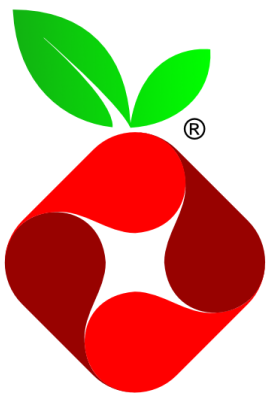
# Regularly refreshing Pi-Hole Regex Block List from external sources

Ben Tasker — 2022-06-11 10:42



Pi-Hole provides simple tooling for managing lists of ad domains to block, but sometimes simple blocklists don't provide enough coverage on their own.

## Blocking Xiaomi's Tracking

The mobile phone manufacturer Xiaomi is a good example of why a more flexible blocking approach is sometimes called for.

Various views within the MIUI system UI contain tracking/ads with a broad range of regionalised addresses used to support these data harvesting activites.

For example, Xiaomi phones sometimes contact the domain `tracking.intl.miui.com`, but there are also regionalised variations such as `tracking.rus.miui.com` and `tracking.india.miui.com`.

Once known, these domains are easy to block, but a purely reactive approach means that there will always be periods where data is collected unimpeded.

It's far preferable, then, to be able to predict what their other tracking domains might be. Unfortunately the regionalisation of Xiaomi's services isn't particularly consistent:

- There are services at `fr.app.chat.global.xiaomi.net`
- But there are none at `tracking.fr.miui.com`
- There are also no services at `tracking.gb.miui.com` **but** DNS lookups for it behave differently to those for `tracking.fr.miui.com`

This inconsistency makes effective blocking of Xiaomi's tracking domains via blocklists quite difficult: not only do we need to be able to enumerate all current domains, we're also reliant on Xiaomi not launching stalkerware services in a new region.

---

## Enter Regex

Regular expressions (regex) provide a tool by which we can improve the effectiveness of our blocks.

Rather than needing to enumerate every variation of `tracking.$foo.miui.com` we can instead provide a pattern to match against

```
^tracking\..+\.miui.com$
```

For those not familiar with Regex, this breaks down as follows

- `^tracking.` : queried name must begin with `tracking` (the `^` signifies start of the input)
- `.+` : allow an unlimited number of any characters
- `\.miui.com$` : the queried name must end with `.miui.com` (the `$` signifies end of the input)

As if this wasn't powerful enough, PiHole also supports approximate matching allowing things like stemming to be used.

For example, this allows us to trivially create a regular expression that'll accept TLD substitutions:

```
^tracking\..+\.miui.(com){#3}$
```

This expression will match any of the following

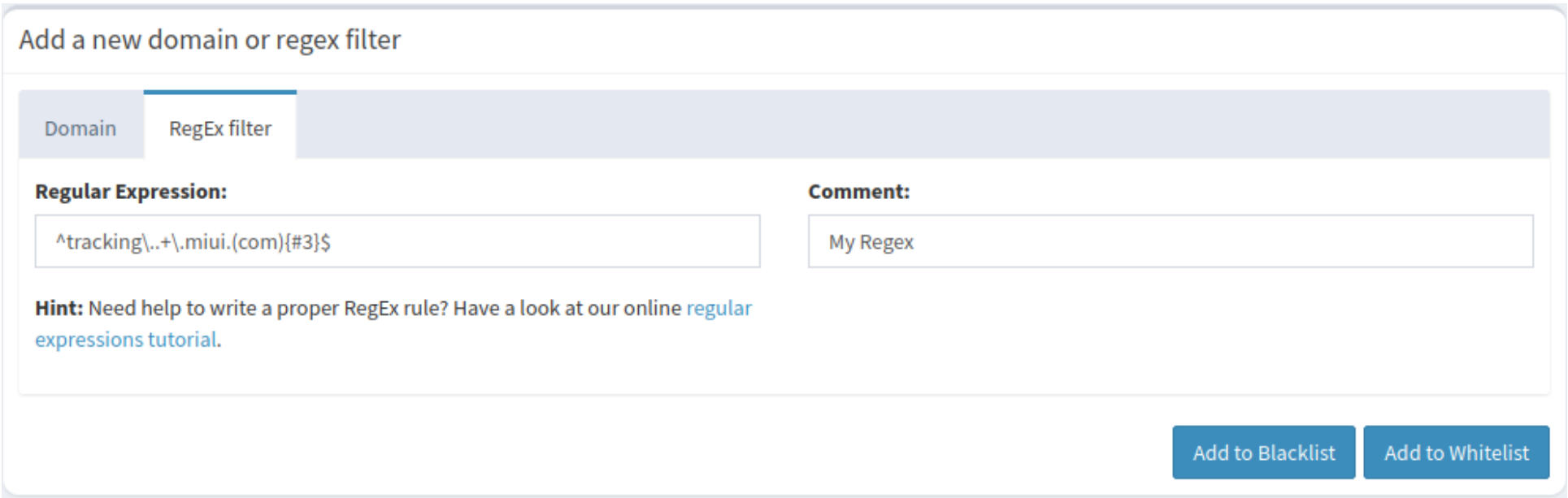
- `tracking.foo.miui.com`
- `tracking.foo.miui.org`
- `tracking.foo.miui.net`

---

# Managing Regex in Pi-Hole

So, why do we need an entire post for this?

Adding regex blocks to Pi-Hole *individually* is trivial as they can be added through the web interface




However, adding a bulk list or linking a remotely maintained list provides a bit more of a challenge.

Older versions of Pi-Hole referenced a file called `regex.list` on disk, allowing easy automation of updates.

But support for that file was dropped when Pi-Hole Version 5 was released last year and regexes now need to be written into the gravity database.

This post details the process of automatically fetching and refreshing lists of regular expressions for Pi-Hole versions later than version 5.

---

## Overview

In version 5, Pi-Hole moved to using the gravity database (`gravity.db`) for list storage. This provided some *serious* manageability benefits, but does mean that ingesting lists of regular inspections became a little more complicated.

The process we need to follow, though, is still fairly simple:

- Fetch list(s) of regular expressions
- Write them into the appropriate table in `gravity.db`
- Tell Pi-hole to reload the list

The final step is necessary because Pi-Hole is incredibly efficient about how it handles regex matching:

Pihole only checks each domain once for regex matches, so if you query `foo.example.com` it'll be checked against each of the regular expressions and the result of that check will be cached. the next query for `foo.example.com` will use that cached evaluation rather than evaluating the expressions again.

That cache persists until `pihole-FTL` is restarted (or is otherwise told to reload lists).

To implement this, we're going to create a short python script to fetch the lists, and (if they've changed) write them in before signalling pi-hole.

---

## The Refresh Script

The script we need is fairly straight forward:

```
#!/usr/bin/env python3
#
# Download zone and regex lists, format them and write into Pi-Hole's gravity DB
#
# Pihole <v5 used regex.list, this script works with v5 onwards
#
# Older versions can use https://github.com/bentasker/adblocklists/blob/master/bin/pihole_apply_regexes.sh
#

'''
License: BSD 3-Clause

Copyright 2022 B Tasker

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materi

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific pri

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, IN
'''

import hashlib
import os
import requests
import subprocess
import sqlite3
import sys

# Define the command to be used to tell Pihole to refresh lists
#
# pihole restartdns reload-lists is preferred over a service restart as it involves no service interruption
# and doesn't blow away the DNS results cache
restart_cmd = ["pihole", "restartdns", "reload-lists"]

# Comment to write into the DB alongside the regexes
comment = 'bentasker/adblock_lists_v2'

def fetchList(url):
    ''' Fetch a text file and return a list of lines
    '''
    r = requests.get(url)
    return r.text.split("\n")
```

We're actually going to go a *little* further than originally specced: we'll also fetch a list of [blocked DNS zones](). This is a list of domains where we shouldn't just block the listed domain, but all of it's subdomains too (something else that static blocklists can't really achieve).

To implement this, we create a function that fetches the zone list and converts each entry into a pair of regular expressions

```
def fetchZoneList(url):
    ''' Fetch the list of zones and translate into regexes
    '''
    t = fetchList(url)

    # Currently just a domain list, we need to turn it them into regexes
    regexes = []
    for line in t:
        # Ignore empty lines
        if line == "":
            continue

        # escape periods
        regex = line.replace(".","\.")

        # Add the apex domain
        regexes.append(f"^{regex}$")

        # Add a regex to match subdomains
        regexes.append(f"^.+\.{regex}$")

    return regexes
```

So, for example, if the fetched list contains `aaxads.com` we'll generate `^aaxads\.com$` and `^.+\.aaxads\.com$`. The first regex blocks the apex domain (`aaxads.com`), whilst the second blocks all subdomains (`foo.aaxads.com`, `bar.sed.aaxads.com` etc).

Now that we can fetch and generate regular expressions, we need a function to write them into Pi-Hole's gravity database.

`gravity.db` is a `sqlite` database and we need to write into table domainlist, specifying the value `3` (regex blacklist) for the `type` column.

We also want to delete anything we inserted previously to ensure that removing things from the list leads to them being removed from Pi-Hole (otherwise a mistake in a regex would persist forever).

```
def writeRegexes(regex_list,comment):
    ''' Write regexes into the gravity database
  '''
    conn = sqlite3.connect("/etc/pihole/gravity.db")
    c = conn.cursor()

    # We're refreshing the list, so delete any existing entries that use our list comment
    #
    # This ensures that if a regex is removed from the list, it also gets removed from Pi-Hole
    c.execute('DELETE FROM domainlist WHERE comment=?',(comment,))

    # Insert the regexes
    c.executemany('INSERT OR IGNORE INTO domainlist (type, domain, comment, enabled) '
                  'VALUES (3, ?, ?, 1)',
                  [(x, comment) for x in sorted(regex_list)])

    conn.commit()
```

Now that we've created the functions we need, we need to actually call them in order to fetch and process the lists.

We'll use my adblock lists as our sources.

```
# Fetch the two lists
regexes = fetchList('https://raw.githubusercontent.com/bentasker/adblock_lists_v2/master/lists/regexes.txt')
zones = fetchZoneList("https://raw.githubusercontent.com/bentasker/adblock_lists_v2/master/lists/zones.txt")

# Merge and remove any empty values
merged = list(filter(None, regexes + zones))

# De-dupe and sort
merged = list(set(merged))
merged.sort()
```

We now have a de-duplicated and sorted list of regular expressions, but we don't want to update `gravity.db` unless something changed - otherwise we'd be signalling pi-hole every time our task fires which isn't particularly efficent.

As a simple check for changes, we'll

- Calculate a checksum of the generated list
- Compare that to a checksum of the list generated when the last change was made
- Update the DB (and the cached list) if the checksums differ

We achieve this by converting our list to a string (so that it can easily be written to disk) and use a fast checksum.

```
# Convert to a string so that we can hash it to check for changes
mergedstr = "\n".join(merged)

# Calculate a SHA1
sha1 = hashlib.sha1()
sha1.update(mergedstr.encode('utf-8'))
merged_sha = sha1.hexdigest()

# Initialise variable for later
cache_sha1 = ""

# Read the cachefile if it exists
if os.path.exists("/etc/pihole/regex_list_cache"):
    with open("/etc/pihole/regex_list_cache") as f:
        cache = f.read()
        # Hash the contents
        sha1 = hashlib.sha1()
        sha1.update(cache.encode('utf-8'))
        cache_sha1 = sha1.hexdigest()

# Has the list changed?
if cache_sha1 != merged_sha:
    # Yes!

    # Write the regexes into pihole
    writeRegexes(merged, comment)

    # Write the new list to the ondisk cache
    fh = open("/etc/pihole/regex_list_cache", "w")
    fh.write(mergedstr)
    fh.close()

    # Signal Pi-Hole
    subprocess.run(restart_cmd, stdout=subprocess.DEVNULL)

    # Tell the calling script that a change was made
    sys.exit(0)
else:
    # Signal that no change occurred
    sys.exit(2)
```

A copy of the final script is available on Github.

---

## Deploying

Deploying the script is pretty simple.

Fetch it onto the system running Pi-Hole and make it executable

```
wget https://raw.githubusercontent.com/bentasker/adblock_lists_v2/master/examples/pihole_update_regex_lists.py
chmod +x pihole_update_regex_lists.py
sudo mv pihole_update_regex_lists.py /usr/local/bin/
```

Create a cronjob to run the script

```
echo '0 */4 * * *  root  /usr/local/bin/pihole_update_regex_lists.py' | sudo tee /etc/cron.d/update_pihole_regexes
systemctl restart cron
```

This will schedule the script to run every 4 hours (on the hour). We run as `root` because the script needs to have sufficient privileges to both write into `gravity.db` and to tell Pihole to reload lists.

---

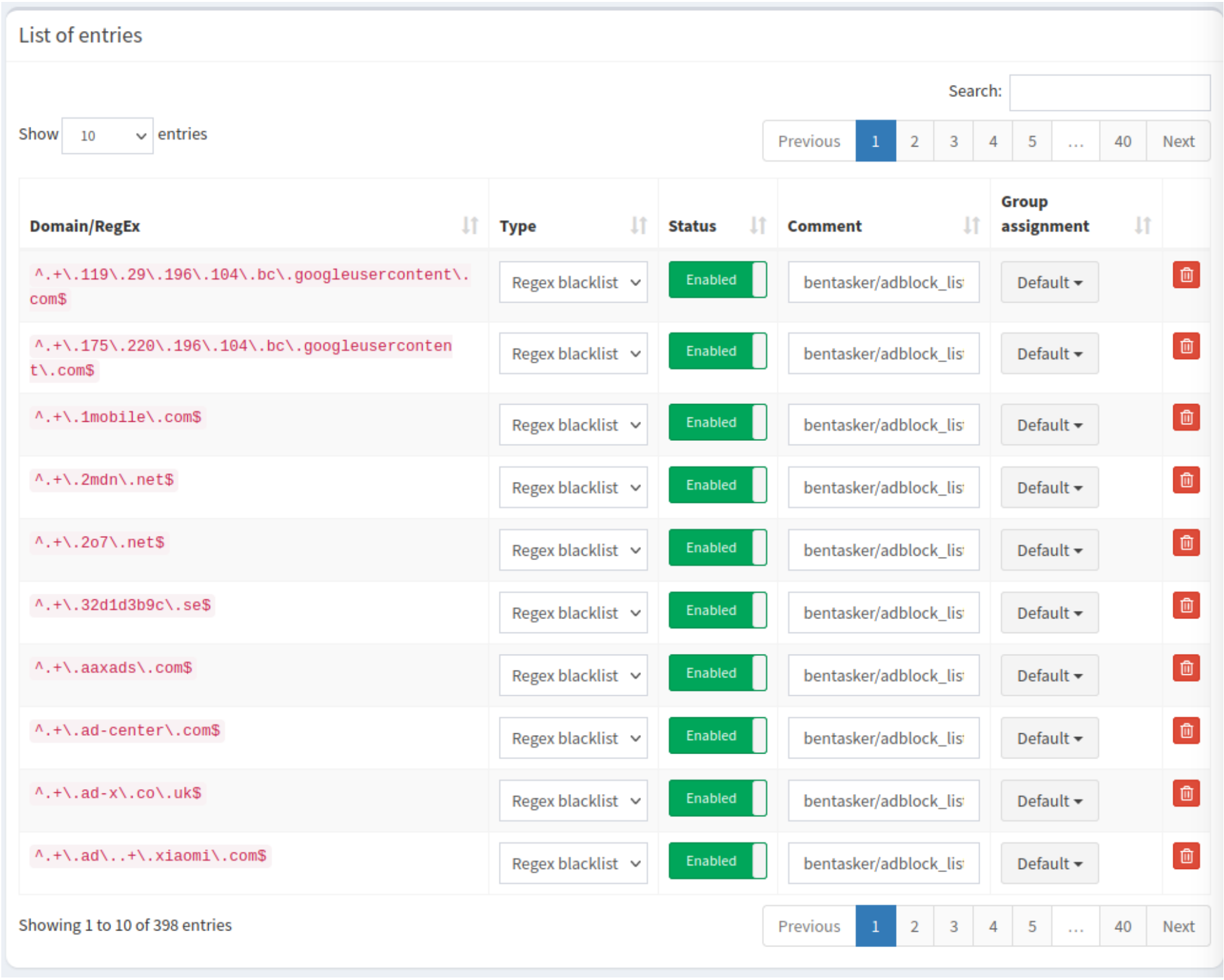
## Verifying

To verify that the regexes have been fed in and will be used, we use Pihole's web interface

- `http://[yourpihole]/admin/`
- `Login`

- Enter password
- `Group Management`
- `Domains`

This should display a list of entries




We can use one of these records to create domain names to test with.

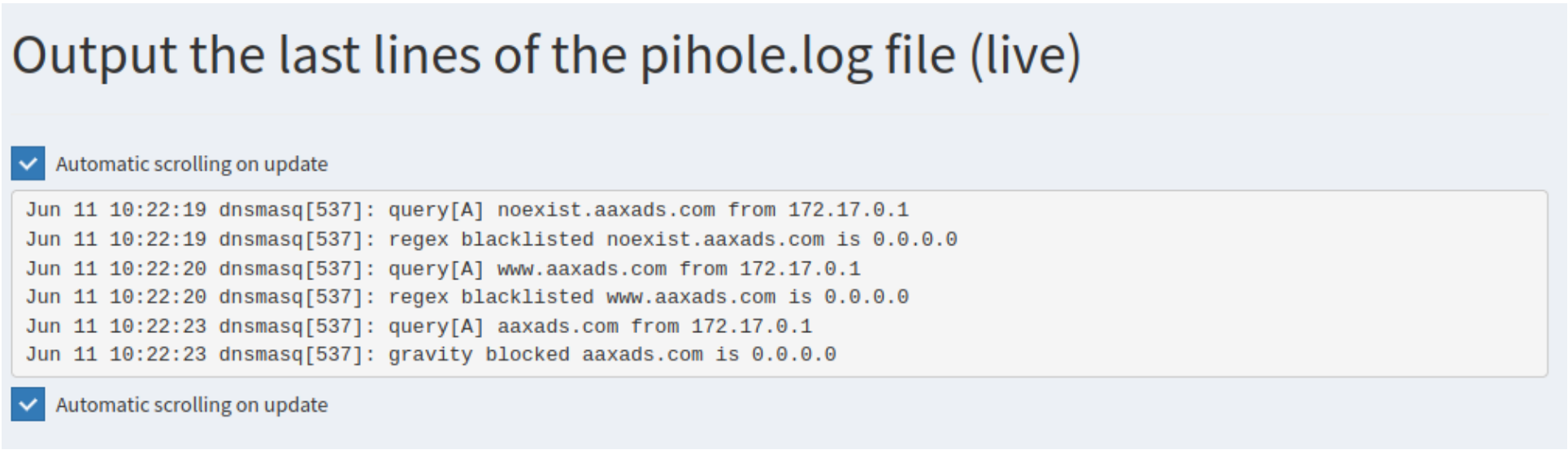
For example, in the screenshot above we can see that `aaxads.com` is present, so we can test by creating nonesensical subdomains for it as well as testing the apex domain itself.

We can also use Pi-Hole's UI to tail its log to verify that blocks are definitely because of our regexes: `Tools` -> `Tail pihole.log`

We use `dig` (or `nslookup` if you prefer) to send Pi-Hole DNS queries for domains that we expect to be blocked (in these examples, the IP address of my Pi-Hole server is `172.17.0.5`)

```
$ dig +short @172.17.0.5 noexist.aaxads.com
0.0.0.0
$ dig +short @172.17.0.5 www.aaxads.com
0.0.0.0
$ dig +short @172.17.0.5 aaxads.com
0.0.0.0
```

We can see that they've been blocked (the response is `0.0.0.0`), and the Pihole log shows us why




Our two subdomains were blocked because of matches to our regexes. However, the apex domain (`aaxads.com`) was blocked by a gravity list - it exists in one of the static blocklists, so a regex wasn't needed in this case (blocklists are checked before regexes).

---

## Testing Regexes

You might decide that you want to add your own regexes, but because they're powerful they're always worth testing first.

Pihole provides a command line argument that you can use to test regex matches, so that you don't need to set an untested regex live:

```
$ pihole-FTL regex-test 'tracking.foo.miui.net' '^tracking\..+\.miui.(com){#3}$'
[i] Compiling regex filter...
    Compiled regex filter in 0.063 msec

Checking domain...
    ^tracking\..+\.miui.(com){#3}$ matches
   Time: 0.033 msec
```

Remember to test similar names that you *don't* want your regex to match, and don't use regex's unnecessarily:

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems. Zawinski, 2007

---

## Conclusion

Regex filters provide an incredibly powerful tool to help block ads and tracking at the DNS level.

Although the method of feeding lists into Pi-Hole has changed in more recent versions, it's still reasonably straight forward to do so and is easily achieved with a cronjob.

As a result, you can implement some incredibly broad blocks that simply aren't possible with a static list of exact matches.

adblock ads advertising documentation pihole privacy regex site news

Previous post

Next post

## Related Snippets


- Listing current switch interface status with Flux (Flux)
- List processes currently using swap space (BASH)
- Scale a list of numbers down proportionally (Python3)
- List USB Flash Drive History (PowerShell)
- Sort list as if it were numbers (Python)


---

## Recent Posts


- Adding Social Media Sharing Buttons to a Nikola Site
- Stop Requiring Phone Numbers
- Enabling and monitoring the zswap compressed page cache on Linux

- The Effectiveness Of SSH Tarpits
- Designing Software to Minimise Harm Whilst Complying With Legal Obligations

Search…