Jacob Freck
12/12/15

MeasureSPV

The project intends to explore and uncover information about the ways in which most end users interact with Bitcoin network -- through thin or lightweight wallets (SPV clients). With so many users choosing lightweight clients, with the benefits of low storage demands and lower computational overhead, research into the security risks and usage of these clients becomes essential to measuring the Bitcoin network's health. Primarily, this project aims to aid those who wish to better understand how SPV wallets work on an implementation level and interact with the larger network. From this analysis arises both security risks and opportunities to better measure the use of SPV wallets. This project aims to provide answers on all of these fronts and eventually produce an estimate for the transaction volume being received by SPV wallet addresses.

In order to begin to estimate the transaction volume being output to SPV wallet addresses, a sophisticated understanding of how SPV wallets work is necessary. To this end, Bitcoinj will be used as a case study into SPV wallet implementations. Bitcoinj is a Java library primarily authored by Mike Hearn that is used by many popular SPV clients, like Hive, MultiBitHD, Bitcoin Wallet for Android, and other services. The library serves as an early, but fairly standard and well documented study in SPV wallet implementations, with many of the design decisions appearing in other SPV implementations. For discussing SPV interactions with full nodes, the latest released version of Bitcoin Core (v0.11.0) will be used, as it is the most used full known implementation.

An SPV client holds all of the block headers, but none of the actual blocks necessary to check and verify transactions or value currently held in addresses. In order to still achieve this functionality, the SPV wallet connects to full nodes on the network, and requests this data by sending certain messages and receiving certain response messages, all outlined in BIP 37. Rather than directly send full nodes the addresses in the wallet, SPV wallets embed the addresses they hold (or are interested in) in a probabilistic, space-efficient data structure called a bloom filter, which guarantees all embedded addresses match the filter. Bloom filters supposedly add privacy by also allowing false positives, intended to hide which addresses the SPV wallet is actually looking for. The SPV client creates the bloom filter and sends it to a full node with the filterload message defined in BIP 37. Seen here, in Bitcoin Core, the message is interpreted and handled by the connected full node to which it is sent. Eventually, the full node searches through the entire UTXO set's output addresses as well as incoming relayed transactions output addresses for transactions relevant to the Bloom filter (note: each full node undergoes this process for each connected SPV peer). Once all transactions with outputs that match the bloom filter are computed, the full node returns a partial merkle block that the SPV client uses to verify a transaction or compute the value stored by an address. This offloading of computational overhead and storage onto full nodes allows SPV wallets to have minimal hardware demands, thus appealing to a wider range of users. However, it also opens wallets up to potential security risks.

Gervais et. all tested the effectiveness of Bitcoin's bloom filter privacy in their paper *On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients*. Their research demonstrated the potential risks all SPV clients face simply by interfacing with full nodes through these bloom filters. The researchers showed that it is possible to determine with a high probability which addresses the bloom filter request was looking for if either of the following conditions were true: 1) the number of addresses generated by wallet and embedded into the filter was small (below 20), or 2) the number of addresses was very different from the maximum that the bloom filter can hold (in Bitcoin, this is 100 addresses). The actual addresses requested by a bloom filter could be guessed with an even higher probability if the

SPV client sent the same full node a bloom filter request twice, as each bloom filter contains some randomness, and would thus contain all of the same 'real addresses' with likely a different set of false positives. This research intends to demonstrate the flaws in the SPV privacy model, however, the same conclusions can be leveraged as a tool to study SPV wallet use on the network.

In order to record the interaction between SPV clients and full nodes, a forked version of Bitcoin Core v0.11.0 was set up. This forked node listened for and recorded data about incoming filterload, filteradd, and merkleblock messages. Once connected to the network itself, the node should allow for SPV clients to connect to it. These SPV peers then, will send the filterload message, which will be recorded, and the full node will return the merkleblock message, which will also be recorded. Over a suitably long period of time, the node should begin to compile a large enough amount of data to estimate the output volume to SPV wallets for that time period. A few assumptions are made in order to obtain this estimate. First, the estimate relies on the assumption that all nodes in the network are of equal likelihood to be connected to (this assumption will be addressed later). Additionally, the estimate assumes that this forked node receives a proportionate amount of connections from SPV wallets. Or in other words, all full nodes receive equal amounts of peer connections to SPV wallets. The first assumption will be discussed later in the report, in relation to how SPV wallets connect to the network. The second assumption should be accounted for by distributing multiple nodes across the network. By having multiple nodes, each connected to a different set of peers, each running in a different geographic location, and each recording messages the same, differences in network traffic across these variables can be better accounted for in the estimate. Current project progress is here: multiple nodes distributed across multiple AWS instances are sitting on the network, recording incoming messages and gathering data. So far, no estimate can be made.

To understand why no estimate has been made thus far, we return to assumption 2, that all connectable full nodes are of equal likelihood to be connected to by an SPV wallet. The faults of this assumption have limited ability to effectively record incoming SPV messages. SPV wallets must connect to the network every time they start in order to retrieve the most recent information about the addresses they hold and in order to propagate transactions their users wish to create. Nothing in the protocol dictates how an SPV client must connect to the network (or a full node for that matter), however, Bitcoinj opts to connect via DNS seeds, similar to how Bitcoin Core connects to the network for its initial connection.

DNS seeds are community run, known DNS names that, when connected to, return a list of IP addresses corresponding to known Bitcoin nodes. Some DNS seeds are static, meaning that the IP addresses that they return are more likely to be inactive. Most, however, are dynamic. They use crawlers that continually send getaddr messages to different nodes on the network, obtain all of its peers, and repeat the process on the peers. The dynamic DNS seeds update their list of IP addresses as the crawler finds new nodes that come online and old nodes that exit. Bitcoin core has its own implementation of a seeder, authored by Pieter Wuille, and a corresponding code of conduct for operators of the dns seeds. Notably, however, these seeds are unverifiable and not authenticated; nothing prevents operators from only releasing certain IP addresses, or blacklisting certain IP addresses. There are also no guarantees about how often the list of IP addresses each seed returns is updated. These DNS seeds are also highly centralized with Bitcoinj and Bitcoin Core having a combined total of 7 different DNS seeds (4 overlap), seen here in Bitcoinj and here in Bitcoin Core. This sort of centralization arguably violates principles of decentralization central to Bitcoin's success by restricting the network access to certain gatekeepers, certainly raising the power these individuals have in the network (note: most of these seeds are run by core developers).

To connect to the Bitcoin network, Bitcoinj and all of the SPV implementations built on it default to using these DNS seeds to find peers. Bitcoinj does not store previously connected IP addresses, like Bitcoin Core does, in an attempt to connect more quickly and reliably to the network. Many SPV implementations default to dynamic DNS seeds for these same reasons, especially since previous peers often exit or change IP addresses. Bitcoin Core's crawler compiles statistics on available nodes and ranks them based on reliability across different time periods (2hr, 8hr, 1d, 7d, 30d). In practice, the rate at which these crawlers are run, and the rate at which DNS seeds update is uncertain. After running multiple connectable nodes for a few days, none of these node's associated IP addresses appeared in the list of available nodes (supplied by one of the seeders, here). Thus, the assumption that all full nodes are equally connected to by SPV wallets has some faults: nodes are ranked, assuming a hierarchical structure where one does not otherwise exist in the network. The lack of transparency in how DNS seeds return known IP addresses, and the variation in how SPV wallets choose IP addresses returned by the DNS seeds makes it unclear whether this hierarchical ordering of nodes actually causes disproportionate connections to these reliable (read: trusted) nodes. If such an effect exists this may mean that SPV wallets are even more susceptible to privacy loss, as they would be more likely to reconnect to the same nodes, which, according to Gervais et. all, drastically reduces the privacy gains of bloom filters. Further investigation and testing would be necessary to see if any such impact truly exists.

Regardless, SPV wallets (at least all of those that default to bootstrapping into the network through DNS seeds) will only connect to nodes with IPs returned by the seeds. Inability to successfully have the nodes this project set up recognized by the seeds has thus far prevented an estimate on the network traffic. If no DNS seeds return the IP address of the listening nodes, then no (or very little) data can be collected on SPV wallet use. The next step for the project is clear: investigate how the DNS seeds work, and ensure that the deployed nodes listening for SPV messages are available to SPV clients that connect to these DNS seeds. Then, after a suitably long time listening on the network, the collected data can be processed and studied. The nature of the data collection allows for a wide range of tests available, but this report will outline how the transaction volume output to SPV wallets will be estimated once a large enough data set is obtained.

The set of recorded bloom filters sent to all of the listening nodes will be tested against all output addresses in transactions included in blocks mined since the node started listening (the other addresses can be ignored as there is no value output to them). The set of matching addresses will be recorded, along with the value output to them. The sum of all of the output values for all of the matching addresses will be calculated, giving the total output volume seen my all of the listening nodes since they began recording data. From here, the output volume will be extrapolated by multiplying it by the number of listening nodes, and dividing the product by the total number of nodes on the network. Under the assumption that the listening nodes saw a typical distribution of filterload messages and received those messages from representative SPV wallets, this will produce a rough estimate of the total output volume conducted by SPV wallets.