Onion Plan

Usability Roadmap - Service Discovery - Technical details

Silvio Rhatto

2022.Q4

Onion Support Group - The Tor Project

Summary

Summary

1. Context.
2. What really matters?
3. What can easily be replaced or thrown away?
4. Why pluggable?
5. The Tor NS API.
6. Phase 1 Proof of Concept.

1. Context

1. Context

- This presentation covers the technical details for Onion Plan's usability proposal for service discovery (slides) focusing in the long term.
- It needs further discussion, ideas an especially some stress analysis to ensure it's doable.
- Also needs syncing with the upcoming arti 1.2.0 and C Tor maintenance roadmaps.

2. What really matters in the usability proposal?

2. What really matters in the usability proposal?

- Basic properties: incremental, modular etc.
- Opportunist discovery of .onion addresses.
- Pluggable methods.

3. What can easily be replaced or thrown away?

3. What can easily be replaced or thrown away?

1. Proposal 279 is not a MUST, but having a Tor Name System API would make naming systems and service discovery development, testing and adoption way easier.
2. The proposed phases can be changed and re-arranged.
3. Implementation on C Tor can be excluded from the roadmap or have a lower priority. This may make sense given the current development availability. In the other hand, it can slow down the timeframe to put things into production.

4. Why pluggable?

4. Why pluggable?

1. Allow third parties to provide their own discovery methods regardless of what Tor ship by default as supported technology.
2. The community could contribute more and the roadmap could be split between teams with only minimum point of contact.
3. Technical and governance criteria: aiding decision-making in what should be officially supported (and enabled by default) and what should leave as unofficial, third-party plugins or disabled by default.
4. After Phase 1 is implemented, the community may also contribute unofficially with their own resolvers.

5. The Tor NS API

Proposal 279 (2016)

> *[...] a modular Name System API (NSA) that allows developers to integrate their own name systems in Tor. [...] It should be flexible enough to accommodate all sorts of name systems*
> *[...] Tor asks the name system to perform name queries, and receives the query results. [...] It aims to be portable and easy to implement.*

See https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/279-naming-layer-api.txt

What it brings

```
# New torrc(5) config
OnionNamePlugin 0 .hosts.onion    /usr/local/bin/local-hosts-file
OnionNamePlugin 1 .zkey.onion     /usr/local/bin/gns-tor-wrapper
OnionNamePlugin 2 .bit.onion      /usr/local/bin/namecoin-tor-wrapper
OnionNamePlugin 3 .scallion.onion /usr/local/bin/community-hosts-file
```

Implementations

- TorNS (2017-2019):
  - Tor NS API based on Proposal 279
  - Proof of concept using txtorcon.
  - https://github.com/meejah/torns
- StemNS:
  - TorNS fork using Stem.
  - https://github.com/namecoin/StemNS
- C Tor, arti and Tor Browser:
  - No built-in implementation exists.

How they work

For the prop279 implementations based on the control spec – TorNS and StemNS, discovery happens in the following way at the client side:

1. They require `__LeaveStreamsUnattached` to be set.
2. Once a new stream appears, they proceed doing a name resolution.
3. If a name is found, they proceed with a `REDIRECTSTREAM` before attaching the stream to a circuit.

Transparent connections

This is completely transparent to the HTTP client relying on SOCKS5h.

Maybe a custom `REP` field from the SOCKS response could indicate to clients that an opportunistic Onion Service discovery happened.

Or maybe the HTTP client like Tor Browser could detect whether BND.ADDR from the SOCKS reply is a Onion Service and offer some UI indicator.

Check socks-extensions.txt and RFC 1928 for details.

What if. . . ?

An hypothetical example:

```
# New torrc(5) config
OnionNamePlugin  0 .some.onion /usr/bin/some-onion-resolver    # Phase 3
OnionNamePlugin 98 *           /usr/bin/dns-to-onion-resolver  # Phase 1
OnionNamePlugin 99 *           /usr/bin/sauteed-onion-resolver # Phase 2
```

Which means

1. In Phase 1, the DNS-based address translation is implemented, with a catch-all rule for all domains.
2. In Phase 2, the Sauteed Onions or other address translation method is implemented with a fallback catch-all rule for all domains if the DNS resolver fails for some reason.
3. In Phase 3, "pure" Onion Name plugins can be officially included.
4. Matching will happen from the specific (like `.some.onion`) to the general (`*`).
5. For non-.onion TLDs, priority will be from the DNS to the Sauteed Onion (or other fancier methods).

Does Proposal 279 should be amended or replaced?

- There's an extensive evaluation of what needs to be defined/done to make Proposal 279 work with the current Onion Plan.
- Given the stability of arti's API and the support for Rust's Foreign Function Interface (FFI), it's worth thinking in different ways to plug a Tor NS API not considered by the time Proposal 279 was written.

Some possibilities

1. Have Tor NS API as a library and a configuration format similar to Proposal 279. Build on arti only the point of contact with this library (the same if a C Tor implementation is to be considered).
2. Build the Tor NS API it directly on arti (which can be safer), supporting pluggable, external resolvers.
3. Support only resolver plugins that can be included using FFI (during compilation). Trade off: not very pluggable but may be easier to maintain. Can make iteration faster an does not block having an additional pluggable NS API in the future.

6. Phase 1 Proof of Concept (PoC)

6. Phase 1 Proof of Concept (PoC)

Putting aside any Tor NS specifics, let's think about a resolver method for a moment.

DNS, TLS SNI and .onion: proof of concept

Setup:

- An existing site: https://autodefesa.org.
- It's existing Onion Service: `autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclpiad.onion`.

Today's behavior

- Attempt to access https://autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclpiad.onion.
- Address is hard to remember.
- HTTPS connection will fail since the certificate is not valid for the .onion address.

Querying for an onion TXT record

Just an example, without DNSSEC, output formatted for readability:

```
$ dig autodefesa.org TXT

[...]

;; ANSWER SECTION:
autodefesa.org.     3600    IN  TXT
  "onion=autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclpiad.onion"

;; Query time: 60 msec
[...]
```

Testing SNI

If we use OpenSSL via Tor, we can get the cert via Onion Service using TLS SNI:

```
torsocks openssl s_client -servername autodefesa.org \
  -tlsextdebug -connect \
  autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclpiad.onion:443
```

Using curl

Almost working PoC: fetching the site via Onion Services using TLS SNI using curl:

```
torsocks curl -vik --resolve \
  autodefesa.org:443:autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclp
  https://autodefesa.org
```

But it won't work with `curl(1)`, since `--resolve` requires an IP address.

Using OpenSSL

Working PoC with OpenSSL:

```
echo -e \
  "GET / HTTP/1.1\r\nHost:autodefesa.org\r\n\r\nConnection: Close\r\n\r\n"
  torsocks openssl s_client -quiet -servername autodefesa.org -connect \
  autodefcecpx2mut5medmyjxjg2wb6lwkbt3enl74frthemyoyclpiad.onion:443
```

What it does:

1. Opens a TLS connection to the Onion Service using Tor.
2. During handshake, asks for the `autodefesa.org` certificate.
3. Do a regular HTTP `GET` after the connection is established.

Result: page is fetched via Onion Service and HTTPS with a validated certificate!

What else for DNS support?

Check the Appendix: Specs for DNS-based .onion records.

Questions?

Questions?

rhatto@torproject.org