Threat Hunting via DNS

Eric Conrad (econrad@backshore.net)
https://ericconrad.com
Twitter: @eric_conrad
CIS 8.7: Malware Defenses

- Enable Domain Name System (DNS) query logging to detect hostname lookups for known malicious domains.¹
- DNS logs are one of the most actionable threat hunting/SOC/SIEM data sources
- In addition to logging, viewing/dumping and inspecting the DNS cache is a good short-term investigative tool
- Note that DNS may be logged on the DNS server or endpoints (CSM), or sniffed on the network using tools like Zeek (NSM)
  - Encrypted DNS is impacting both, as we will discuss shortly
DNS Logging Options

• Point internal clients to a local DNS server and log there
• Log on the clients (for example: using Sysmon)
• Use Zeek to sniff DNS packets off the network
  o Normally one of the best options, but DNS encryption is rapidly changing this
DNS Encryption

• A big trend on the encryption front is impacting a vital analytics source: DNS queries

• DNS query encryption concerns itself primarily with increasing the privacy of users' communications
  o This dovetails nicely with the push toward ubiquitous HTTPS from a traffic privacy perspective

• Inscrutable DNS queries can pose secops challenges:
  o Blindness to adversaries' intentional use of DNS
  o Diminished user monitoring/analytic capabilities
Facing Reality

• This talk will not debate the merits of encrypted DNS vs. plaintext DNS via UDP/TCP port 53 (now called Do53)
  o Encrypted DNS provides more privacy to the end user
    • Ultimately, most encrypted DNS will resolve via Do53 somewhere upstream
  o Do53 provides easy centralized monitoring for companies
    • And easy monetization for ISPs

• Years of network defense have taught me to be a realist

• DNS over HTTPS (DoH) is coming on like a freight train
  o Network defenders need to prepare accordingly
DNS over HTTPS (DoH) and DNS over TLS (DoT)

- DNS over HTTPS (DoH) and DNS over TLS (DoT) are impacting the ability to monitor DNS queries
  - This is true for Intrusion Detection Systems such as Zeek, as well as logging requests on the local DNS resolver/forwarder
- DNS over HTTPS uses TCP port 443 and is normal HTTPS traffic from a network perspective
- DNS over TLS uses TCP port 853, so network operators/defenders know that it’s (encrypted) DNS traffic
  - DoT can be easily blocked by a firewall, forcing resolution back to Do53
- Analyzing the content on the wire requires SSL/TLS interception
The Only Constant is Change

• This talk with track DoH in Firefox most closely
  o Firefox is the currently the most aggressive browser regarding adopting DNS encryption
• DoH/DoT adoption is evolving very rapidly
• I will track updates on https://ericconrad.com
• Jim Troutman’s 2020 Shmoocon Firetalk is fantastic:
  o http://www.nepeeringforum.org/troutman/troutman-DoH-DoT-QuadX-Da-Faq.pdf
DNS over TLS (DoT)

RFC 7858\(^1\) defines a means of sending DNS over TLS

- Specifies TLS 1.2, but some implementations support TLS 1.3

Explicitly uses TCP Port 853

- However, RFC allows nonstandard ports if clients/servers agree to leverage one (e.g. malware implants)\(^2\)

Advantages:

- Users - Increased privacy and integrity
- Analysts - Easy to detect via TCP:853...just not to analyze
- Architects - Easy to block default outbound port of TCP:853
DoH and DoT

• The early trend: browsers tend to support DNS over HTTPS (for resolution within the browser), while Linux operating systems tend to support DNS over TLS for default operating system resolution
  o DNS over TLS is now used by default by Android (called “Private DNS Mode”)
• Firefox and Chrome now support DNS over HTTPS
• Microsoft recently announced plans to support DoH in Windows 10
• In the short-term DoH is “winning”
Paul Vixie on DoH

Rfc 8484 is a cluster duck for internet security. Sorry to rain on your parade. The inmates have taken over the asylum.

5:49 PM · Oct 20, 2018 · Twitter Web App

Paul Vixie @paulvixie · Oct 21, 2018

DoH is an over the top bypass of enterprise and other private networks. But DNS is part of the control plane, and network operators must be able to monitor and filter it. Use DoT, never DoH.

💬 3  ⬇️ 15  ❤️ 63
Firefox Has Begun Enabling DoH by default in the U.S.

Firefox continues push to bring DNS over HTTPS by default for US users

Selena Deckelmann  |  February 25, 2020

Today, Firefox began the rollout of encrypted DNS over HTTPS (DoH) by default for US-based users. The rollout will continue over the next few weeks to confirm no major issues are discovered as this new protocol is enabled for Firefox’s US-based users.

https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/
Firefox/DoH

• Firefox bypasses the local system DNS settings when using DoH, and sets the DNS provider to Cloudflare by default
  o Other options include NextDNS and Custom (discussed next)
• To disable DoH, go to Settings -> Network Settings -> Connection settings, and uncheck "Enable DNS over HTTPS"
Firefox/DoH Options

• If you're not currently logging DNS: continue as you were
• Another Firefox blocking option: prevent this canary domain name from resolving:
  o use-application-dns.net
• Or allow DoH and:
  o Enable logging within Firefox locally (discussed next)
  o Set up a custom DoH server and log there
Logging DNS Resolution Locally in Firefox

• Type "about:networking" in the URL bar
• Go to the "Logging section" and set "nsHostResolver:1" under "Set Log Modules"
  o Default is "nsHostResolver:5" (quite verbose)
  o Consider removing other modules to reduce log volume
• Press “Start Logging”

See: https://developer.mozilla.org/en-US/docs/Mozilla/Debugging/HTTP_logging
Trigger Warning: Firefox DNS Logging is Not Pretty

[Parent 5643: DNS Resolver #7]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [incoming.telemetry.mozilla.org].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [dns.qeze.me].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [dns.qeze.me]: success.
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [static.ads.twitter.com].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [static.ads.twitter.com]: success.
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [bat.bing.com].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [stats.g.doubleclick.net].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [stats.g.doubleclick.net]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [static.hotjar.com].
[Parent 5643: DNS Resolver #19]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [analytics.twitter.com].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [static.hotjar.com]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [analytics.twitter.com].
[Parent 5643: DNS Resolver #19]: E/nsHostResolver DNS lookup thread - lookup completed for host [analytics.twitter.com]: success.
[Parent 5643: DNS Resolver #7]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [cdn.segment.com].
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [cdn.segment.com]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [cdn.segment.com]: success.
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [cdn.bloomtrain.com].
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [cdn.bloomtrain.com]: success.
[Parent 5643: DNS Resolver #19]: E/nsHostResolver DNS lookup thread - lookup completed for host [cdn.bloomtrain.com]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [js-sec.indexww.com].
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [dpm.demdex.net].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [dpm.demdex.net]: success.
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [data.cnn.com].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [match.adsrvr.org]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [w.usabilla.com].
[Parent 5643: DNS Resolver #8]: E/nsHostResolver DNS lookup thread - calling getaddrinfo for host [api.segment.io].
[Parent 5643: DNS Resolver #17]: E/nsHostResolver DNS lookup thread - lookup completed for host [api.segment.io]: success.
[Parent 5643: DNS Resolver #18]: E/nsHostResolver DNS lookup thread - lookup completed for host [api.segment.io]: success.
Chrome/DoH

• Firefox makes the DoH provider Cloudflare (regardless of the system’s previous DNS settings)
  o This has proven quite controversial
  o Chrome uses a different approach

• If the system is using a provider on this list for DNS resolution, Chrome will "auto-upgrade" Do53 to DoH, and keep the same provider:
  o Cleanbrowsing, Cloudflare, Comcast, DNS.SB, Google, OpenDNS, Quad9
  o Otherwise: Chrome will continue using regular DNS, and the existing provider

• This change began rolling out in late 2019
  o “We've enabled an experiment in Chrome 79 for a fraction of our users.”
Setting up your own DoH server

• This guide is excellent
• Instructions for Ubuntu 18.04
  o Also has sections on setting up PiHole and DoT
  o I was able to set up a DoH server in Digital Ocean’s cloud in <10 minutes

• https://www.aafalo.me/2018/10/tutorial-setup-dns-over-https-server/
DoH is a Web Application

• DoH is just a web application accessed HTTPS
  o Uses web servers such as Nginx and Apache, leverages x.509 certs, etc.

• For example:
  o https://dns.zez.me – regular HTTPS site
  o https://dns.zez.me/dns-query - resolves DoH requests via a POST

```
[root@DoH2:/var/log/nginx#  tail dns.access.log
198.255.243.192 -- [27/Feb/2020:17:47:43 +0000] "GET / HTTP/1.1" 200 108 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:73.0) Gecko/20100101 Firefox/73.0"
198.255.243.192 -- [27/Feb/2020:17:47:43 +0000] "GET /doh.jpg HTTP/1.1" 200 38572 "https://dns.zez.me/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:73.0) Gecko/20100101 Firefox/73.0"
```
Network-based DoH prevention

• DoH can be prevented via client configuration
• Network-based DoH prevention (such as firewalling) isn’t practical, short of SSL/TLS proxying and inspection
• HTTPS access to **known** DoH resolvers can be blocked
  o 1.1.1.1:443, 8.8.8.8:443, etc.
• HTTPS access to **unknown** DoH resolvers cannot be easily blocked
  o 206.189.185.210:443 (my custom DoH server)
Network-based DoH Detection

• Detect known DoH resolvers via simple IP/port-based IDS rules (1.1.1.1:443, etc.)

• Beacons detection can detect DoH, including unknown resolvers
  o A browser typically connects to the same DoH resolver (HTTPS site) thousands+ times/day

• RITA is a great tool for detecting beaconsing
  o https://www.blackhillsinfosec.com/projects/rita/

• Check out SANS STI student Drew Hjelm's paper: A New Needle and Haystack: Detecting DNS over HTTPS Usage
  o https://www.sans.org/reading-room/whitepapers/dns/paper/39160
DNS Logging via Sysmon

- Microsoft’s Sysmon now supports logging of local DNS queries
- Plays nicely with centralized event collection via Windows Event Forwarding
- Killer threat hunting feature: it shows the client application that made the DNS request
- Note that Firefox’ DoH implementation bypasses the local resolving entirely
  - Sysmon does not log Firefox’s DoH DNS requests
Sysmon DNS Logging Example

Image:

C:\WINDOWS\SYSTEM32\PING.EXE
Malware, like most network software, uses DNS for resolving names to IP addresses (and so on)

It also uses DNS for command and control (C2) traffic
- It’s usually allowed outbound
- It’s usually ignored

The following should be monitored:
- Requests to thousands of hosts or subdomains in one domain
- Large DNS queries with high entropy
- Large TXT record responses
- High volumes of DNS resolution failures
- Domain names that are very similar to yours
- DNS requests to domains registered very recently ("baby domains")
Zeus Botnet C2 via DNS

Note the large DNS TXT records used by the Zeus botnet for Command and Control (C2):

Non-authoritative answer:

12192 pf.zonesenoz.com       text =

"52g/s93XtdsK/b4lyx5iY3yjEkY80e17UgY9QYsv9XhTrl29e9eLpK1fg5b9/hMPnKcZojcPOtbHY8i
Rm6ZqldS6UOvTkua5rUzvv2u39bE5+OcdtCc5i2iGSr7COzxfd08DuS8Sdii22Y+OUT2wy/0Z2vFYptQ
76FUBX3Ml6fXZNrXuk01owePv7pdYwcXfGQyb9Fhr5aFo25zbn+2gaR3fsMOy"
DNS: the Ideal C2 Channel

• DNS tunnels are the ideal C2 channel, IMO
  o DNS is usually allowed outbound
  o It’s usually ignored
  o Works via multiple forwarders (i.e. DNS proxies)
  o Locked down internal subnets with 'no internet access' often allow public DNS resolution

• An internal system has direct bidirectional internet access if it can resolve 'google.com' and receive the answer

• DNS tunnels are much more difficult to mitigate via preventive controls
Iodine: Advanced DNS Tunneling

• Iodine offers a true routable tunnel via DNS
  o Can tunnel any IPv4 protocol
  o Quite easy to set up, and NIDS detection is poor

• Available at: http://code.kryo.se/iodine/

• Can forward via a local DNS server, or...
  o *it may also happen that _any_ traffic is allowed to the DNS port (53 UDP) of any computer. Iodine will detect this, and switch to raw UDP tunneling if possible.*
Iodine Wireshark View – DNS Tunnel (Forwarded via Local Resolver)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td>175</td>
<td>query response 6x7t26 NULL zovcaA-Aaahh-Drink-mal-ein-Jgermeister-.-J.3.eej.me</td>
</tr>
<tr>
<td>DNS</td>
<td>130</td>
<td>query 6x86db NULL zovdaA-La-fl\373te-na\357ve-fran\347aise-retir\351\346-Cr\350te.3.eej.me</td>
</tr>
<tr>
<td>DNS</td>
<td>193</td>
<td>query response 6x86db NULL zovdaA-La-fl\373te-na\357ve-fran\347aise-retir\351\346-Cr\350te.3.eej.me</td>
</tr>
<tr>
<td>DNS</td>
<td>136</td>
<td>query 6xdf18 NULL zovcaAbBcCdDeEffGhHIJjKkLmMnNoOpPQrRstTuUvWwxxyYzZ.3.eej.me</td>
</tr>
<tr>
<td>DNS</td>
<td>285</td>
<td>query response 6xdf18 NULL zovcaAbBcCdDeEffGhHIJjKkLmMnNoOpPQrRstTuUvWwxxyYzZ.3.eej.me</td>
</tr>
<tr>
<td>DNS</td>
<td>116</td>
<td>query 6x8442 NULL zovfaA0123456789\274\275\276\277\300\301\302\303\304\305\306\307\310\311\312\313\314\315\316\317</td>
</tr>
</tbody>
</table>
Zeek View – sort, sed, etc...

```
$ cat dns.log | zeek-cut query | sort -u | sed "s/^[a-zA-Z0-9-]*$/./g" | sort | uniq -c | sort -n
```
Programmatic Entropy Analysis

• Without trying, the human brain often can detect something as potentially random generated
  o Programmatically achieving this proves more difficult than expected
• Many tools exist for calculating entropy, the often built-in Linux tool, **ent** being a simple example
• Classic entropy analysis using tools like **ent** can be leveraged to determine the degree of randomness of provided input...
  o ...but ASCII has 256 characters
  o A DNS name containing letters (26 characters) and numbers (10 characters) uses a maximum of 36 of 256 total ASCII values (14%)
  o Any cryptologist will tell you: that equals low entropy
Bring Out the Baggett

- Solving problems like detecting random (before morning break) is why you always have @MarkBaggett (GSE #15) take your classes
  - freq.py tool is a huge boon to finding random generated strings where they perhaps shouldn't be
  - https://github.com/sans-blue-team/freq.py

- The approach looks at the likelihood of character occurrence based on frequency analysis
  - Simple example: in English text, “q” is pretty much followed by a “u,” so seeing a “q” followed by something else would be rather unlikely to occur
Domain Generation Algorithms DGAs

- One of the most obvious, and incredibly useful, ways to employ `freq.py` is looking at DNS names for signs of randomness

- You will necessarily need to do whitelisting
  - Public CDNs (Content Delivery Networks)
  - Major cloud services (Microsoft, Amazon, Google) often have their own CDN

<table>
<thead>
<tr>
<th>Proto</th>
<th>Len</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td>73</td>
<td>Standard query 0xc0b7 A olyedawaki.pl</td>
</tr>
<tr>
<td>DNS</td>
<td>73</td>
<td>Standard query response 0xc0b7 No such name A olyedawaki.pl</td>
</tr>
<tr>
<td>DNS</td>
<td>72</td>
<td>Standard query 0x6e61 A uydvrqwgg.su</td>
</tr>
<tr>
<td>DNS</td>
<td>72</td>
<td>Standard query response 0x6e61 No such name A uydvrqwgg.su</td>
</tr>
<tr>
<td>DNS</td>
<td>71</td>
<td>Standard query 0x7d3d A udfaexci.ru</td>
</tr>
<tr>
<td>DNS</td>
<td>71</td>
<td>Standard query response 0x7d3d No such name A udfaexci.ru</td>
</tr>
<tr>
<td>DNS</td>
<td>78</td>
<td>Standard query 0xd06c A 1kdcjcyjtpsc.work</td>
</tr>
<tr>
<td>DNS</td>
<td>78</td>
<td>Standard query response 0xd06c No such name A 1kdcjcyjtpsc.work</td>
</tr>
<tr>
<td>DNS</td>
<td>74</td>
<td>Standard query 0x4f67 A mrjuvawlwa.xyz</td>
</tr>
<tr>
<td>DNS</td>
<td>74</td>
<td>Standard query response 0x4f67 No such name A mrjuvawlwa.xyz</td>
</tr>
<tr>
<td>DNS</td>
<td>77</td>
<td>Standard query 0x5e78 A owvtbqledaraqq.su</td>
</tr>
<tr>
<td>DNS</td>
<td>77</td>
<td>Standard query response 0x5e78 No such name A owvtbqledaraqq.su</td>
</tr>
<tr>
<td>DNS</td>
<td>80</td>
<td>Standard query 0x6660 A uxfuskfqxhydqawmF.su</td>
</tr>
<tr>
<td>DNS</td>
<td>80</td>
<td>Standard query response 0x6660 No such name A uxfuskfqxhydqawmF.su</td>
</tr>
<tr>
<td>DNS</td>
<td>79</td>
<td>Standard query 0x7bd9 A osxbymbjwudtd.click</td>
</tr>
<tr>
<td>DNS</td>
<td>79</td>
<td>Standard query response 0x7bd9 No such name A osxbymbjwudtd.click</td>
</tr>
<tr>
<td>DNS</td>
<td>71</td>
<td>Standard query 0x2bf A wrbwttcvc.su</td>
</tr>
<tr>
<td>DNS</td>
<td>71</td>
<td>Standard query response 0x2bf No such name A wrbwttcvc.su</td>
</tr>
<tr>
<td>DNS</td>
<td>78</td>
<td>Standard query 0xea2f A uwiyk1nltlpxj.work</td>
</tr>
<tr>
<td>DNS</td>
<td>78</td>
<td>Standard query response 0xea2f No such name A uwiyk1nltlpxj.work</td>
</tr>
<tr>
<td>DNS</td>
<td>70</td>
<td>Standard query 0xc660 A eabfhwl.ru</td>
</tr>
<tr>
<td>DNS</td>
<td>78</td>
<td>Standard query response 0xea2f No such name A uwiyk1nltlpxj.work</td>
</tr>
</tbody>
</table>
Though DGA detection can be very effective, think more broadly about places where adversaries might programmatically generate large volumes of data.

Detecting randomness can be a tremendous indicator of otherwise unknown malice.

- Thread/Process names
- File names (binaries, scripts, etc.)
- Workstation names
- Service names
- Subdomains (Domain Shadowing\(^1\))
- Certificate subject names and issuers
- Usernames
- Many additional possibilities
freq_server.py - freq-ing At Scale

As additional use cases are discovered, you will soon feel the need to wield freq.py at scale.

Although the initial script is, without question, a work of art, it was not intended to have a system perform 100,000+ freq.py/sec.

Have no fear, @MarkBaggett worked with SANS SIEM course author and 511 instructor Justin Henderson (@SecurityMapper, GSE #108, SANS SIEM Author) and developed a new feature/deployment model.

• freq_server.py – https://github.org/sans-blue-team/freq.py/

• freq_server.py designed to allow for remote calls from tools such as LogStash

• Implementation and analysis techniques discussed in SANS SIEM class.
• Use **dnstwist** to protect against cousin domains (**sec530.com** vs. **sec530.com**) and Internationalized Domain Name (IDN) homograph attacks
  - Block with firewall/proxy, or detect via DNS and other sources
  - dnstwist calculates permutations against a given domain
  - Also checks to see if any domains have been registered
  - And provides additional information about the domain

• Use **dnstwist** with scripting to handle evil cousins and homographs
domain_stats

- Domain_stats is another great tool by Mark Baggett
  - https://github.com/MarkBaggett/domain_stats
- Can query the Alexa or Cisco Umbrella top million
- Can also query RDAP data to discover domain creation time (to discover newly-registered "baby domains")
  - And much more
- RDAP (Registration Data Access Protocol) is the (eventual) replacement for WHOIS
  - WHOIS: blobs of inconsistent and poorly-formatted data
  - RDAP: can output in JSON
RDAP can output in JSON, which is very handy for integration with other tools

- `rdap -j --type=domain sans.org`

```json
"events": [
  {
    "eventAction": "registration",
    "eventDate": "2003-01-22T21:23:12.000Z"
  },
  {
    "eventAction": "last changed",
    "eventDate": "2020-02-18T22:45:35.960Z"
  }
],
```
domain_stats in action

```
root@DoH:~# curl http://127.0.0.1:8000/alexa/sans.org
64900
root@DoH:~# curl http://127.0.0.1:8000/domain/country/sans.org
US;
root@DoH:~# curl http://127.0.0.1:8000/domain/creation_date/sans.org
1995-08-04 04:00:00;
root@DoH:~#
```
Thank you! – econrad@backshore.net

• Thank you for attending my talk!
• A copy of these slides are available on https://ericconrad.com