Talla

An Erlang implementation of Tor

Alexander Færøy
• Introduction.
• C.
• The architecture of Talla.
• Testing.
• Representation of the Tor network in Erlang.
• Questions.
Introduction
What is Talla?

• An attempt to build a robust implementation of a Tor relay daemon in Erlang.

• An opportunity for me to understand the inner workings of the Tor network better.

• A typical “evenings-only open source project” :-)

History

• Met Linus Nordberg from The Tor Project at the Erlang User Conference in 2015.

• Started development in August 2015.

• Very simple (read: messy) proof of concept up and running in December 2015.

• More sensible design work started in January 2016.

• Lasse Andersen joined in 2016.
• The official Tor in C.

• PurpleOnion in C#.

• GoTor in Google’s Go language.

• Galois Inc’s Haskell implementation.

• Orchid, tor-research-framework, and OnionCoffee in Java.

• node-Tor in JavaScript.

• Oppy, pycepa, and TorPylle in Python.

• Complete list on https://trac.torproject.org/projects/tor/wiki/doc/ListOfTorImplementations
Minimum Viable Product

• Due to Tor’s end-to-end construction we really want to avoid implementing a Tor **client**.

• Primary priority: middle relay, secondary priority: exit nodes.

• Onion services.

• Use “as few” C dependencies as possible.

• Rewriting from scratch should always be an option.
Carefulness

• Running experimental Tor implementations on the “production network” would be irresponsible.

• Test networks.

• See email thread on tor-dev: https://lists.torproject.org/pipermail/tor-dev/2016-August/011300.html
Erlang

- Functional programming language designed by Ericsson in Sweden.
- Focus on concurrency via message passing.
- Extremely powerful when it comes to working with network protocols.
- Running on the BEAM virtual machine.
• Model your code using modules and functions.
• Model your complex stateful objects as processes.
• Use processes to handle ordering of events.
• Rich mocking possibilities in the language.
• Rich testing frameworks.
The Erlang mentality

- Program as if you are writing a domain specific operating system.
- Your “program” consists of multiple applications, which consists of multiple modules, which is running in the Erlang BEAM VM.
- Hot loading of modules and applications without restart.
Community

People who are interested in following the development of Talla should feel free to join #talla on irc.baconsvin.org or 6nbtgccn5nbcodn3.onion with TLS on port 6697.
• It is very hard to write complicated and safe C code.

• The “Tor Daemon” that is in used today is written in C.

• The “Tor Daemon” is generally considered by security experts to be a very high quality C project.
#Tor #Tor2017 #InfoSec #torbrowser #TorProject #privacy #bugs PVS-Studio Team: No Bugs! viva64.com/en/b/0507/
“Special congratulations go to the Samba, tor, OpenPAM, and Ruby teams for being the first to reach Coverity Rung 3 certification.”

–2009 Coverity Scan Open Source Report
Tor and C

• Very high test coverage. “New features implies new tests” policy.

• Team rotation duties for things like Coverity issues.

• Fuzzing with and without Google’s oss-fuzz.

• Code review.

• Rust in Tor (work by Sebastian Hahn, Chelsea Komlo, and Isis Lovecrufit).
Talla and C

• The BEAM VM have quite a lot of C code.

• We use libsodium for some cryptography.

• We use libcrypto from OpenSSL or LibreSSL, but do not use libssl.

• Some small C functions that was not exposed in Erlang.
The architecture of Talla
• Written by Jesper Louis Andersen.

• Used for its /dev/urandom interface.

• Used for x25519 Diffie-Hellman.

• Source code: https://github.com/jlouis/enacl
Ed25519

- Used for ed25519 signatures to the directory services.
- Multiple implementations of Ed25519 :-(
- We should probably switch to the Donna implementation instead.
- Major thanks to Yawning Angel from Tor.
- Source code: https://lab.baconsvin.org/talla/ed25519_ref10
• Experimental Erlang NIF of the New Hope Post-Quantum cryptographic system.

• Supports “normal” New Hope and Tor New Hope (Tor Proposal #270 by Isis Lovecruft and Peter Schwabe).

• Source code: https://lab.baconsvin.org/ahf/luke
Onion

- Shared utilities needed for working with the Tor network.
- Small C function for generating an RSA key pair. Not needed for OTP 20.
- Well-tested code.
- Automated test execution.
- The most stable part of Talla right now :-)
- Source code: https://lab.baconsvin.org/talla/onion
Talla

- Core application is for centralised services to the system.
- One application for Onion Routing.
- One application is for Directory service (announcement only as of 2016).
Applications and Libraries

Talla
- Core
- Onion Routing
- Directory

Onion
- RSA (!)

enacl

Ed25519 (ref 10)

Luke (New Hope)
Testing
Classical Unit Testing

hkdf_tor_test() ->
%% Taken from test_crypto_hkdf_sha256() in tor/src/test/test_crypto.c.
Salt = <<"ntor-curve25519-sha256-1:key_extract">>,
Expand = <<"ntor-curve25519-sha256-1:key_expand">>,
[
  ?assertEqual(hkdf(""), Salt, Expand, 100),
  base16_decode(["d3490ed48b12a48f9547861583573fe3f19aafe3f81dc7fc75",
                 "eed96d741b3290f941576c1f9f0b2d463d1ec7ab2c6bf71cd",
                 "d7f826c6298c00dbfe6711635d7005f0269493edf6046cc7e7",
                 "dcf6abe0d20c77cf363e8ffe358927817a3d3e73712cee28d8"])),
  ?assertEqual(hkdf("Tor"), Salt, Expand, 100),
  base16_decode(["5521492a85139a8d9107a2d5c0d9c91610d0f95989975ebee6",
                 "c02a4f8d622a6cfdf9b7c7edd3832e2760ded1eac309b76f8d",
                 "66c4a3c4d6225429b3a016e3c3d4591152fc87bc2de9630c3",
                 "961be9f3db9f39197ea8e5977180801926d3321fa21513e59ac"])),
  ?assertEqual(hkdf("AN ALARMING ITEM TO FIND ON YOUR CREDIT-RATING STATEMENT"),
              Salt, Expand, 100),
  base16_decode(["a2aa9b50da7e481d30463adb8f233ff06e9571a0ca6ab6df0f",
                 "b206fa34e5bc78d063fc291501beec53b36e5a0e434561200c",
                 "5f8bd13e0f88b3459600b4dc21d69363e2895321c06184879d",
                 "94b18f078411be70b767c7fc40679a9440a0c95ea83a23efbf"]))
].
hkdf_rfc5869_1_test() ->
IKM = base16_decode("0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b"),
Salt = base16_decode("00010203040506070809090a0b0c"),
Info = base16_decode("f0f1f2f3f4f5f6f7f8f9"),
L = 42,
[
    ?assertEqual(hkdf(IKM, Salt, Info, L),
        base16_decode(["3cb25f25faacd57a90434f64d0362f2a",
                       "2d2d0a90cf1a5a4c5db02d56ecc4c5bf",
                       "34007208d5b887185865"]))
].
Property based tests

- Generalise your tests into “properties”.
- Generators, shrinkers, and their relationship.
- We use the free and open source QuickCheck implementation named Proper. For more information see: http://proper.softlab.ntua.gr
- Mostly stateless testing right now.
-module(onion_base64).

-export([encode/1, decode/1, valid/1]).

-spec encode(Data) -> Encoded
  when Data :: binary(),
       Encoded :: binary().
  encode(Data) -> ...

-spec decode(Encoded) -> {ok, Decoded} | {error, Reason}
  when Encoded :: binary(),
       Decoded :: binary(),
       Reason :: term().
  decode(Encoded) -> ...

-spec valid(Data) -> boolean()
  when Data :: binary().
  valid(Data) -> ...
prop_base64_iso() ->
  ?FORALL(Data, binary(),
    begin
      Encoded = onion_base64:encode(Data),
      true = onion_base64:valid(Encoded),
      {ok, Decoded} = onion_base64:decode(Encoded),
      Data =:= Decoded
    end).

$ rebar3 proper
 ===> Testing prop_base64:prop_base64_iso()

..............................
..............................
..............................
..........

OK: Passed 100 test(s).
-module(onion_dh).
-export([keypair/0,  
         shared_secret/2, 
         is_degenerate/1, 
         params/0]).

-define(G, 2).
-define(P, ...).

keypair() -> ...

shared_secret(SecretKey, PublicKey) -> ...

is_degenerate(Value) -> ...

params() -> [\?P, \?G].
keypair() ->
   #{ secret := SecretKey,
     public := PublicKey } = onion_dh:keypair(),
   {SecretKey, PublicKey}.

prop_shared_secret() ->
   ?FORALL({{AS, AP}, {BS, BP}}, {keypair(), keypair()},
     begin
     {ok, SharedA} = onion_dh:shared_secret(AS, BP),
     {ok, SharedB} = onion_dh:shared_secret(BS, AP),
     SharedA =:= SharedB
     end).
p() -> ?LET(L, onion_dh: params(), hd(L)).
g() -> ?LET(L, onion_dh: params(), lists:last(L)).
bad_public_key() ->
  ?LET(P, p(),
  ?LET(G, g(),
    begin
      oneof([
        %% All negative numbers.
        neg_integer(),
        %% 0 and 1.
        integer(0, 1),
        %% The generator.
        G,
        %% P - 1 towards infinity.
        integer(P - 1, inf)
      ])
    end)).
prop_degenerate() ->
  ?FORALL(BadPublicKey, bad_public_key(),
    onion_dh: is_degenerate(BadPublicKey)).
prop_public_key_not_degenerate() ->
  ?FORALL({_, P}, keypair(),
    not onion_dh: is_degenerate(P)).

-module(onion_dh).
-export([keypair/0,
  is_degenerate/1,
  params/0])).
(define(G, 2).
(define(P, ...).
keypair() -> ...
is_degenerate(PublicKey) -> ...
params() -> [?P, ?G].

}
Network testing

• We use Chutney for spawning an independent Tor network with C Tor directory authorities, middle relays, exit nodes, client nodes and Talla middle nodes.

• Very easy way to catch “stupid” bugs while developing.

• Fast to setup, easy to use:

  $ chutney configure misc/chutney/talla-mixed
  $ chutney start misc/chutney/talla-mixed
  $ chutney status misc/chutney/talla-mixed
  $ chutney stop misc/chutney/talla-mixed
Internals of Talla
Peer

Peer Send
Resources

- Tor specifications: gitweb.torproject.org/torspec.git - we are focused on tor-spec.txt and dir-spec.txt as of 2016.

- Ferd Hebert’s Learn You Some Erlang for Great Good: learnyousomeerlang.com and erlang-in-anger.com

- The C Tor source code: https://gitweb.torproject.org/tor.git/
Source Code

The source code and issue tracker can be found at the Baconsvin Gitlab instance at

https://lab.baconsvin.org/talla
Conclusions

• Writing a **safe** Tor implementation from scratch using the specifications only is close to “impossible”.

• Great way of getting a much deeper understanding of how Tor works.

• Relaying traffic works; still missing code for handling onion services.

• Running on a small test network, but not on the production network yet.
Questions?

Come by the Baconsvin Village in the Turing field if you want to talk about Tor and Talla.

ahf@torproject.org

FDB1 6F45 A477 B314 E874 32EC 61A2 08E1 6E7C B435