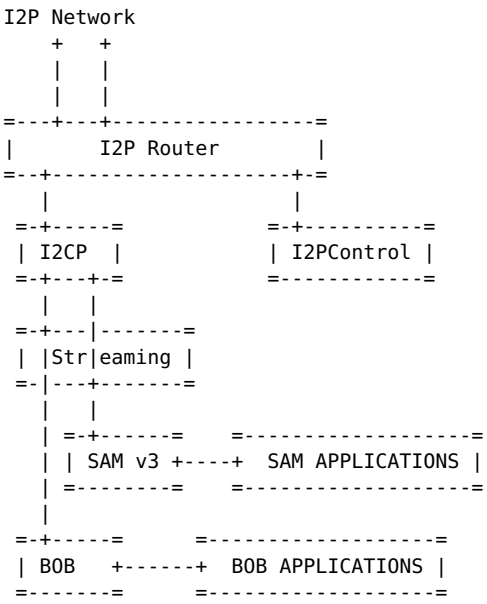
Workshop Title: I2P for Application Developers

Presenters

- idk
- I2P Application Developer
- Mostly my desk or the yard, sometimes a hammock!
- hankhill19580@gmail.com
- https://github.com/eyedeekay
- https://reddit.com/u/alreadyburnt

Current and Former I2P API's



A sketch of the I2P API's, their relationship to eachother, and their relationship to the router.

I2CP

I2CP is the core I2P API, it underlies many of the other I2P API's. It is also the most complex to use outside of Java, but recently several libraries have emerged that provide partial or full I2CP functionality from other languages. C, Java, and Javascript libraries are available, as is one Go library with partial support.

BOB

BOB is a less complicated to use API, but it is currently unmaintained. We don't plan on dropping it soon, but we also have not been adding the new features that are available in SAM Version 3. Many of the best ideas from the BOB API have been ported to the more modern SAMv3 API. Our advice is that new applications should not be using BOB, and that the

SAM Version 1, 2

Deprecated and for all intents and purposes removed.

I2PControl

I2PControl is a little different. Rather than setting up connections between I2P applications, it's used for configuring and retrieving information about the router programmatically.

SAM Version 3

The current recommended API for applications of all types to communicate via I2P is the SAMv3 API. It provides a convenient way to set up, communicate through, and tear down I2P connections. It is designed so that it can implement the API familiar to your programming language in a simple and straightforward way. For example, we can implement a Socket in Java or a net.Conn in Go.

SAM will by and large be the focus of this workshop.

Which API do You Need?

If you need to make connections between applications automatically, then you need the **SAMv3 API**.

If you need to monitor or adjust the I2P router's connection, bandwidth usage, or change it's status, then you need the **I2PControl API**.

If you need to simply check the presence of an I2P router before making connections, one way is to make a quick connection to the **I2CP API**. If you're writing a Java application, the I2CP API may also be a good choice. Besides that, unless you know why you need to use I2CP, you probably just need SAM.

But Why Not Just Set Up I2PTunnel?

I2Ptunnel is good at forwarding existing services to I2P, and it can concievably be used for many applications. It does provide a SOCKS proxy after all. However, setting up i2ptunnels is an involved process, with lots of settings that are intimidating to your users. Using SAM, you set up the connections and apply all the options inside the application itself, **giving you the all-important oppourtunity to set up sane defaults** on behalf of your users.

A good example can be found in applications that are federated with Activitypub. While I2Ptunnel is perfectly capable of making AP applications available over I2P, not many new users will correctly configure the AP-based service correctly on their first try. The process of setting up connections, deciding whether or not to "Bridge" clearnet connections or remain strictly anonymous, deciding tunnel length and the number of tunnels in your destination "Pool," and most other I2P connection-related functions.

What is SAM

SAM is a simple API for controlling **connections** on the I2P i2p router in a way which is familiar to people who write internet applications. To use it, you simply set up a SAM connection and then use it like a streaming connection or to send datagrams, either with or without a repliable address. You can use these connections just like their TCP/IP equivalents for basically every intent or purpose.

Stages of the SAM Setup process

1. Handshake

- This is done so that you can negotiate the features of your SAM client with the SAM service.
- First, establish a socket connection to the I2P router's SAM Port.
- From the client, it's a simple "HELLO" message which can contain optional version and authentication information.
- When the server replies, it will respond with OK and the maximum supported SAM version.

2. Session Establishment

- Once your handshake is complete, you need to establish a session with SAM to control connections.
- To create a session, you send a "SESSION CREATE" message which must declare the type of connection and messaging you will be doing, a unique name for the connection which will allow you to refer to the client, and either a full public/private base64-encoded key pair for the local tunnel or TRANSIENT for a tunnel created with a new keypair for this session.
- Optionally, it can specify a signature type. From now on, it is recommended that libraries supporting SAM 3.1 or greater use ed25519 signatures by default.
- When the SAM service replies, it will return a result of either OK indicating that the session was established successfully or a string indicating the type of error that was encountered. If the session was established successfully, the reply will also include the destination keypair or the newly established session.

3. Connections/Messaging

- Now that you've established a session, you can start making connections and/or sending messages.
- Streaming connections are bi-directional, and can either be connected as a client to a server or listened upon to accept connections as a server to a client. Predictably, the commands you send to the SAM bridge to set up each kind of connection is "STREAM CONNECT" for connections and "STREAM ACCEPT" for listeners.
- Datagrams can be sent after a datagram style session has been established by sending datagrams to the socket. They can be repliable and include a return address or raw and not include a return address.
- Once you have created a Streaming connection, any further communication on that socket will be done with I2P, whether it be an HTTP Client, a connection between bittorrent peers, or any other kind of Streaming communication.

Make your Code Re-Usable!

Because of the deliberate similarity to existing streaming and datagram communications, every language makes it possible to reduce this process to one or two steps at sensible layers of abstraction. Starting from the most similar, like a Socket in Java, a connection in Javascript, or a net.Conn in Go. The actual thing will vary from language to language, but when creating a library, you should probably start soon.

Once you've done that, you've laid the foundation to alter the other network parts of your language. In many cases, it may be possible to forward a connection using the code you've already written, or to replace an underlying structure with your SAM-enabled version.

In a surprisingly short amount of time, you too can develop extensive tooling that makes building new I2P applications and, more importantly, adapting your existing applications to use I2P simple, reliable, and familiar.

Or you can literally just write your own i2ptunnel that you can embed in your existing

application. I did that once. It works really well. I don't think we need a gazillion 'socat for I2P' out there but some would argue we didn't need a third so who am I to judge.

A Very Simple SAM Client

TODO

Bundling an I2P Router with your SAM Application

Sometimes, the details of setting up your SAM application require you to know whether an I2P router is present and ready to accept SAM connections or not. As of release 0.9.42 in a few weeks, this becomes a very easy problem to solve. Let's take a slightly complicated case as an example, a non-JVM, non-plugin application for Windows.

Since there's a good chance your SAM Application is in a non-Java, non-JVM language, it may be difficult or impossible to build as a plugin for the I2P router. If that's the case, then we can't *assume* a router is there.

Since this is a Windows machine, we can't *assume* that a package manager is available with a viable I2P router to install. If that's the case, we'll have to install our own.

Kicking off a child installer with NSIS

```
Section "GetI2P"
  SetOutPath $INSTDIR
    IfFileExists "$PROGRAMFILES\i2p\i2p.exe" endGetI2P beginGetI2P
    Goto endGetI2P
  beginGetI2P:
    MessageBox MB_YESNO "Your system does not appear to have i2p installed.$\n$\nDo you
wish to install it now?"
    File "i2pinstaller.exe"
    ExecWait "$INSTDIR\i2pinstaller.exe"
    SetOutPath "$PROGRAMFILES\i2p"
    File "clients.config"
    SetOutPath "C:\\ProgramData\i2p"
    File "clients.config"
    SetOutPath "$AppData\I2P"
    File "clients.config"
  endGetI2P:
SectionEnd
```

Wait, how can I make sure the router I am bundling is current?

Well here's how I once did it in a Makefile:

```
geti2p: i2pinstaller.exe

i2pinstaller.exe: url
    wget -c `cat geti2p.url` -O i2pinstaller.exe

url:
    echo -n 'https://launchpad.net' | tee .geti2p.url
    curl -s https://launchpad.net/i2p/trunk/+rdf | \
        grep specifiedAt | \
        head -n 3 | \
        tail -n 1 | \
        sed 's|                <lp:specifiedAt rdf:resource="||g' | \
        sed 's|+rdf"/>||g' | tee -a .geti2p.url
    echo -n '+download/i2pinstall_' | tee -a .geti2p.url
    curl -s https://launchpad.net/i2p/trunk/+rdf | \
        grep specifiedAt | \
        head -n 3 | \
        tail -n 1 | \
        sed 's|                <lp:specifiedAt rdf:resource="/i2p/trunk/||g' | \
        sed 's|/+rdf"/>||g' | tee -a .geti2p.url
    echo '_windows.exe' | tee -a .geti2p.url
    cat .geti2p.url | tr -d '\n' | tee geti2p.url
    rm -f .geti2p.url
```

Wait, what if I don't want to make my clients install a JVM

Enter Jlink, i2pd TODO

Abstract

The workshop provides an introduction to the ways an application can be made to work with the I2P Anonymous Peer-to-Peer network. Developers should learn that the use of anonymous P2P in their applications need not be that different than what they are already doing in non-anonymous Peer-to-Peer applications. It begins with an introduction to the I2P plugins system, showing how the existing plugins set themselves up to do communication over i2p and what's good and bad about each approach. Afterwards, we'll continue on to the programattically controlling I2P via its SAM and I2PControl API's. Finally, we'll take a dive into the SAMv3 API by starting a new library utilizing it in Lua and writing a simple application.

We(The I2P Project) would also like to request a table for 4 in the main room for the duration of the conference so that we can hang a banner and have meetings with other users and organizations about incorporating I2P into their projects. *We would like this table even if the workshop proposal is rejected. Whitney advised us that this was the place to apply for table space.*

Intended Audience

Developers of new and existing applications that make use of networked communications that may benefit from enhanced privacy.

Presenter Biographics

idk writes I2P applications and libraries and likes to point out all the cool things you actually can do with anonymous networks. He originated one SAM library and maintains two others, and tries to participate in pretty much all of them. idk also likes to make sure I2P noobs get the help they need on Reddit and blogs about I2P application development.

I2P is a 17-year-old Open-Source project dedicated to enabling privacy and anonymity using advanced cryptography. It enables client-server applications, hidden services, and peer-to-peer applications to work anonymously and provides API's orientated toward application development in addition to browsing.

Materials provided

Each participant will be provided with a detailed handout which will provide a schedule of the workshop with references to additional resources relevant to each section. The full source code of the Lua library that will be developed will be publshed prior to Def Con to be used as a resource during the workshop. Practical activities will be distributed during the workshop to the participants to demonstrate the techniques that will be illustrated.

Audio/Visual/Computer Requirements

Participants are welcome, but not required, to follow along with the examples and practicum provided during the workshop. In order to do so, they will need a laptop with internet access and their choice of Lua interpreter installed. They should also have I2P routers installed, either I2P or i2pd will do, but if they do not wish to install a router, I will bring a machine which will expose the API's they need for testing purposes they can use while they are here. I will also need a way to display examples to the workshop participants, either a projector or a large television would be best.