Darkening the Heart of Phantom

Jeff Tyson
jmtyson@cs.ucsb.edu

ABSTRACT

Many Internet users see anonymity as an inherent property of the Internet. The truth is anonymity is not inherent, and in fact strong anonymity is quite challenging to achieve. Nevertheless, there are a number of solutions to this problem that offer strong anonymity, each with their own benefits and drawbacks.

In this work, we will focus on the Phantom protocol. The protocol is designed to be general, secure, decentralized, fast, and compatible with existing protocols. Furthermore, this work will discuss the architecture of the Phantom implementation, as well as the changes that were made in order further the implementation. Finally, we will discuss some of the challenges facing Phantom in order to allow it to achieve its goal of strong anonymous communication using the Internet.

1. INTRODUCTION

The uptake of anonymous networks has been quite successful for the past decade. The Onion Router (Tor) is probably one of the mostly widely used anonymity protocol in use today. It has more than 1500 relay nodes, and hundreds of thousands of users [7]. Tor also has usage around the world in at least 126 countries with some of the highest proportion of users in Germany. Most of that traffic is HTTP traffic, but a disproportionately large amount of the traffic is BitTorrent [9]. The usage of bulk transfer non-interactive protocols has caused a significant slowdown for interactive protocols.

Tor is one example of design choices that could make up an anonymity protocol. There are alternatives to Tor that provide anonymous communication. Almost all the alternatives focus on some type of decentralized architecture. The Phantom Protocol is one such protocol which also offers other new features in anonymous communication. As of yet, there are still a few features that have yet to be fully implemented. Exploring the reasoning and implementation behind these features—as well as already implemented features—is focus of this work.

2. BACKGROUND

The Internet has long been considered an anonymous communication medium by most of its users. With this in mind, a number of people use the Internet to do or say what they otherwise would not. To be clear, anonymity can be used in a number of ways, each of which fall somewhere on the continuum of positive to negative. Some of the most widely cited positive examples include whistle blowing without fear of reprisal, and speaking out against repressive governments. The opposite side of the spectrum fall in to the category of non-accountable illegal actions. Often times, it is these illegal uses that drive lawmakers to try to outlaw anonymity entirely. In most cases, legislation aimed at eliminating anonymity has been ruled too broad and far reaching [5].

The truth about Internet anonymity is that it is closer to pseudo-anonymity. That is, communication is not inherently anonymous because identifying information can be retrieved if needed. Perhaps there was a time when the Internet did have true anonymity. At some point though, the Internet lost the notion of a centralized authority, and became a collection of independently controlled networks. This property resulted in de facto anonymity, the likes of which had never been seen before. Given time though, many of the networks started to become more advanced in their record keeping and tracking. This can be seen in numerous instances of users being tracked for good or bad based on their IP address alone. Additionally, governments are attempting to maintain surveillance of large parts of the Internet by wiretapping large Internet service providers. Given the advances of technology and legislation, the current state of Internet anonymity is precarious. Hence the need to be innovative in making anonymous communication possible.

3. THE PHANTOM PROTOCOL

3.1 Design

The Phantom Protocol was originally proposed by Magnus Bråding at DEFCON 16 [2]. It is designed based on these assumptions [2]:

1. The traffic of every node in the network is assumed to be eavesdropped on (individually, but not globally in a fully correlated fashion) by an external party.
2. Arbitrary random peers participating in the anonymization network/protocol are assumed to be compromised

and/or adverse.

3. The protocol design cannot allow for any trusted and/or central entity, since such an entity would always run the risk of being potentially forced offline or manipulated, if for no other reason due to large enough quantities of money (or lawyers) being "misplaced."

From the design assumptions comes the idea that the protocol must be decentralized, trust should not be assumed wherever possible, and probabilistic algorithms should be used. The protocol design directives also states that given the choice between better performance and security, security should be the winner within reasonable bounds.

3.2 Goals

The elements in the design described so far lead to a number important goals the protocol is aiming to achieve.

3.2.1 Complete decentralization

As many of the systems on the Internet become more organized and centralized, there are a number of properties of decentralization that are useful to the phantom protocol. The most important of which is that decentralization makes the system more chaotic, and thus harder to understand and subsequently break. This is a different design choice than Tor, which chooses to keep a centralized list of independently controlled relay and exit nodes. It turn outs that this goal may be the most challenging of all to satisfy because of the conflicting nature of the protocol's needs.

3.2.2 Maximum resistance to all kinds of DoS attacks

Assuming all the design principles hold, the only way to stop the protocol is to use a denial of service attack. Thus, all of attempts must be made to mitigate this threat.

3.2.3 Theoretically secure anonymization

As with many things in security, it is a wise idea to build the protocol based on theoretically secure principles. This goal may slow down the initial implementation but it is crucial to ensuring the protocol stands the test of time.

3.2.4 Theoretically secure end-to-end encryption

The protocol should emulate direct communication. That is there should be no opportunity for eavesdropping.

3.2.5 Complete (virtual) isolation from the Internet

Isolation from the Internet is an important goal because anything that travels to the Internet weakens anonymity properties. Tor decided not to isolate the network from the Internet which has led to some negative consequences. One of which is the "exit-node problem". In this situation, a Tor exit node appears to be responsible for all of its traffic, both good and bad. This leads to a number of legal, and ethical issues that lead most people to avoid running exit nodes [12]. The next problem is that an insecure protocol allows confidential information to be easily gathered at an exit node [9]. Finally, protocols that are not designed with anonymity in mind have a tendency to leak information through alternate channels of communication [1].

3.2.6 Maximum protection against protocol identification/profiling

It is well known that technology has given ISPs the ability to throttle/block certain types of traffic in realtime. This might be because the stream consumes a lot of bandwidth, or it is considered subversive material by the local government. While it is very challenging to make any protocol completely unidentifiable, the goal is to make it difficult to do in realtime. Hence, it would be possible to stay ahead of the profiling to some degree.

3.2.7 High traffic volume and throughput capacity

Many existing anonymous protocols are not designed for high traffic volumes. Tor for example focuses on low latency interactive traffic both in principle and in the implementation. A usable alternative to the normal Internet must not alter throughput.

3.2.8 Generic, well-abstracted and backward compatible design

A generic design stands to test of time much better than a more specific one. Creating a well-abstracted protocol allows for compartmentalization. In one sense, it means that a design change in one area does not effect the other parts of the protocol. In another sense, it means that people can fine-tune the protocol to their needs. An example of this could be using more nodes in the onion routing process, or choosing nodes based on some criteria important to the anonymous user. Finally, since the goals look for secure end-to-end encryption and isolation from the Internet, it is still possible to tunnel existing protocols through the Phantom protocol. This is important because people do not need to reinvent all existing protocols in order to use the Phantom protocol. It should be noted that great care should be used when using a protocol that is not meant to be used as an anonymous protocol [1]. For example, most web browsers tell a great deal identifying information to web servers by default. The Tor project and Electronic Frontier Foundation have developed privacy filters to combat this, but it is still non-intuitive to many users.

3.3 Protocol details

There are many small details to the Phantom Protocol. Still, there are far less fundamental details needed to obtain an understanding of the protocol.

At the most basic level, there is each and every node on the Internet itself. Some of these nodes may choose to participate in the Phantom protocol. Every node running the protocol is also presumably running the distributed hash table (DHT). The DHT is responsible for storing all the information about participating nodes.

3.3.1 Path creation

Participating nodes can be grouped together to form a combined path. The path has endpoint nodes which are a source or destination of traffic for the protocol. Each endpoint node also has an anonymous protocol (AP) address associated with it. These are used as unique identifiers for each and every node on the network. The APs are also stored in the DHT for later retrieval. Furthermore, endpoint nodes

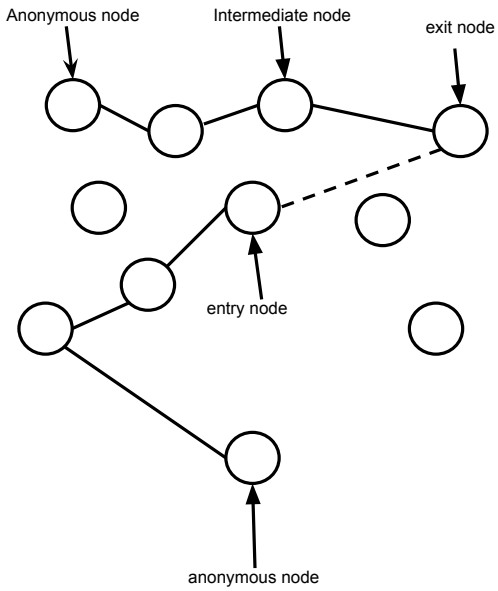

Figure 1: An example a completed phantom path.

may choose to be anonymous or non-anonymous. Choosing non-anonymous, for example, might be a smart choice for company looking to provide services without learning the user's true identity. Finally, the last type of nodes, called intermediate nodes, are all the nodes in between the endpoint nodes which establish the paths and relay traffic.

For a number of reasons, the path is only one of two pieces based on the request of the endpoint nodes. An example of this can be seen in Figure 1. Each endpoint node can request to have either an entry or exit path built. The communication in this model can only be initiated in one direction. That is, an exit path must initiate a connection with an entry path.

To create a path, an endpoint node first chooses a group of nodes to use as intermediate nodes. The only real requirement for these nodes is that they are not colluding in some way to harm anonymity. Without any information about the network, a random selection of nodes from the DHT should suffice. With additional information, a node could choose based on country location, latency, bandwidth, web of trust, etc.

The next step is to designate nodes as either X or Y nodes. The Y nodes only exist to build the path, while the X nodes stick around to act like traditional relay nodes. For the Y nodes, there must be at least the number of X nodes minus one in the path. Each Y node is placed between each X node. The purpose of the Y node is to act as a buffer between the X nodes. As we will soon see, each node only knows about the previous and next nodes in the path, making collusion more challenging. Finally, the endpoint chooses one X node to be a terminating X node. The result is visualized in Figure 2.

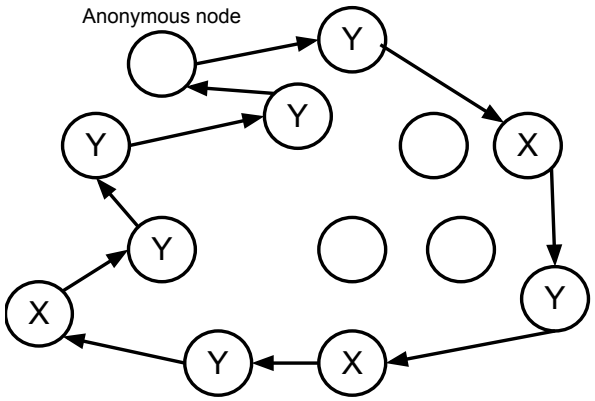

Figure 2: An example of Phantom in the path building stage.

The next step is to prepare information for each node in the path. The information contains each node's role, the keys it should be using for communication, and hashes to check the integrity of everything. The initiating node sends this information out to the first node in the sequence, along with a unique value. Each node unpacks its information, learns of its role, and forwards the information to the next node in the sequence. Finally it reaches the initiating node, where the initial value along with a number of other hashes are checked.

In the second round, things are nearly the same except that the X nodes now attempt to connect to each other directly. Additionally, a routing table entry is prepared for the terminating X node to publish to the DHT. After each connection passes through a Y node, it disconnects from its peers and returns to the state it was in beforehand.

3.3.2 Tunnel creation

The routing paths are created to be reused over a period of time in order to communicate with specific endpoints. Routing tunnels on the other hand are created for an individual end-to-end connection. Keep in mind though, an end-to-end connection actually has two routing paths involved.

There are actually two separate processes to create a tunnel depending on if it is an outgoing or incoming connection. The only difference occurs at the terminating X node. This is done mostly because it makes it more difficult to ascertain which direction flows toward the anonymous node. For our purposes, it will be viewed from the point of an outgoing tunnel. First, the anonymous node picks a random piece of data. This is then forwarded to the terminating node, and encrypted with keys given by the anonymous node in path creation. The terminating node adds a crypto initialization block with some known property. Afterwards, the whole processes is now done in reverse with the addition of the crypto initialization block. Finally, when it arrives at the anonymous node, both the random data and crypto block are fully decrypted. The anonymous node takes the crypto block, and decrypts it using a bruteforce method. It can do this quickly given that it has all the possible keys the X

nodes could have used.

Once the keys are obtained, the anonymous node fills another crypto block with an AP address and port to which it wants to communicate with. The exit node then queries the DHT to find the entry node for the AP address. The entry node receives the request, and follows a similar process to the above. After all setup procedures are finished the data is decrypted/encrypted and forwarded as soon as it is seen by each node. To ensure end to end confidentiality and authentication the tunneled protocols are encapsulated inside SSL transparently.

4. ARCHITECTURE

The Phantom Protocol itself is very well laid out, but there are still a number of design decisions left to an implementer. Luckily, many of the design decisions have been made [13], and in this section we will go through them in detail.

First of all, the language of choice used is C with strict compilation flags and adherence to POSIX.1-2001-standard. In some ways, the POSIX standard can be quite restricting as a choice, but it does allow for easier portability. This is seen in Phantom's use the `snprintf` function, which can not be included in a header file as it is not part of the POSIX standard. Furthermore, Phantom is implemented on the Linux operating system. This further restricts the use of the POSIX standard, as there are simply some functions missing from the GNU C Library.

The implementation makes a few changes on the protocol specification, and solidifies other aspects. First, the AP addresses are strictly IPv6 addresses with the prefix fc00::/8 and fd00::/8. The fc00::/8 is proposed to have a central authority where the other prefix for lab use. Of course, there is no reason someone could not change the implementation ever-so-slightly to use their own prefixes. The use of AES256 and RSA, TLS, and SHA are some of the important cryptographic tools in use. These are all provided by the OpenSSL library. All of this information is encapsulated in a protocol buffer provided by libprotoc. The use of protocol buffers makes the protocol much easier to change and understand while looking at the source code, while still remaining efficient on the wire.

The implementation also solidifies the one server, one port idea—specifically, one TCP port. Many people are quick to point out that tunneling all types of traffic over TCP is inefficient and also suffers from flow control problems [3]. These claims may or may not apply to Phantom, but the challenges of implementing UDP are high. For one, TLS over UDP (DTLS) is only a few years old under OpenSSL, and it suffers from a number of shortcomings. Then there is the need to implement a userland TCP stack to reproduce expected TCP behavior This would be quite challenging and prone to errors.

4.1 The DHT

4.1.1 Overview

Kademlia is the source of inspiration for the DHT [8]. The basic idea of any DHT is to apply order to what might otherwise look like chaos. The DHT is a collection of network

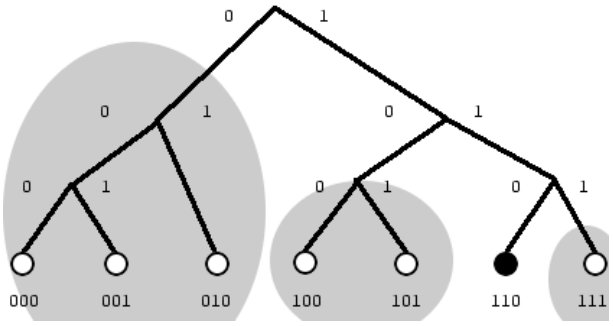

Figure 3: An example of Kademlia network with 8 possible IDs. The graph is organized from the perspective of the node 110. The grey areas represent the buckets that each ID would fall into.

nodes, each with their own ID, that keep track of key-value pairs. Node IDs themselves could be considered a special key-value pair type, because they get stored in the routing table. A Kademlia routing table is composed of a list of k entries for each bit of the ID. Each node organizes the table according to whatever is closest to its own ID. This is achieved by XORing the IDs together, finding the least significant bit of difference, and then storing the ID in a list related to the particular bit. This list is also known as a bucket. Nodes are added to the buckets as they encountered, and they are only removed when they are shown to be offline.

Since the DHT is a data store, there is a procedure to find data. The nodes find information using consistent hashing [6] as follows:

1. The key the node is searching for is hashed.
2. The node finds the closest node in its buckets and asks for the value belonging to the key.
3. The queried node either returns a value, or the next k closest nodes it knows to the key.
4. The previous two steps are repeated until either a value is returned or no new nodes are returned.

4.1.2 Phantom specifics

There are many implementations and studies of Kademlia based DHTs in the world. Some examples include KAD as part of Overnet, and BitTorrent. Many of these well founded constants and ideas were kept, but some were changed to fit the needs of the Phantom Protocol. All communication preformed with the DHT is encrypted with SSL. Each Phantom node derives its ID based on the SHA-1 hash of its communication certificate. This makes the ID 160 bits long, and leads to 160 buckets per node. Each bucket allows for 20 entries (k), even though some buckets will never be filled. The buckets are pinged by opening an SSL connection to the node in question. This is done for every entry in every bucket at least once an hour. The buckets are also kept fresh by randomly querying a node that falls within that bucket's space every hour.

The Phantom Protocol uses the DHT to store the contact information for a particular AP address. To find the key for this value, a node hashes the AP address and then proceeds to query progressively closer and closer nodes to the key ID. The implementation does this with 3 different queries at once, in order to facilitate quicker look ups. Also, every new node found is added to the appropriate buckets if they are not already full. Finally, each value is signed with a routing certificate in order to give a form of authenticity.

4.2 End to end communication example

To show the basic operations of the Phantom implementation, we will step through a simplified example. First, the server is started along with a thread poll. The primary purpose of the server module is to handle incoming connections and handle the various roles a node may take on during its lifetime. The might include, being an X node, entry node, or exit node as well as waiting on connections for other modules.

Soon after the server is created, each node starts the DHT module. It starts by creating the routing table based on the ID which is a hash of the communication certificate. It should be mentioned that all certificates are serialized by a module that flattens them and presents them as a byte array. The next step is to create a disk store for each value that may be stored by the DHT on this node. There are also a ping, update, and house keeping worker which all aid in the process of maintaining the DHT.

After the DHT is successfully started, a node would start to build an entry path and/or an exit path. Before the node builds its actual long term path, it will build a temporary exit path to publish a routing entry containing all the contact information to reach its AP address. To do this, first the path module will grab the necessary number of nodes from the DHT. It will then take these nodes and assemble them into a sequence and flag the nodes as X or Y nodes with one node being a terminating node. The path will be reversed with a 50% chance. It will continue to follow the procedure laid out in the protocol specification until the path is built. At this point, the exit node will publish the routing table entry to the DHT, communicate the result to the anonymous node, and finally shut down. Assuming the routing table entry was successfully published, the node finishes building the long term path. A TUN interface is then brought up and assigned this address so that regular IPv6 supporting applications can communicate using Phantom.

Now suppose that a user with an exit path would like to SSH into another anonymous node with an entry path. The packets are sent to the device by SSH, then they are read by a thread in the TUN module and passed on to the Phantom tunnel module. Here, the tunnel module would indicate to the exit node that it would like to create a tunnel that connects to the AP:port combination. The exit node at the end of the path would then query the DHT for the entry node information, and proceed to connect to it. When the tunnel is successfully created, both ends can access it through their TUN device. It should be noted, that after successful creation the entry node can now access the exit node as if it were an entry node. This is only the case after successful tunnel creation between to AP addresses.

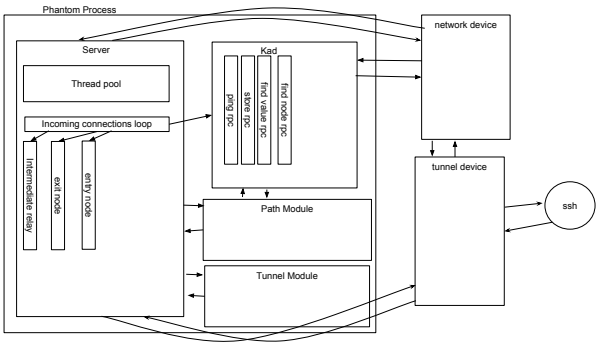

Figure 4: The basic Phantom Architecture.

5. IMPLEMENTATION ADDITIONS

The implementation of the Phantom protocol is currently somewhere between proof of concept and ready for its intended use. The main objective of this work is to further the implementation of the protocol such that it is ready for regular use. In the following sections, we will discuss the testing framework, additions, changes, and on-going problems to the Phantom implementation.

5.1 Development and testing architecture

One of the major areas of contributions from this project test framework for use with development and testing. The reason such a framework is needed is because Phantom is designed to run in a large scale distributed manner. This means we need of about 20 nodes at a minimum to test that a change does not break the build. It should also be possible to allow Phantom to run on a single machine running on 20 different ports, but as of yet that is still a work in progress.

To accomplish testing, a cloud computing service using Eucalyptus was utilized. The script to accomplish this, written in python using the boto library, starts by reserving the number of instances asked for by the user. After they boot up, it patches and upgrades various parts of the operating system to a unified state such development can be carried out smoothly. Then, it downloads the Phantom source code, compiles it, and installs it into the system. Finally, each Process is run inside of a tmux session so that the output can be inspected later.

After everything is started up, the script drops into iPython, where it can be used to script changes to the entire environment without recreating all of the images. By using tmux as persistent terminal, it makes scripting a very simple and straight forward process because all direct key presses can be simulated through commands. GNU screen can also achieve this, but there seem to be a number of bugs that prevent it from working like expected. Some of the possibilities of scripts that could be written include:

- full connectivity testing
- latency information gathering
- bandwidth gathering

- restart phantom (under GDB, valgrind, plain, etc) process
- recompile the code
- check for interesting output during runtime

5.2 Changes

In much of the Phantom literature, there is noticeable ambiguity in regards to the DHT. One of the goals was research DHTs, and improve the design wherever it was needed. In general, much of the DHT is implemented and quite complete.

The first part that was worked on ended up taking most of the time requiring the most small changes to many areas of the protocol. It was in regards to publishing the routing table entry for the AP addresses. Initially, the routing table entry was being published once but it was never republished. As part of Kademlia, it is generally required that nodes republish their records every hour, and that the original publisher publish its value every 24 hours. The problem starts to expand once it is realized that the original publisher must be identifiable on the anonymous network.

In order to do this, the publisher would simply need to sign the routing table entry with their route certificate. The nodes that would store this value would also need to keep the routing certificate close by. This was achieved by putting the packed routing table entry inside a new protocol buffer message along with a signature of the data, signature length, and serialized routing certificate. Signing the same data every 24 hours is not proof enough for the nodes that the original publisher is still alive. Thus, a 32-bit version number was introduced into the routing table entry to help this process along. The value of 0 is reserved for future use, possibly as a permanent revocation of the routing table entry. Beyond 0, the value is incrementally increased for every publication. When a node receives the publication, it first checks if the signature and information contained within are valid, then it stores it on the disk cache. To ensure that stale data is not being seen as new data, it is also important to keep a list of deleted entries.

In the course of tampering with the DHT, it was necessary to overhaul the disk cache module, which is used by the DHT to store values. This is mostly because if two phantom nodes ever ran on the same operating system, they may end up having file conflicts. It is also the case that the original way of writing files was locally insecure. The most secure methods were attempted in creating files, but as it turns out they are part of POSIX, but not part of GLIBC in Ubuntu 10.04. However, the latest version of Ubuntu does have the necessary function. In the meanwhile, the `tempfile()` function was used—which has the only downfall of not being portability secure. There was also a significant amount of trouble attempting to use the higher level file I/O functions, so the code simply uses the lower level I/O functions. The disk cache was also changed to be indexed by a kademlia key in memory rather that would map to a unique file.

To have a fully functional DHT, it is important that each node keeps its information current. The implementation has a housekeeping thread for kademlia and for the disk cache. Since routing table entries can be republished, they can also now be deleted from the disk cache. Kademlia also tries to insure data integrity by having each node republish all of the values it has stored. Also, when a node searches for a value and successfully finds it, it will store the value at the "closest node to the value" that did not have the value. Additionally, each node that is contained in a bucket is pinged to ensure it is still alive and a search is performed in each bucket to keep them populated. Finally, in order to keep the network more tidy, nodes keep keys for an amount of time that is inversely proportional to the distance between the node and the key. This is to facilitate good network caching on "hot keys" but not allow them to swamp the entire network for a long time.

Action	Time (sec)
bucket refresh	3600
republish values	3600
republish AP info	86400
expire value	86420

The default timeouts for housekeeping are given in Table 5.2.

5.3 Additions

In the course of doing work and research it became obvious, for reasons that we will see later, that some kademlia RPCs would need anonymity. Clearly, publishing an AP address from the anonymous node itself is a terrible idea. It is also useful to be able to perform the kademlia store, and find RPCs without the loss of anonymity.

The AP publishing was already implemented, but to facilitate the other RPCs it was generalized for their support. Doing this required a new protocol buffer type, as well as new flags for the path creation process. When the function is called, it will start by creating an exit path. It will also add a payload meant for the exit node. After the path is successfully built, the exit node will perform the RPC based on the flags it receives. It will then pack a protocol buffer with any information that the RPC returned. This buffer will be encrypted by the exit node as if it were a tunnel and sent down the path to the anonymous node. Upon reception of the packed buffer, the node will unpack it and use the results as if it had performed the RPC itself.

5.4 Problems

6. FUTURE CHALLENGES

There is one problem with the DHT that will prove challenging for the foreseeable future. It stems mostly from the fact that the DHT is structured and organized, but sometimes what we need is chaos. The place in the protocol where this is the largest problem is path building. Path building requires that only the anonymous node know which nodes are actually in the path. In order to build a path though, a node must communicate directly with the possible nodes in its path. Herein lies the problem, in order to get to an anonymous state, the protocol must first do some non-anonymous communication.

This problem could be forgotten, but if the assumptions about the strength of the attacker added the protocol has difficulty. By strengthen, we mean the attacker has control of 20% of all the nodes. In terms of sheer number of real nodes, a botnet could pose a real problem with some totaling 200,000 nodes. Since anonymous systems are still relatively small, a botnet could easily garner 20% of the total nodes. To combat this, there have been a few proposed DHTs (torsk [10], and NISAN [11]) with security as part of the design objectives. Unfortunately, the 20% compromised rule turns out be a very hard thing to overcome as shown by Wang et al. [15]. This paper finds flaws in every anonymous low latency look-up system to date.

There are also a number of other attacks which we can mention here briefly. One of the most widely applicable attacks is based traffic patterns and characteristics. These work by either creating or observing certain variables into the encrypted traffic that do not effect the correctness of the protocol. DoS attacks also are very viable. In fact, an attacker can easily use a DoS attack to create a favorable situation to unmask any anonymity [15]. Finally, the protocol will also need to contend with sybil attacks [4].

6.1 An example attack on Phantom

There is also an information leakage problem that all consistent hashing DHTs suffer from. The problem is again based on the need for global structure and organization. Each AP address is shared information, with a hash value that is the same at each node. Furthermore, since values are stored on which ever nodes are convenient, the attacker can quickly become the source of information. The attacker could use this to estimate how much a particular AP address is being used. This is known as an eclipse attack [14] because a particular node's routing table is eclipsed by malicious nodes. Phantom makes this attack a little more difficult because the node must have a communication certificate that hashes to the correct key. Still, a node would only need to have an ID that is closer to the particular key in question than the rest of the network.

7. CONCLUSION

The Phantom Protocol is well on its way to a dark, anonymous future. The thrust of this work is to identify, and fix areas that needed improvement. An important question to consider is can data be stored in a such a way that it retrieved quickly and anonymously. In the meanwhile, the protocol can provided at least a weaker form of anonymity. Solving this problem is certainly a worthy pursuit, given the world's hunger to unmask every person for everything from advertising to tracking. Phantom is one step technological towards getting the privacy back that technology has slowly eroded away.

8. REFERENCES

[1] S.L. Blond, P. Manils, C. Abdelberi, M.A.D. Kaafar, C. Castelluccia, A. Legout, and W. Dabbous. One bad apple spoils the bunch: exploiting p2p applications to trace and profile tor users. *Arxiv preprint arXiv:1103.1518*, 2011.

[2] Magnus Bråding. The phantom protocol. DEFCON 16, 2008.

[3] R. Dingledine and S.J. Murdoch. Performance improvements on tor or, why tor is slow and what we're going to do about it. 2009.

[4] J. Douceur. The sybil attack. *Peer-to-peer Systems*, pages 251–260, 2002.

[5] G.F. Du Pont. Criminalization of true anonymity in cyberspace, the. *Mich. Telecomm. & Tech. L. Rev.*, 7:191, 2000.

[6] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[7] K. Loesing, S. Murdoch, and R. Dingledine. A case study on measuring statistical data in the tor anonymity network. *Financial Cryptography and Data Security*, pages 203–215, 2010.

[8] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[9] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the tor network. In *Privacy Enhancing Technologies*, pages 63–76. Springer, 2008.

[10] J. McLachlan, A. Tran, N. Hopper, and Y. Kim. Scalable onion routing with torsk. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 590–599. ACM, 2009.

[11] A. Panchenko, S. Richter, and A. Rache. Nisan: Network information service for anonymization networks. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 141–150. ACM, 2009.

[12] Phobos. Five years as an exit node operator. `https://blog.torproject.org/blog/five-years-exit-node-operator`, 2008.

[13] J. Schlumberger and D.I.M. Gernoth. Implementing the phantom protocol. 2010.

[14] M. Steiner, T. En-Najjary, and E.W. Biersack. Exploiting kad: possible uses and misuses. *ACM SIGCOMM Computer Communication Review*, 37(5):65–70, 2007.

[15] Q. Wang, P. Mittal, and N. Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 308–318. ACM, 2010.