Bananaphone Pluggable Transport Documentation

Release 0.0.1

Leif Ryge and David Stainton

July 17, 2015

Contents

Bananaphone is a stream encoding toolkit written in Python by Leif Ryge.

- https://github.com/leif/bananaphone

Bananaphone transport is a Tor pluggable transport that uses the Bananaphone codec, written as an Obfsproxy module by David Stainton with help from George Kadianakis and Leif Ryge.

- https://github.com/david415/obfsproxy/tree/david-bananaphone

What are Tor Pluggable Transports? Read about them here:

- https://www.torproject.org/docs/pluggable-transports
- https://www.torproject.org/projects/obfsproxy
- https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports
- https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/180-pluggable-transport.txt

CHAPTER 1

Reverse Hash Encoding

Reverse hash encoding is a steganographic encoding scheme which transforms a stream of binary data into a stream of tokens (eg, something resembling natural language text) such that the stream can be decoded by concatenating the hashes of the tokens.

TLDR: Tor over Markov chains

This encoder is given a word size (number of bits), a tokenization function (eg, split text on whitespace), a hash function (eg, sha1), a corpus, and a modeling function (eg, a markov model, or a weighted random model). The range of the hash function is truncated to the word size. The model is built by tokenizing the corpus and hashing each token with the truncated hash function. For the model to be usable, there must be enough tokens to cover the entire hash space (2^(word size) unique hashes). After the model is built, the input data bytes are scaled up or down to the word size (eg, scaling [255, 18] from 8-bit bytes to 4-bit words produces [15, 15, 1, 2]) and finally each scaled input word is encoded by asking the model for a token which hashes to that word. (The encoder's model can be thought of as a probabilistic reverse hash function.)

CHAPTER 2

Bananaphone usage notes

The tokenization function needs to produce tokens which will always re-tokenize the same way after being concatenated with each other in any order. So, for instance, a "split on whitespace" tokenizer actually needs to append a whitespace character to each token. The included "words" tokenizer replaces newlines with spaces; "words2" does not, and "words3" does sometimes. The other included tokenizers, "lines", "bytes", and "asciiPrintableBytes" should be self-explanatory.

For streaming operation, the word size needs to be a factor or multiple of 8. (Otherwise, bytes will frequently not be deliverable until after the subsequent byte has been sent, which breaks most streaming applications). Implementing the above-mentioned layer of timing cover would obviate this limitation. Also, when the word size is not a multiple or factor of 8, there will sometimes be 1 or 2 null bytes added to the end of the message (due to ambiguity when converting the last word back to 8 bits).

The markov encoder supports two optional arguments: the order of the model (number of previous tokens which constitute a previous state, default is 1), and –abridged which will remove all states from the model which do not lead to complete hash spaces. If –abridged is not used, the markov encoder will sometimes have no matching next token and will need to fall back to using the random model. If -v is specified prior to the command, the rate of model adherence is written to stderr periodically. With a 3MB corpus of about a half million words (~50000 unique), at 2 bits per word (as per the SSH example below) the unabridged model is adhered to about 90% of the time.

Example usage running Bananaphone codec as a standalone app

encode "Hello\n" at 13 bits per word, using a dictionary and random picker:

```
echo Hello | ./bananaphone.py pipeline 'rh_encoder("words,sha1,13", "random", "/usr/share/dict/words'
```

decode "Hello\n" from 13-bit words:

```
echo "discombobulate aspens brawler Gödel's" | ./bananaphone.py pipeline 'rh_decoder("words,sha1,13"
```

decode "Hello\n" from 13-bit words using the composable coroutine API:

```
>>> "".join( str("discombobulate aspens brawler Gödel's\\n") > rh_decoder("words,sha1,13") )
'Hello\\n'
```

start a proxy listener for $sshhost, using markov encoder with 2 bits per word:

```
socat TCP4-LISTEN:1234,fork EXEC:'bash -c "./bananaphone.py\\ pipeline\\ rh_decoder(words,sha1,2)|soc
```

connect to the ssh host through the $proxyhost:

```
ssh user@host -oProxyCommand="./bananaphone.py pipeline 'rh_encoder((words,sha1,2),\"markov\",\"corpu
```

same as above, but using bananaphone.tcp_proxy instead of socat as the server: server:

```
python -m bananaphone tcp_proxy 1234 localhost:22 rh_server words,sha1,2 markov corpus.txt
```

client:

```
ssh user@host -oProxyCommand="python -m bananaphone tcp_client $proxyhost:1234 rh_client words,sha1,2
```

start a webserver at localhost:8000 with an interactive composition interface which shows all tokens available for encoding each word of input:

```
python -mbananaphone httpd_chooser asciiwords,sha1,8
```

CHAPTER 3

Bananaphone Tor Pluggable Transport

Bananaphone obfsproxy module can be used with tor in managed mode or in external mode to obfuscate traffic.

CHAPTER 4

Obfsproxy bananaphone usage

```
$ obfsproxy bananaphone
usage: obfsproxy bananaphone [-h] [--corpus CORPUS]
                             [--encoding_spec ENCODINGSPEC]
                             [--model MODELNAME] [--order ORDER] [--abridged]
                             [--dest DEST] [--ext-cookie-file EXT_COOKIE_FILE]
                             {server,ext_server,client,socks} listen_addr
```

- encoding_spec: string containing comma seperated values consisting of a tokenizer, hash function and bits per token. For instance: words,sha1,2
- corpus: the text file to be used as a data source to generate the model
- model: currently only "markov" and "random" (random weighted) models are implemented
- order: number of previous tokens which constitute a previous state
- abridged: if set to "true" then all states in the model which do not lead to complete hash spaces will be removed

All options are required except "abridged". Bananaphone will only advertise the "encoding_spec" transport option to the Tor bridge database.

CHAPTER 5

Bananaphone Pluggable Transport Threat Model

An observer only needs to guess your tokenization function, hash function and word size to see your encapsulated traffic.

CHAPTER 6

Test bananaphone obfsproxy transport

This is how I test obfsproxy transports locally on my Debian wheezy system.

First, build the latest tor:

```
git clone https://git.torproject.org/tor.git
cd tor
./autogen.sh
./configure --prefix=/opt/tor
make
sudo make install
```

Install obfsproxy in a python virtualenv (from a previously verified recent version of python virtualenv) :

```
./virtualenv-x.xx.x/virtualenv.py $HOME/virtualenv-obfsproxy
. $HOME/virtualenv-obfsproxy/bin/activate
pip install git+https://github.com/david415/obfsproxy.git
```

Setup a torrc for the client and for the bridge:

```
cat <<EOT>bananaphone-client-torrc
Log notice stdout
SocksPort 8040
DataDirectory ./client-data

UseBridges 1

Bridge bananaphone 127.0.0.1:4703 modelName=markov corpus=$HOME/corpera/text1 encodingSpec=words,sha1
ClientTransportPlugin bananaphone exec /home/human/virtenv-obfsproxy/bin/obfsproxy --log-min-severity
EOT
```

```
cat <<EOT>bananaphone-bridge-torrc
Log notice stdout
SocksPort 0
ORPort 7001
ExitPolicy reject *:*
DataDirectory ./bridge-data

BridgeRelay 1
PublishServerDescriptor 0

ServerTransportListenAddr bananaphone 127.0.0.1:4703
ServerTransportPlugin bananaphone exec $HOME/virtenv-obfsproxy/bin/obfsproxy --log-min-severity=info
ServerTransportOptions bananaphone corpus=$HOME/corpera/text1 encodingSpec=words,sha1,4 modelName=mar
EOT
```

First start your tor bridge:

```
/opt/tor/bin/tor -f bananaphone-bridge-torrc
```

And then start our client side tor:

```
/opt/tor/bin/tor -f bananaphone-client-torrc
```

For troubleshooting it is helpful to watch the obfsproxy log outputs and tcpdump output. Here's some sample tcpdump output:

```
sudo tcpdump -A -ni lo port 4703

He reached down some extent Party propaganda. white-jacketed chestnut palm bulk yapped Syme the fende
15:09:03.890842 IP 127.0.0.1.54119 > 127.0.0.1.4703: Flags [.], ack 3725471, win 1007, options [nop,n
```

CHAPTER 7

Test obfsproxy bananaphone in external mode

First start the server side obfsproxy:

```
obfsproxy --log-min-severity=info bananaphone --order=1 --model=markov --corpus=/usr/share/dict/words
```

Start the TCP service on 127.0.0.1 port 3600:

```
nc -l 3600
```

Start the client side obfsproxy:

```
obfsproxy --log-min-severity=info bananaphone --order=1 --model=markov --corpus=/usr/share/dict/words
```

Connect to the client side obfsproxy:

```
nc 127.0.0.1 3602
```

CHAPTER 8

Run a Tor bridge using obfsproxy in external mode

...

CHAPTER 9

Indices and tables

- genindex
- modindex
- search