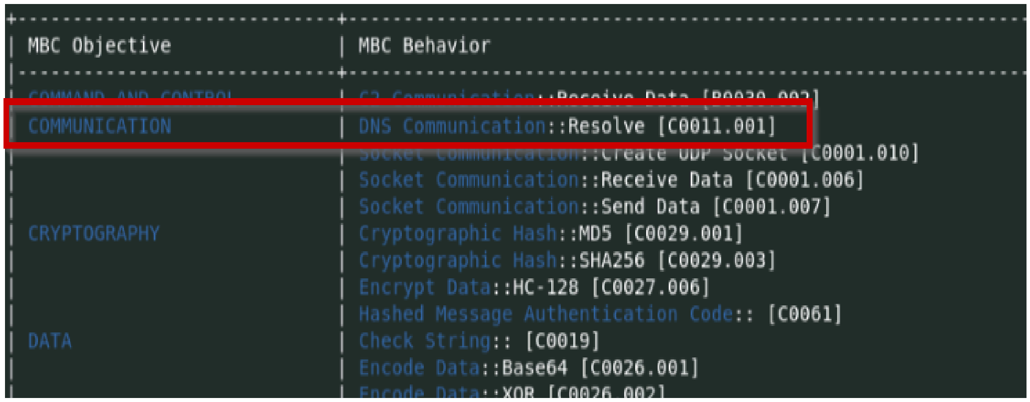
File Name	c721189a2b89cd279e9a033c93b8b5017dc165cba89eff5b8e1b5866195518bc
MD5	7d2595904aa6feb46b3e8f3262963042
SHA256	c721189a2b89cd279e9a033c93b8b5017dc165cba89eff5b8e1b5866195518b
TLSH Hash	T119F46A0776E214BEC1A2D474836BD172AD36B4241222BD7F76C4DA313E56E201F7EB62
File Size	782424 bytes
Compile Time	N/A (GNU GCC 9.2.0 and Debian 6.3.0)
Architecture	x86_64
Operating System	Linux
Format	ELF
Virustotal Score	35 of 62

Embedded PE File - x86

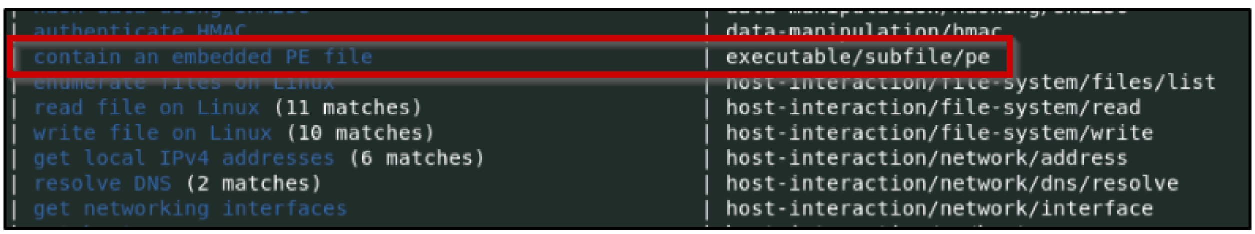
File Name	N/A
MD5	3A9F9CC4F0610AD974FF9251A567CAEF
SHA256	A69B6197AE512BEE4601F2E7494E675EC8C596EE175A453D2E6B16C0F2F1B3C7
TLSH Hash	T1F4543A4377E59C67E1217D708518E9E2AE6CF520038344BFBB8593147A6A1B18F3BA73
File Size	292024 bytes
Compile Time	0x5DFBAE33 (Thu Dec 19 17:06:59 2019 | UTC)
Architecture	X86 (32-bit)
Operating System	Windows
Format	PE
Virustotal Score	No Matches Reporting

Capabilities

Overview

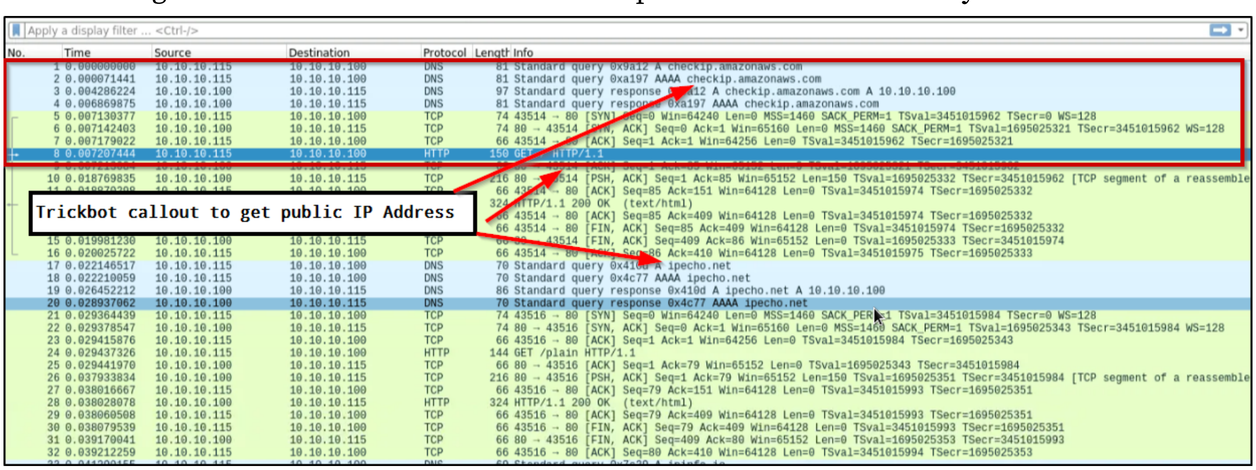


This sample of Trickbot is designed to be a downloader for further malware, and an initial backdoor to communicate with the C2 servers. Capa provides a detailed overview of the various capabilities found in the ELF malware sample. Importantly, capa highlights the use of DNS as a C2 communication path, and that a PE file is embedded within the ELF executable.



Persistence

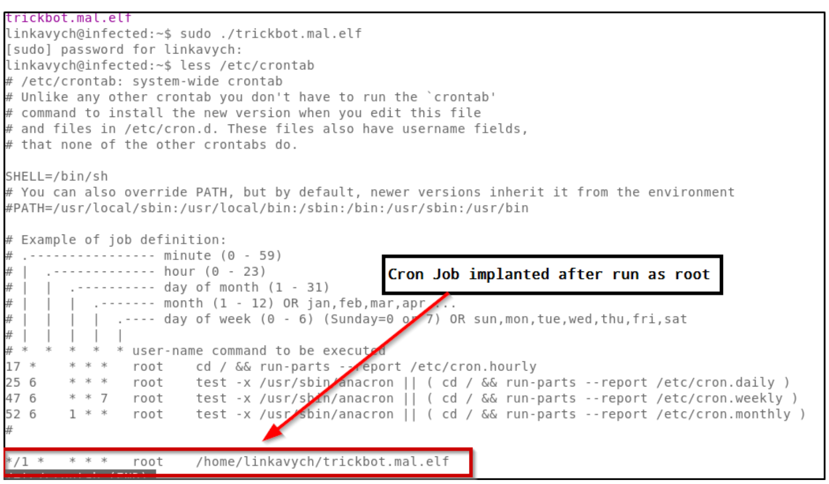
The first purpose of this malware is to create a backdoor onto the system for future access. To do this, the malware must be run as the root user on the victim machine. Once run, the malware begins its initial checks to establish the public IP address of the system.



Several IP address checking services are coded into the binary to get the public IP address of the victim. The malware will cycle through each of these domains until it confirms the public IP address and moves on to the next task.

IP Address Check Domains
hxxp://checkip[.]amazonaws[.]com
hxxp://api[.]ipify[.]org
hxxp://ipinfo[.]io/ip
hxxp://ipecho[.]net/plain
hxxp://ip[.]anysrc[.]net/plain/clientip
hxxp://wtfismyip[.]com/text
hxxp://myexternalip[.]com/raw
hxxp://icanhazip[.]com

After gathering the public IP address, the malware establishes persistence on the target by writing a task to the ***'/etc/crontab'*** file. This cron job will execute the malware on a recurring basis.



This cron job is not hidden by the malware, and should be easy for a defender to locate in the ***'/etc/crontab'*** file. A Sigma rule is provided in this report to aid detection as well.

Write and Execute

Once established on the victim, the malware provides the attacker with the ability to download files to the target and execute them. The function responsible for writing the files is below:

```
FILE_PATH = L'\x706d742f';
puVar5 = local_118;
for (lVar4 = 0x3c; lVar4 != 0; lVar4 = lVar4 + -1) {
  *puVar5 = 0;
  puVar5 = puVar5 + 1;
}
local_120 = 0;
local_160 = ZEXT816(0);
GET_RANDOM_NAME(local_160,0xf,0,1,0,0,0,0);
local_160 = local_160 & (undefined  [16])0xffffffffffffffff;
strcat((char *)&FILE_PATH,local_160);
__s = fopen((char *)&FILE_PATH,"w+b");
if (__s != (FILE *)0x0) {
  sVar3 = fwrite(BUFFER,1,SIZEOF(BUFFER),__s);
  if (SIZEOF(BUFFER) != sVar3) {
    FILE_PATH = FILE_PATH & 0xffffffffffffff00;
  }
  fclose(__s);
}
if ((char)FILE_PATH != '\0') {
  chmod((char *)&FILE_PATH,0x777);
  uVar1 = fork();
  if (uVar1 != 0xffffffff) {
    if (uVar1 == 0) {
      DAT_006bea84 = 1;
      _Var2 = setsid();
      if (_Var2 == -1) {
        return;
      }
                /* SEND STDOUT TO /dev/null */
      chdir("/");
      open("dev/null",2);
      dup(0);
      dup(0);
      if (param_1 != 10) {
        local_148[0] = (ulong *)0x0;
        local_150 = &FILE_PATH;
                /* EXECUTE THE TMP FILE */
        execve((char *)&FILE_PATH,(char **)&local_150,(char **)0x0);
        return;
      }
```

File path will be '/tmp/'

Generate random 15 character file name

Write bytes of the file to disk

Mark the file as executable, and fork the current process.

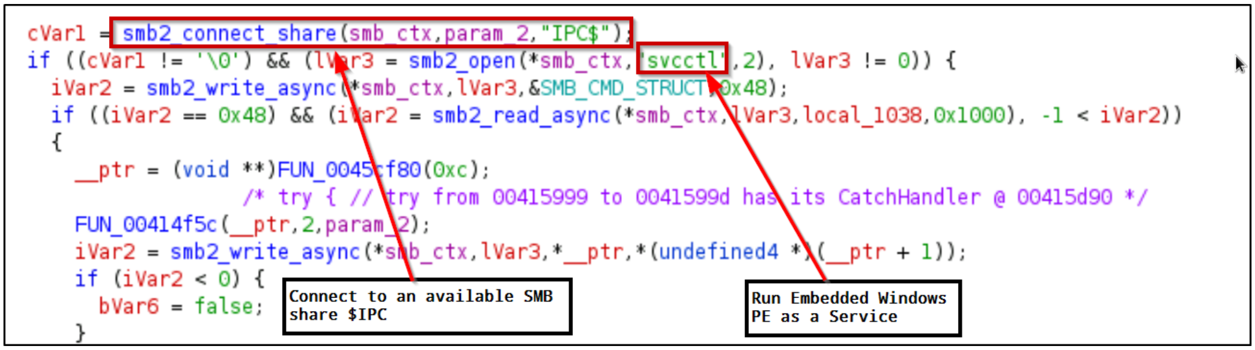
Execute the file on the victim.

This function allows the attackers to send additional malware to the target system, and execute the malware immediately. This increases the overall modularity of the approach Trickbot has taken in the development of their malware. It allows the attacker to choose additional payloads to execute against the target system depending on their objectives.

This function generates a random 16 character name within the ***/tmp*** directory, writes the file, and then immediately executes the file. Defenders should consider mounting ***/tmp*** with ***noexec*** mode enabled to deny this ability to the malware.

SMB Module

This sample also contains a module for connecting to a remote SMB share ($IPC), and executing an embedded Windows PE version of the malware using **svcctl.exe**.



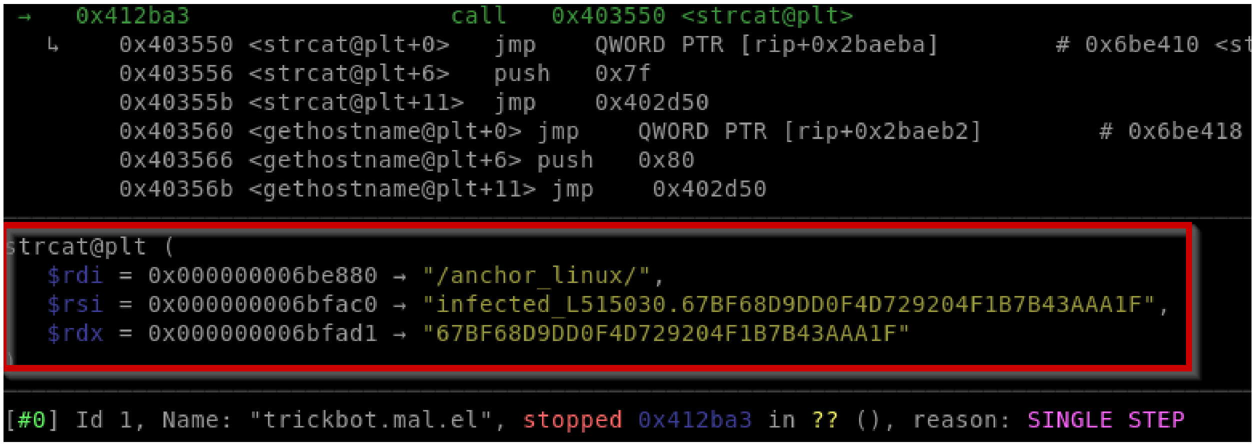
This module allows the malware to spread within a mixed environment of Windows and Linux systems. The PE version of this malware, embedded in the ELF, establishes the same backdoor and C2 protocol in order to spread within the network. The PE version of anchorDNS is not analyzed in this report.

DNS C2

By establishing communications over DNS, Trickbot is leveraging well known communications protocols which are not always well defended and monitored. When this malware was initially discovered, it was also a newer C2 channel for the Trickbot group . When the malware begins the main communications loop, just after checking for its public IP address, it builds a string containing some identifying information of the victim machine.

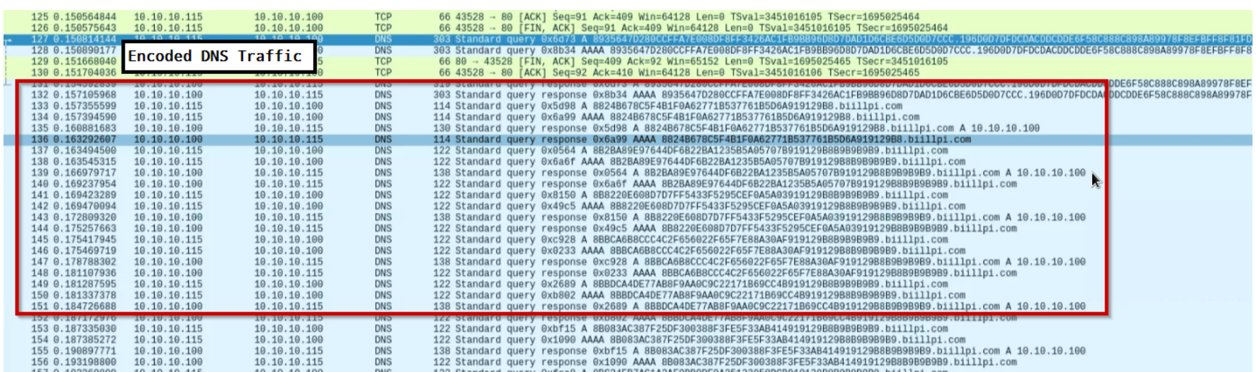
/anchor_linux/HOSTNAME_KERNEL.ID/

The "anchor_linux" string is hardcoded into the binary and found when using strings. The malware then executes the **"uname"** twice, each time with a different flag: "**-n**" printing the network node (hostname), and "**-r**" printing the kernel release date information. This information is then concatenated with the hardcoded **id** string. On the analysis system, this string resulted in the below:



Once this is complete, the eventual payload and the public IP address of the victim are appended to the end of this string to send back to the C2 servers. This information is further encrypted and XOR encoded prior to sending the data.

The data is sent as a series of DNS queries to a single domain (in the case of this sample). No live communication was captured for this analysis, but an example of the pattern is show below from a contained lab environment.



The malware will continue to listen from DNS responses which would contain encrypted and encoded commands for execution. Again, the main purpose of this sample appears to be to act as a dropper for additional malware, and establish the C2 channel for follow on attacks from the Trickbot group.

For additional analysis on the DNS communication structure, I recommend the following reports:

1. Anchor_dns malware goes cross platform
2. Trickbot variant "Anchor_DNS" communicating over DNS

In particular, the second report examines a live sample in active communication, but from a Windows platform.

Actionable Artifacts

File Hashes

Version	SHA256 Hash
Linux - Anchor DNS	c721189a2b89cd279e9a033c93b8b5017dc165cba89eff5b8e1b5866195518b
Windows - Anchor DNS	A69B6197AE512BEE4601F2E7494E675EC8C596EE175A453D2E6B16C02F1B3C7

Strings

```
http://checkip.amazonaws.com
http://ipecho.net/plain
http://ipinfo.io/ip
http://api.ipify.org
http://icanhazip.com
http://myexternalip.com/raw
http://wtfismyip.com/text
http://ip.anysrc.net/plain/clientip
https://checkip.amazonaws.com
https://ipecho.net/plain
https://ipinfo.io/ip
https://api.ipify.org
https://icanhazip.com
https://myexternalip.com/raw
https://wtfismyip.com/text
https://ip.anysrc.net/plain/clientip
/C timeout 5 && %ssc.exe stop %S
/C timeout 10 && %ssc.exe delete %S
/C timeout 15 && del %S
SYSTEM\CurrentControlSet\Services
*/1 *   * * *   root
/etc/crontab
/proc/%s/cmdline
/tmp/anchor.log
--debuglevel=
L0000000
/var/lib/libuuid/clock.txt
GCC: (GNU) 9.2.0
GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516
```

Sigma Rules

```yaml
title: anchorDNS Tunneling
description: Detect potential DNS tunneling based on volume of queries to a given domain
author: Adam Link
date: 2022/05/16
logsource:
  category: DNS
detection:
  selection:
    query|contains: '.biilpi.com'
falsepositives:
  - Unknown
level: high
```

```yaml
title: anchorDNS - File Create and Execute
description: Detect a 15 character file created by anchorDNS malware
author: Adam Link
date: 2022/05/16
logsource:
    product: linux
    category: file_create
detection:
    selection:
        TargetFilename|re: ^[a-zA-Z0-9]{1,15}$
    condition: selection
falsepositives:
    - Unknown
level: high
```

```yaml
title: Cron Files
id: 6c4e2f43-d94d-4ead-b64d-97e53fa2bd05
status: experimental
description: Detects creation of cron files or files in Cron directories.
Potential persistence.
date: 2021/10/15
author: Roberto Rodriguez (Cyb3rWard0g), OTR (Open Threat Research), MSTIC
tags:
    - attack.persistence
    - attack.t1053.003
references:
   -
https://github.com/microsoft/MSTIC-Sysmon/blob/main/linux/configs/attack-ba
sed/persistence/T1053.003_Cron_Activity.xml
logsource:
   product: linux
   category: file_create
detection:
   selection1:
      TargetFilename|startswith:
         - '/etc/cron.d/'
         - '/etc/cron.daily/'
         - '/etc/cron.hourly/'
         - '/etc/cron.monthly/'
         - '/etc/cron.weekly/'
         - '/var/spool/cron/crontabs/'
   selection2:
      TargetFilename|contains:
         - '/etc/cron.allow'
         - '/etc/cron.deny'
         - '/etc/crontab'
   condition: selection1 or selection2
falsepositives:
   - Any legitimate cron file.
level: medium
```

Yara Rule

```
rule trickbot_mal {
   meta:
      description = "anchorDNS - file trickbot.mal.elf"
      author = "@linkavych"
      date = "2022-05-16"
      hash1 =
"c721189a2b89cd279e9a033c93b8b5017dc165cba89eff5b8e1b5866195518bc"
   strings:
      $x1 = "Failed to parse fixed part of command payload. %s" fullword
ascii
      $x2 = "Failed to parse variable part of command payload. %s" fullword
ascii
      $x3 = "/tmp/anchor.log" fullword ascii
      $x4 = “/etc/crontab” fullword ascii
      $s1 = "curity><requestedPrivileges><requestedExecutionLevel
level=\"asInvoker\"
uiAccess=\"false\"></requestedExecutionLevel></requeste" ascii
      $s2 = "https://checkip.amazonaws.com" fullword ascii
      $s3 = "http://checkip.amazonaws.com" fullword ascii
      $s4 = "/C timeout 5 && %ssc.exe stop %S " fullword ascii
      $s5 = "Failed to create read command" fullword ascii
      $s6 = "No more connections allowed to host %s: %zu" fullword ascii
      $s7 = "http://icanhazip.com" fullword ascii
      $s8 = "https://wtfismyip.com/text" fullword ascii
      $s9 = "https://icanhazip.com" fullword ascii
      $s10 = "https://myexternalip.com/raw" fullword ascii
      $s11 = "http://wtfismyip.com/text" fullword ascii
      $s12 = "/C timeout 10 && %ssc.exe delete %S" fullword ascii
      $s13 = "http://myexternalip.com/raw" fullword ascii
      $s14 = "RESOLVE %s:%d is - old addresses discarded!" fullword ascii
   condition:
      uint16(0) == 0x457f and filesize < 2000KB and
      (
          ( 2 of ($x*) and all of ($s*)) or ( all of them )
      )
}
```

Snort and Suricata Rules

```
# snort - specific rule for anchorDNS domain
alert udp any any -> any 53 (content:"biilpi.com"; nocase;priority:1;
msg:"potential anchorDNS C2 traffic"; classtype:string-detect; sid 1000001;
gid:1; rev:1; )

# snort generic DNS query rule for large queries
alert udp any any -> any 53 (msg:"DNS query larger than 100 bytes"; dsize:>
100; sid:12345;)

# suricata - specific rule for anchor DNS domain
alert dns any any -> any 53 (msg:"dns query to known ANCHOR_DNS domain;
possible C2"; dns.query; content:"biilpi.com";nocase;sid:1;)
```

References

- https://medium.com/stage-2-security/anchor-dns-malware-family-goes-cross-platform-d807ba13ca30
- https://services.global.ntt/zh-cn/insights/blog/trickbot-variant-communicating-over-dns
- https://www.cybereason.com/blog/research/dropping-anchor-from-a-trickbot-infection-to-the-discovery-of-the-anchor-malware
- https://www.netscout.com/blog/asert/dropping-anchor