Public Suffix List DNS Query Service

OARC 36
November 30, 2021

https://publicsuffix.zone/

Peter Thomassen (deSEC, Secure Systems Engineering)

The Public Suffix List (PSL)

> *A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are `.com`, `.co.uk` and `pvt.k12.ma.us`. The Public Suffix List is a list of all known public suffixes.*
> — https://publicsuffix.org/

What does that mean?

- Informs about organization and policy boundaries in the domain space
- Supports wildcards, and exceptions from wildcards
- Maintained by the community (on GitHub) and provided as a text file

PSL Use Cases

- Browsers
  - cookie/script scoping, domain highlighting / phishing prevention, …
- Certificate issuance
  - think of `*.co.uk`
- Multi-tenant DNS operation ← our motivation
  - think of a customer creating `co.uk`, blocking others from creating `example.co.uk`
- DMARC
  - identify the “organization domain” (= public suffix plus previous label, e.g. example.co.uk)

Why a PSL Query Service?

Situation without Query Service:

- Applications have to bring a copy of the list, and need to keep it up to date
- Applications have to parse the list
- Extracting information from the PSL requires a multi-staged algorithm

With a DNS-based Query Service:

- No need for applications to parse or refresh the PSL altogether
- Public suffix can be retrieved ad-hoc with a simple lookup, cacheable
- No need for specialized tooling

How it works

- In a special zone, **public suffixes are stored as PTR owner names and values**
  - `co.uk PTR co.uk.`
- All **other names have a CNAME record** (or are covered by a CNAME wildcard)
- A domain's **public suffix is retrieved as the PTR record at the domain's name**
  - CNAMEs take care of "routing"
- **Auxiliary rules** that influenced the PTR outcome are **given as a TXT record**
  - e.g. in case of wildcard exceptions: parent rule is given in PTR, wildcard + exception in TXT
- We implemented this under `query.publicsuffix.zone`
  - **Authenticity** is provided by DNSSEC

Implementation Challenges

- The PSL **parsing algorithm is not trivial**
  - for example, it’s important to get rule precedence right
- PSL rules ***almost* match DNS data structures**, but not quite (see limitations)
- PSL rules on a deeper level cause empty non-terminals
  - intermediate levels need CNAME but can’t be covered with a DNS wildcard

  → Things need to be glued together with a CNAME chain
- **~75k records total** (~20k for PSL mapping, ~55k for DNSSEC)
  - incremental updates require **calculating large diff**

Implem

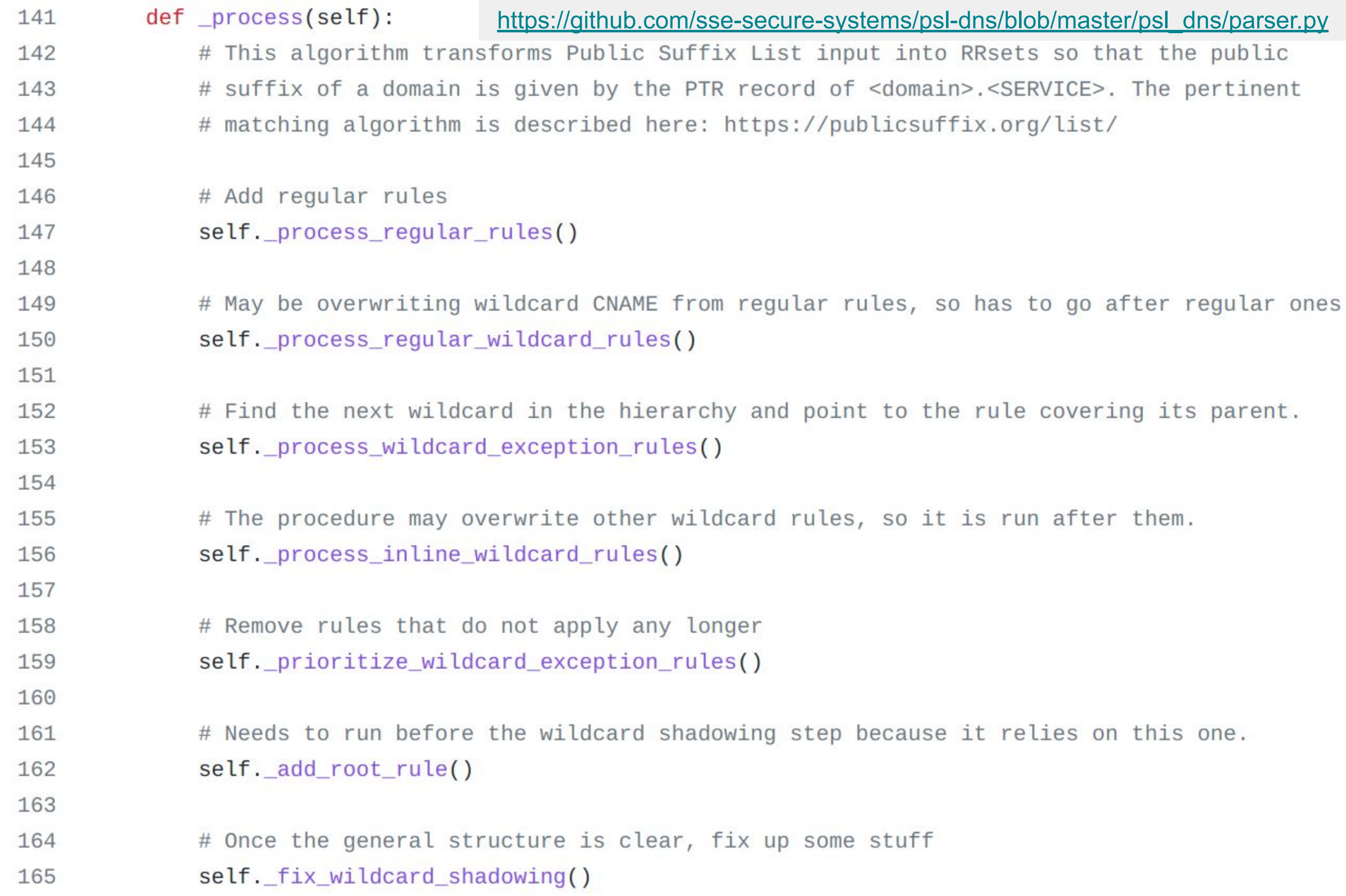
———

- The P
  - f
- PSL ru
- PSL ru
  - in

→ Th

- ~75k
  - in

https://github.com/sse-secure-systems/psl-dns/blob/master/psl_dns/parser.py



```python
    def _process(self):
        # This algorithm transforms Public Suffix List input into RRsets so that the public
        # suffix of a domain is given by the PTR record of <domain>.<SERVICE>. The pertinent
        # matching algorithm is described here: https://publicsuffix.org/list/

        # Add regular rules
        self._process_regular_rules()

        # May be overwriting wildcard CNAME from regular rules, so has to go after regular ones
        self._process_regular_wildcard_rules()

        # Find the next wildcard in the hierarchy and point to the rule covering its parent.
        self._process_wildcard_exception_rules()

        # The procedure may overwrite other wildcard rules, so it is run after them.
        self._process_inline_wildcard_rules()

        # Remove rules that do not apply any longer
        self._prioritize_wildcard_exception_rules()

        # Needs to run before the wildcard shadowing step because it relies on this one.
        self._add_root_rule()

        # Once the general structure is clear, fix up some stuff
        self._fix_wildcard_shadowing()
```

Examples

Standard cases:

```
$ dig +noall +answer PTR indico.dns-oarc.net.query.publicsuffix.zone
indico.dns-oarc.net.query.publicsuffix.zone.   21530 IN CNAME  net.query.publicsuffix.zone.
                net.query.publicsuffix.zone.    7199 IN PTR    net.

$ dig +noall +answer PTR s3.dualstack.eu-west-1.amazonaws.com.query.publicsuffix.zone
s3.dualstack.eu-west-1.amazonaws.com.query.pu… 21600 IN PTR    s3.dualstack.eu-west-1.amazonaws.com.

$ dig +noall +answer PTR s4.dualstack.eu-west-1.amazonaws.com.query.publicsuffix.zone
s4.dualstack.eu-west-1.amazonaws.com.query.pu…  7198 IN CNAME  dualstack.eu-west-1.amazonaws.com.query.pu…
   dualstack.eu-west-1.amazonaws.com.query.pu…  7198 IN CNAME            eu-west-1.amazonaws.com.query.pu…
             eu-west-1.amazonaws.com.query.pu…  7198 IN CNAME                      amazonaws.com.query.pu…
                       amazonaws.com.query.pu…  7198 IN CNAME                                com.query.pu…
                                 com.query.pu…  7198 IN PTR    com.
```

Wildcard with exception:

```
$ dig +noall +answer ANY www.ck.query.publicsuffix.zone | grep -v RRSIG
www.ck.query.publicsuffix.zone.                21600 IN PTR      *.
www.ck.query.publicsuffix.zone.                21600 IN TXT      "!www.ck"
www.ck.query.publicsuffix.zone.                21600 IN TXT      "*.ck"
```

Implementations / Demo

- Lookup zone implemented under `query.publicsuffix.zone`
- https://publicsuffix.zone/ has a live demo
  - uses JavaScript requests to Google's DoH resolver
- Python implementation: https://pypi.org/project/psl-dns/
  - library + CLI
  - implements both querying and parsing (for preparing zone updates)
  - currently supports deSEC implementation, but interface is provider-agnostic

Limitations

Inline wildcards (`foo.*.example.com`)

- **not possible in DNS**, but the PSL supports them
- **no such entries** at the moment
  - support **may be dropped** soon: https://github.com/publicsuffix/list/issues/145

→ DNS implementation provides **full coverage in practice**

Updates

- currently **every few weeks** (not automated)
- could be **automated easily** based on GitHub action or atom feed

Next Steps?

- The PSL Query Service works perfectly well for internal use case at deSEC
- Are there any use cases beyond that?
  - Do they need automated zone updates?
  - Other features? (e.g. distinguish between ICANN and PRIVATE section)
- It has been suggested to make this a “permanent service” embedded in the community
  - Does that make sense?
  - If yes, what kind of oversight is needed / who does that?
- …

Thank you!

… also to our sponsors:

SSE

PCH
Packet Clearing House

Questions?

Backup

Addressing Privacy Concerns

- DNS **resolvers learn about domains** that get queried
- Depending on the use case, this may not be up to required privacy standards

Solution ideas

- **Resolver-local copy** (e.g. via AXFR)
  - deSEC use case: we resolve directly against our own auth → no leakage
- *k*-anonymity: replace all labels by truncated hashes → collisions intended
  - **queries are fuzzy**
  - **returns list of hashes** that matched the truncated query (client infers the answer from the list)
  - inference from hierarchy patterns still possible
  - required API changes not very DNS-like → perhaps **not the best idea**