A Traceback Attack on Freenet

Guanyu Tian, Zhenhai Duan
Florida State University
{*tian, duan*}*@cs.fsu.edu*

Todd Baumeister, Yingfei Dong
University of Hawaii
{*baumeist, yingfei*}*@hawaii.edu*

***Abstract*—Freenet is a popular peer to peer anonymous network, with the objective to provide the anonymity of both content publishers and retrievers. Despite more than a decade of active development and deployment and the adoption of well-established cryptographic algorithms in Freenet, it remains unanswered how well the anonymity objective of the initial Freenet design has been met. In this paper we develop a traceback attack on Freenet, and show that the originating machine of a content request message in Freenet can be identified; that is, the anonymity of a content retriever can be broken, even if a single request message has been issued by the retriever. We present the design of the traceback attack, and perform Emulab-based experiments to confirm the feasibility and effectiveness of the attack. With randomly chosen content requesters (and random contents stored in the Freenet testbed), the experiments show that, for 24% to 43% of the content request messages, we can identify their originating machines. We also briefly discuss potential solutions to address the developed traceback attack. Despite being developed specifically on Freenet, the basic principles of the traceback attack and solutions have important security implications for similar anonymous content sharing systems.**

I. Introduction

Freenet is a distributed content sharing system, where users can both insert and retrieve files [7]. As a popular peer to peer anonymous network [2], Freenet aims to provide the anonymity of both content publishers and retrievers. (In this paper we use the two terms *file* and *content* interchangeably.) In Freenet, users contribute a portion of their hard disk space to form a global distributed storage sharing system. Global file operations such as insertion, retrieval, and deletion are all managed by the Freenet system itself. The location where a file is stored in Freenet is determined by a unique routing key associated with the file. Each node in Freenet only knows the information of their immediate neighbors. Mechanisms such as hop-by-hop forwarding of user messages, and rewriting the (source) address of the messages at each node, are employed in Freenet to support user anonymity.

Freenet supports two operational modes—Darknet and Opennet. In Darknet, only trusted friends can get connected to each other, where in Opennet, anyone can get connected on Freenet. In this paper we focus on the Opennet mode of Freenet, and we always mean the Opennet mode whenever we refer to Freenet. We note that the large-scale public Freenet on the Internet is operating in the Opennet mode in order for arbitrary users to join the Freenet. (Private) Freenet operating in the Darknet mode tends to be of smaller scale among a limited number of trusted friends. In addition, the stronger security provided by Darknet is not based on improved protocols or architectures, but rather on assumed stronger trust among members of a Darknet. Attacks on Opennet, such as the traceback attack developed in this paper, can be launched on a Darknet if a member of the Darknet decides to do so (or it is subverted and controlled by an attacker).

Freenet has undergone more than a decade of active development and deployment, and is widely used by privacy-conscious users for sharing files [2]. The high-level security mechanisms adopted by Freenet, such as hop-by-hop message forwarding and address rewriting, are time-proven means to support user anonymity; in addition, the cryptographic algorithms used in Freenet, such as hash algorithm, symmetric and asymmetric key algorithms, are all well-established. However, we note that various finer-grained design and development decisions of Freenet have not been thoroughly investigated, and it remains unanswered how well the anonymity objective of the original Freenet design has been met.

A number of watermarking-based traceback attacks (see, for example, [9], [10], [17]) have been developed on *low-latency* peer to peer anonymous networks such as Tor [4], [15], which aims to support anonymous communication services for interactive applications. In such low-latency anonymous networks, the message forwarding delay budget at each node is limited. Consequently, watermarking-based traceback attacks can be successfully carried out on such networks. In contrast to low-latency anonymous networks, anonymous content sharing systems such as Freenet do not have much constraint on the message forwarding delay budget. Any traffic patterns that may be embedded in messages of such networks can be easily destroyed. Existing watermarking-based traceback attacks on low-latency anonymous networks will not work well on anonymous content sharing systems such as Freenet.

In this paper we explore a few fine-grained design and development decisions made in Freenet and develop a traceback attack on Freenet. In particular, we show that the originating machine of a content request message can be identified. That is, the anonymity of a content retriever can be broken in Freenet, even if only a single request message has been issued from the corresponding machine. In developing this traceback attack, we exploit a few design and development features in the Freenet system, including methods to prevent routing loop of content request messages, the handling of various messages in Freenet, and mechanisms for a Freenet node to populate and update its routing table [1].

In the developed traceback attack, an attacker will deploy a number of monitoring nodes in Freenet to passively observe

content request messages passing through the nodes. Once an interested request message (based on routing key) is observed, the attacker will iteratively connect to the neighbors of a node n that has seen (either forwarded or initiated) the interested request message, and query these neighbors to determine other nodes that have seen the message. After all nodes that have seen the message have been identified, the originating machine of the message can be determined if the message forwarding path satisfies certain conditions.

In this paper we will present the details of the developed traceback attack on Freenet, and perform experimental studies to investigate the feasibility and effectiveness of the attack on the Emulab-based Freenet testbed [5]. The experiments are carried out using the source code of Freenet 0.7 (the current version of Freenet), extended to support the traceback attack. With randomly chosen nodes to initiate content requests to random files stored in the Emulab-based Freenet testbed, our experimental studies show that, for 24% to 43% of content request messages, we can identify their originating machines. For the rest of the content request messages that we cannot uniquely determine the originating machines, we are able to identify all the nodes that have either initiated or forwarded a content request message.

Furthermore, we briefly explore a few potential countermeasures to address the developed traceback attack, and provide a simple yet powerful insight into the design and development of peer to peer anonymous networks so that similar traceback attacks can be effectively mitigated. By attacking and providing proper security countermeasures on Freenet, we hope to enhance the anonymity strength of Freenet, and improve the user confidence in this anonymous content sharing system. We leave thorough investigation of potential solutions to the developed traceback attack as future work. We note that although the traceback attack and the solutions are developed specifically on Freenet, the basic principles of the traceback attack and the solutions have important security implications for the design and development of similar peer to peer, anonymous content sharing systems.

The remainder of the paper is organized as follows. In Section II we provide the necessary background on Freenet. We present the traceback attack on Freenet in Section III, and perform experimental studies in Section IV. We briefly discuss the potential solutions to the traceback attack and related work in Section V, and conclude the paper and discuss future work in Section VI.

II. Background on Freenet

In this section we provide a brief overview of the basic operations of Freenet that are most relevant to the current work. We refer interested readers to [7] and [3] for more details of Freenet.

Freenet is a peer to peer, anonymous content sharing system, with each node (a machine running Freenet) contributing a portion of its hard disk space. As a peer to peer system, nodes may join and depart from Freenet dynamically at any time. In Freenet, each node is associated with a location in the *circular* range [0, 1], where location 0 and location 1 are considered the same. The location of a node is randomly chosen by the node when it first joins Freenet. In order for arbitrary nodes to join Freenet, a set of seed nodes are provided, from which a new node can get connected to other nodes on Freenet.

When Freenet program starts on a node with location x (or when the node still needs more neighbors), an announcement message carrying the identification information of the node will be sent out and routed towards location x on Freenet. Intermediate nodes along the forwarding path of the message can add the requesting node as a neighbor if they still need more neighbors. In default, each Freenet node can have up to 40 neighbors. Given that the announcement message is routed towards the location x, it is likely that the majority of these intermediate nodes are close to location x. As a consequence, although nodes join Freenet in a distributed, asynchronous fashion, the topology of Freenet is semi-structured [12] in that, with a high probability, nodes with close-by locations are clustered together, and at the same time, a node may also connect to a neighbor with a far-away location.

Freenet data insertion and retrieval involve a number of different types of messages. In this paper we only focus on *Content Hash Key* (CHK) based content messages, in which the routing key is the SHA-256 hash of the corresponding data to be inserted or retrieved. The CHK routing key is used to uniquely identify the corresponding data on Freenet. To a degree, CHK messages are the most fundamental in Freenet. For routing purpose, the CHK routing key is converted into a location value in the same range of [0, 1], and the corresponding message will be routed towards that location when received by a node.

In the following we will first describe the handling of CHK content request message, which is issued by a node when the user requests a file on Freenet. (The handling of CHK data insertion messages is similar.) Each message is associated with a CHK routing key, a hop-to-live (HTL) value, and a unique identifier (UID). When a node receives a content request message, it will check if it has the corresponding data in its local data store. If it does, it will return the data along the reverse path of the request message, and nodes along the path may cache the data to better serve potential later requests on the same file. If the current node does not have the requested data, it will forward the request message to the next closest neighbor based on the routing key.

In order to improve the likelihood that a message is routed to a destination node storing the requested data, the routing decision is made based on the distance between the CHK routing key (after being converted to a value between 0 and 1), and the locations of the neighbors of the current node, *and* the locations of the neighbors of its neighbors. That is, Freenet uses two-hop routing lookup instead of one-hop lookup (only based on the locations of the immediate neighbors), which helps improve the routing efficiency and avoid local minimum in the Freenet topology. For this purpose, each node in Freenet has the location information of its immediate neighbors, and the neighbors of its immediate neighbors.

The HTL value in a content request message is used to determine the number of hops the message should be forwarded along a forwarding path. Each intermediate node will decrease the value, and when it reaches 0, the corresponding request message will be discarded instead of being forwarded. In addition, a *Data not Found* failure message will be sent back to the upstream node, which will be further propagated back to the content requester to indicate the failure of the content request. In Freenet, for security reasons, a node may not decrease the HTL value to 0 when it already reaches 1, with a configured probability. Otherwise, an attacker can precisely control how far a content request message can be forwarded. As a consequence of this random behavior, a content request message may be forwarded along a path longer than the specified HTL value in the message.

When a request message cannot be forwarded due to reasons other than HTL = 0 (for example, no additional neighbors are available), the request will be backtracked to the upstream node where it comes from, in the sense that the upstream node will forward the request onto the next closest neighbor (if it is available). This process continues until either HTL becomes 0, the requested data is found, or all possible routes have been tried but the data cannot be found. The default maximum initial value of HTL is 18.

The UID in a content request message is used by nodes to uniquely identify a message, and to prevent routing loops. UIDs are randomly generated and are of length of 8 bytes, it is unlikely that two unrelated messages will have the same UID value in Freenet. When a node receives a request message, it will check if it has seen this UID before. If it does, a *reject with loop* message will be sent back to the upstream neighbor where the message comes from. Each node maintains a list of UIDs that it has seen but has not finished processing the associated request message (the corresponding reply has not been received). It will also maintain a queue of the UIDs that the node has finished processing (the corresponding reply has come back), which can hold up to $10,000$ UIDs. The oldest UID will be deleted from the queue when the UID of a newly completely message needs to be inserted into the queue and the queue is already full. Completed request messages of different types share the same UID queue. As a consequence, a *reject with loop* message will be sent back as long as the current node has seen the corresponding UID in the incoming request message, regardless of the type of the request message.

Another relevant message type is probe message, which is mainly used for administrative and debugging purposes. For example, when a node receives a probe message, it will send back its routing table information. Similarly, each probe message is associated with a UID. In addition, it also carries a destination location, to which the probe message should be forwarded. Valid destination location value should be in the same range of [0, 1]. When a probe message with an invalid location value (outside the range [0, 1]) arrives at a node, the message will be discarded instead of being forwarded.

III. Traceback Attack on Freenet

In this section we will present the design of the traceback attack on Freenet. The traceback attack has two important components—connecting an attack node to a suspect node in Freenet, and querying a neighbor to determine if it has seen a content request message with a particular UID value. In the following we will first describe the two important components of the traceback attack, and then we will describe the traceback process to identify all the nodes that have seen (either initiated or forwarded) a content request message, and the difficulties and opportunities in identifying the originating machine of a content request message. Towards the end of this section, we present techniques to identify the originating machine of a content request message when the forwarding path satisfies certain conditions.

A. Connecting to a Freenet Node

As one of the important steps in the traceback attack, an attacker needs to connect an attack node a to a suspect Freenet node n so that a and n become neighbors of each other. The assumption is that the attacker knows the location of the suspect node n. We have developed an effective method to carry out this task and the details are reported in [1]. In the following we provide a brief overview of the method.

In Freenet, each node can have a pre-specified maximum number of neighbors (40 in default). When an announcement message arrives at a node and the node does not have enough neighbors yet, the requesting node will be automatically accepted as a new neighbor. Otherwise (the node already has the maximum number of neighbors), the node will check a key condition to determine if an existing neighbor can be replaced by the requesting node. Neighbors at a node are classified into a few pre-defined categories, depending on how they get connected to the node. For example, one category is the set of neighbors that are connected to the node via announcement messages.

The key condition to determine if the node should perform a neighbor replacement operation is if any of the neighbor categories has successfully served at least a pre-configured minimum number of content requests. The exact intention of this condition is not explicitly stated in the Freenet design (or its source code), we can only speculate that this condition is used to make sure that the node has accumulated enough knowledge of the neighbors in their capacity in serving content requests. When this condition is satisfied, the least recently used (LRU) neighbor of the node will be replaced by the requesting (attack) node, regardless of the category of the neighbor. We note that this condition can be easily satisfied at "busy" nodes, which forwards a large number of requests and replies. The default minimum number of requests to be successfully served for performing neighbor replacement is 10.

If the condition is not currently satisfied at the suspect node n, we can repeatedly perform file insertion and retrieval operations to enforce this condition. Given that we know the location of the suspect node n, we can insert files with routing keys surrounding its location. Given that the routing key of

a file is the SHA-256 hash of the file, a large number of files can be pre-composed so that the location range [0, 1] can be reasonably covered by their routing keys. Note that, due to the nature of hash functions, we do not need to have sophisticated file structure and content in order to have a reasonable coverage of the complete location range. (As a matter of fact, all files used for this purpose in the experimental studies in Section IV are of one line text string, and we only slightly change the text string in different files in order to obtain a totally different routing key.)

In order to enforce the neighbor replacement condition at the suspect node n, we choose the files with routing keys that are close to the suspect node and insert them into Freenet. By this file insertion operation, we know a number of files that are located close to the suspect node. We then request the inserted files on a different attack node. After we have successfully retrieved the files for a number of times exceeding the minimum threshold, we will announce a node into the Freenet with a location that is close to the suspect node. If the new node becomes the neighbor of the suspect node, we are done. Otherwise, we will repeat this process until the new node becomes the neighbor of the suspect node.

B. Querying a Neighbor

Another important component of the developed traceback attack is to determine if a neighbor has seen a message with a particular UID. Recall that each Freenet node maintains a list of UIDs associated with request messages that the node has not finished processing (the corresponding reply has not come back), and a queue of UIDs associated with request messages that the node has finished processing (the corresponding reply has come back). For simplicity, we refer to both as the set of UIDs maintained by the node. In order to determine if a neighbor has seen a content request message with a UID value, we can send a request message with the same UID value.

A key requirement of sending this request message to a neighbor is that, the message should not be forwarded any further by the neighbor. Should this occur, this (forged) request message may pollute the Freenet in terms of the nodes that have seen the UID value. More specifically, let N denote the set of nodes that have initiated or forwarded an interested content request message with a particular UID value. If the forged request message is forwarded beyond the intended neighbor, nodes that have not previously seen the interested content request message will now see the corresponding UID value. Our traceback attack algorithm may falsely identify this node as a member of N, and the result of the traceback attack could be wrong.

Our first try was to send a content request message to a neighbor with the desired UID value, but with the initial value of HTL set to 1. However, it turns out that this cannot prevent the content request message from being further forwarded by the neighbor. As we have discussed in Section II, with a configured probability, the value of HTL will not be decreased when it already reaches 1, and the corresponding content request message will be further forwarded (a message is only discarded when HTL reaches 0 or it cannot be forwarded due to routing issues). Due to this issue, instead of sending a forged content request message, we will send a probe message with the desired UID value to a neighbor.

The trick to prevent this probe message from being further forwarded by the neighbor is to select an invalid destination location value outside the range [0, 1]. Recall that, different request messages, including both content request messages and probe messages, share the same data structures maintained by a node to record recently observed UIDs. Moreover, a probe message carrying an invalid destination location value will be discarded by the receiving node. Combining these two features, we know that, when a neighbor receives a probe message constructed in this way, it will return a *reject with loop* message if the neighbor has seen a message with the UID value previously. And more importantly, regardless if the neighbor has seen the UID value before, it will not further forward this probe message, so that no other nodes on the Freenet will be polluted by this forged probe message.

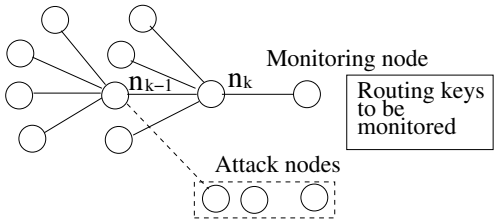

Fig. 1. Illustration of the traceback attack.

C. Identifying All Nodes Seeing A Content Request Message

In this subsection we present the details of the traceback process to identify all nodes that have seen a content request message on Freenet (see Figure 1). After all such nodes have been identified, in the next subsection we present techniques to identify the originating machine of a message. An attacker will deploy a number of monitoring nodes in Freenet, with each maintaining a set of interested routing keys to be monitored (the routing keys are calculated based on the files to be monitored). A monitoring node will passively observe the content request messages passing through the node and try to match their routing keys with the routing keys to be monitored.

When an interested content request message is identified, a few pieces of information will be forwarded to an attack node, including the content request message itself and the set of neighboring nodes to determine which of them (if any) has seen the corresponding UID value. We note that we already know the upstream node n_k from which the request message comes from at the monitoring node. In this case, the neighbors of n_k will be sent to an attack node (instead of neighbors of the monitoring node). Note that we have the neighbor information of n_k at the monitoring node, due to the two-hop routing scheme of Freenet. To ease exposition, we will refer to the set of neighbor nodes forwarded to the attack node (along with the content request message) as the suspect nodes. Note that the set of suspect nodes will not include the downstream

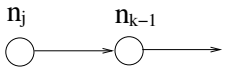

Fig. 2. Case 1: n_j forwarding request to n_{k-1}.

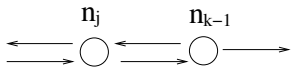

Fig. 3. Case 2: n_{k-1} forwarding request to n_j, but backtracked from n_j.

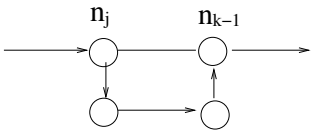

Fig. 4. Case 3: No message forwarding between n_j and n_{k-1}.

node along the forwarding path of the request message, which we know has seen the corresponding UID before.

When an attack node a receives the information, it will try to determine one by one if any of the suspect nodes has seen the corresponding UID value (i.e., the content request message) by utilizing the two components that we have discussed above. In particular, for each suspect node n, the attack node a will first connect to the node (Section III-A), and then it will send a probe message with the corresponding UID value to the node to determine if the suspect node has seen the UID before (Section III-B). Conceptually, we can consider that the attack node maintains a queue of the suspect nodes, and each time it removes one suspect node from the queue to determine if the suspect node has seen a particular UID value.

If a suspect node n_{k-1} has seen the UID value before, the neighbors of n_{k-1} will be added into the queue, and the traceback process continues (by removing the next suspect node from the queue). Note that, given that the attack node is a neighbor of n_{k-1}, we have the neighbor information of n_{k-1}, due to the two-hop routing scheme of Freenet. When the queue becomes empty, the complete traceback process to identify all nodes that have seen the corresponding content request message is finished.

D. Difficulties in Identifying Originating Machine

In order to understand the difficulties in tracing back a general content request message to its originating machine, we consider two different traceback situations. In the first case, at each step of the traceback process, there is only one suspect node that has seen the concerned UID value before; while in the second case, multiple suspect nodes have seen the UID value before. We refer to the traceback paths in the first case as *linear reverse paths*, and in the second case as *non-linear reverse paths*. Note that a reverse path associated with a content request message is concerned with the traceback path starting at a monitoring node (back towards the origin), which is different from the *forwarding path* that the message takes from the origin towards the destination (up to the monitoring node for traceback purpose).

In Freenet, a message forwarding path will be linear if the message does not backtrack along the forwarding path (see Section II). However, as we will show later, a linear forwarding path does not always result in a linear reverse path, which makes it important for us to make the distinction between forwarding paths and reverse paths. In the following we will illustrate the difficulties in identifying the originating machine of a message along a non-linear reverse path. More specifically, during the traceback process of a message, when we trace back from a node n_{k-1} to determine if any of the corresponding suspect nodes (i.e., neighbors of n_{k-1}, but excluding downstream neighbor node n_k, from which we trace back to node n_{k-1}, see Figure 1), more than one suspect nodes have seen the interested UID value.

We note that when we query if a suspect node has seen a UID value, we cannot determine the time when the corresponding content request message is received, and also we cannot determine the direction of the message forwarding. What we can obtain is only the fact if the node has seen the UID value. This makes it hard to determine which of the suspect nodes is the upstream node of n_{k-1} along the forwarding path of the request message, when multiple of them have seen the corresponding UID value.

To illustrate the difficulties in determining the upstream node of n_{k-1} in this case, we show three possible forwarding situations in Figures 2, 3, and 4. We note that they are not all the possible cases, but rather a few representative examples. A neighbor n_j of node n_{k-1} may have seen a particular UID value because it forwards the corresponding request message to n_{k-1} (see Figure 2). However, as shown in Figure 3, it is also possible that node n_{k-1} forwards the request to n_j, but then the message is backtracked from n_j to n_{k-1}, because the message cannot be further forwarded. (It is possible that the message has been further forwarded by node n_j to other nodes, before the backtrack from n_j to n_{k-1} occurs.)

Moreover, as shown in Figure 4, it is also possible that the two neighbors n_j and n_{k-1} have no direct interaction regarding the forwarding of the message, although both of them have seen the UID value. In this case, node n_j did not directly forward the message to n_{k-1}, and node n_{k-1} also did not directly forward the message to node n_j. As shown in the figure, node n_j receives the message with the corresponding UID value but forwards the message to a different node (rather than node n_{k-1}), and similarly, node n_{k-1} receives the message from a different node (instead of node n_j). When this occurs, we note that the traceback process will observe a non-linear reverse path, even if the forwarding path is linear.

Without the information of message forwarding time or direction, in general it is hard for us to distinguish different cases, and uniquely identify the upstream node at a node n_{k-1}, when multiple suspect nodes have seen a particular UID value. Consequently, we will aim to identify the originating machine of a content request message only if the reverse path is linear. However, we note that, identifying the originating machine of a request message associated with a linear reverse path is not trivial, and we cannot always successfully determine the originating machine of a message in this case. The key

challenge is that, when we trace back along a linear reverse path from n_{k-1} to a single suspect node n_j, we still need to determine which of the two cases presented in Figures 2 and 3 is true, that is, to determine if a backtrack has occurred. Note that the case presented in Figure 4 will not occur on a linear reverse path; otherwise, the reverse path will not be linear.

In the next subsection we will present a few techniques to identify the conditions under which we can uniquely determine the originating machine of a message associated with a linear reverse path, by exploiting the routing policy of Freenet. We point out that such conditions can be applied to the traceback of certain messages associated with non-linear reverse paths. However, in this work we will only apply them to messages associated with linear reverse paths for two reasons. First, although they can be applied to messages associated with non-linear reverse paths, the complexity of determining the originating machines of such messages will be much higher than those associated with a linear reverse path. Moreover, the rate to successfully determine the originating machine of such messages can be potentially lower than those associated with a linear reverse path. Second, the experimental studies in Section IV based on realistic Freenet testbeds show that it is not uncommon for a message to be associated with a linear reverse path on Freenet. This phenomena is likely caused by the interplay of a number of factors in Freenet, including the semi-structured network topology of Freenet, strong connectivity among Freenet nodes and two-hop routing lookup, and a reasonably large HTL value.

E. Identifying Originating Machine

In this subsection we present techniques to identify the conditions under which we can uniquely determine the originating machine of a content request message associated with a linear reverse path (unless otherwise specified, all messages considered in this subsection are associated with a linear reverse path).

Recall that, as we have discussed in Section II, a Freenet node n will choose the next closest neighbor to forward a message to based on the distance between the routing key and the location of neighbors (and their neighbors). Consequently, it is possible for us to determine the forwarding direction of a message by exploiting the routing policy of Freenet. We first define a few notations. Consider a forwarding path of a message, we let $n \rightarrow h$ denote that the message is forwarded from node n to node h, and $n \leftrightarrows h$ denote that the message is forwarded from node h to node n, and then backtracked from n to h (it is possible that the message has been further forwarded by n to other nodes, before being backtracked to node h).

Similarly, consider a reverse path associated with a message, we let $n \twoheadleftarrow h$ denote that we trace back the message from node h to node n. In addition, we let $n_0 \twoheadleftarrow n_1 \twoheadleftarrow \ldots \twoheadleftarrow n_{k-1} \twoheadleftarrow n_k \ldots \twoheadleftarrow n_m$ denote the complete reverse path, where n_m is the attacker's monitoring node, and n_0 is the last node along the reverse path, of which no suspect nodes have seen the concerned UID value. In addition, we let $d(n)$ denote the distance from the node n to the destination implied by the routing key of the message. For the convenience of discussion, we define the length of a (linear) path as the number of *nodes* on the path. In the following we establish the conditions under which we can uniquely identify the originating machine of a message, through a series of lemmas (the proofs of the lemmas are given in [14]).

First, we consider a trivial case where the length of the reverse path of a message is two, i.e., $n_0 \twoheadleftarrow n_m$. In this case, it is easy for us to see that n_0 is the originating machine of the message. We state this fact in the following lemma.

Lemma 1 (C1: Path with length of two). *Given a linear reverse path $n_0 \twoheadleftarrow n_m$, which is of length of two, n_0 is the originating machine of the message.*

In the following we focus on linear reverse paths that are longer than two.

Lemma 2. *Given a linear reverse path $n_0 \twoheadleftarrow \ldots \twoheadleftarrow n_{k-1} \twoheadleftarrow n_k \twoheadleftarrow n_{k+1} \ldots \twoheadleftarrow n_m$, backtrack can be started at most one time during the forwarding of the message, along the corresponding forwarding path.*

Lemma 3 (Neighbor preference condition at a node). *Given a linear reverse path $n_0 \twoheadleftarrow \ldots \twoheadleftarrow n_{k-1} \twoheadleftarrow n_k \twoheadleftarrow n_{k+1} \ldots \twoheadleftarrow n_m$, consider an arbitrary node n_k between n_1 and n_{m-1} (inclusive), node n_k cannot have initiated a traceback instance to node n_{k-1}, if the condition $d(n_{k+1}) < d(n_{k-1})$ holds.*

Lemma 4 (C2: Neigbhor preference along path). *Given a linear reverse path $n_0 \twoheadleftarrow \ldots \twoheadleftarrow n_{k-1} \twoheadleftarrow n_k \twoheadleftarrow n_{k+1} \ldots \twoheadleftarrow n_m$, n_0 is the originating machine of the message if, for every node n_k between n_1 and n_{m-1} (inclusive), the condition $d(n_{k+1}) < d(n_{k-1})$ holds.*

We comment that C2 (Lemma 4) does not require $d(n)$ to be monotonically decreasing along the forwarding path of the message. It only requires the condition to be held between each pair of every other nodes along the forwarding path. As an example, consider a simple reverse path $0.76 \twoheadleftarrow 0.87 \twoheadleftarrow 0.10 \twoheadleftarrow 0.05$ (for destination 0.06). For simplicity, let us only use one-hop routing (instead of two-hop routing), that is, the distance from a node to a destination is only based on the location of the node (instead of locations of the node and its neighbors). It is easy to verify that the distance to the destination does not monotonically decrease along the corresponding forwarding path. However, it satisfies C1, and we can determine that 0.76 is the originating machine of the corresponding message.

Lemma 5 (C3: Neigbhor preference at n_1). *Given a linear reverse path $n_0 \twoheadleftarrow n_1 \ldots \twoheadleftarrow n_{k-1} \twoheadleftarrow n_k \twoheadleftarrow n_{k+1} \ldots \twoheadleftarrow n_m$, n_0 is the originating machine of the message if there exists at least one neighbor n of n_1 that has not seen the UID value, but the condition $d(n) < d(n_0)$ holds.*

We will use C1, C2, and C3 (Lemmas 1, 4, and 5) to identify the originating machine of a content request message. Given a linear reverse path identified by the traceback process

presented in the last subsection, we check if either C1, C2, or C3 is satisfied.

IV. Performance Evaluation

In this section we perform Emulab-based experimental studies to evaluate the feasibility and effectiveness of the developed traceback attack on a realistic Freenet testbed. In the following we will first describe the setup of the experimental studies, and then we will present the results of the studies.

A. Experimental Setup

We carry out the experimental studies using the Emulab testbed [5], and Freenet 0.7. We extend the source code of Freenet 0.7 to add the functionalities to support the traceback attack. A number of bash scripts have also been written to largely automate the traceback attack.

The Freenet networks we used in the experimental studies consist of 70 nodes. 4 out of the 70 nodes are seed nodes, through which other nodes can get connected to the Freenet testbed. The set of 70 nodes in each Freenet network does not include the attack nodes (see Figure 1), which are not connected to the network before an attack starts. We use a set of 5 additional nodes as attack nodes (theoretically, one attack node is sufficient to carry out the attack). We perform 3 sets of experiments, each consisting of 100 experimental runs. Each set of experiments use an identical Freenet topology, which is randomly constructed as follows. When we first start a set of experiments, each node will randomly select a location and contact a seed node to join the Freenet. The locations of the seed nodes are also randomly selected by the individual seed nodes, and seed nodes are started before other general nodes are started. After all nodes have joined the Freenet testbed (and therefore the network topology of the Freenet is formed), we then run 100 experimental studies on the Freenet testbed (or simply the Freenet).

The 100 experimental studies in each set are grouped into 10 clusters, with each consisting of 10 experiments. For each group of 10 experiments, we insert a random file into the Freenet, we then randomly choose a node to retrieve the file to complete one experimental run. To make an experiment meaningful, we do ensure that the randomly selected originating machine does not already have the file. After each experimental run, we restore the Freenet to the original state before the file is requested (i.e., we remove the file from all the caches due to this file request, and the data store of the file requester), before we perform the next experimental run, so that it will not be affected by previous experimental runs. After a group of 10 experimental runs, a different file is inserted into the Freenet, and the experiments are repeated.

After a set of 100 experimental runs, we will re-start the Emulab-based Freenet testbed, so that a different Freenet topology will be constructed for the next set of 100 experiments. (Different randomly selected node locations will cause different Freenet topologies.) We repeat this process three times (for the three sets of 100 experimental runs).

In our experiments, each node can have at most 4 neighbors. Without loss of generality, we use the nodes storing a requested file as the monitoring nodes. We note that this maximizes the length of the path that we need to trace back. In addition, a traceback attack is initiated after the monitoring node has sent back the requested file. That is, we rely on the queue of UIDs that a node has finished processing the corresponding request message to determine if it has seen a UID value before. We believe that, in real traceback attacks on the public Freenet, requested files should also be returned to help minimize the suspicion of a file requester that it is being traced back, as the requested file comes back as expected. We refer to the three sets of experimental runs as $S1$, $S2$, and $S3$, respectively.

B. Results

In this subsection we present the results of the Emulab-based experimental studies (additional results are reported in [14]). First we investigate how well we can determine the originating machine of a content request message. Table I shows the results. For three sets of experiments, we can successfully determine the originating machine of 43%, 24%, and 41% of request messages, respectively.

TABLE I
Results of experimental studies.

Set	Total	Successful	
		Number	Percentage
S1	100	43	43%
S2	100	24	24%
S3	100	41	41%

We make two observations from the results. First, the successful rate to determine the originating machine of a content request message is reasonably high (ranging from 24% to 43%). As we have discussed in Section III, this is likely caused by a number of factors of Freenet, including semi-structured network topology, strong connectivity among nodes and two-hop routing lookup, and reasonably large HTL value. Second, the successful rate to determine the originating machine of a content request message varies greatly from 24% to 43%. The specifics of the Freenet network topology, the location of the file to be requested, and the location of the node to initiate a file request will all likely affect the forwarding path of the content request message, and consequently, the chance for the originating machine of the request message to be determined.

Despite the variation in the successful rate, we emphasize that, as shown in Table I, the probability to determine the originating machine of a content request message is reasonably high in the performed experiments. In addition, as we have discussed in Section III, for the rest of the content request messages that we cannot determine the originating machine, we can identify all the machines that have either initiated or forwarded the message, which could be helpful forensic information in some investigative cases.

TABLE II
CLASSIFICATION OF MESSAGES SUCCESSFULLY TRACED BACK.

Set	Total successful	C1	C2	C3	C2 & C3
S1	43	17	19	24	17
S2	24	11	4	13	4
S3	41	25	12	12	8

In order to better understand the results of the experimental studies, in Table II we show the number of messages whose originating machine are successfully identified by rules C1, C2, and C3, respectively. In the table we also show the number of messages that are successfully traced back by both rules $C2$ and $C3$. From the figure we can see that, although the originating machines of a large number of messages are determined because of the path length of two (i.e., $C1$), $C2$ and $C3$ are indeed effective in identifying originating machines of messages that traverse a long path before encountering a monitoring node. In particular, it shows that the rule $C3$ alone is already very effective in helping determining the originating machine of a message.

V. DISCUSSION AND RELATED WORK

In this section we will first discuss potential solutions to the developed traceback attack, and then we will briefly present related work on attacks on peer to peer anonymous networks.

A. Potential Solutions

The design of the traceback attack in Section III depends on a number of design and development decisions made in Freenet, including, for example, the two-hop routing lookup and neighbor replacement policy. It is clear that changing any of these design and development decisions will likely make *the* developed traceback attack on Freenet less effective, or eliminate the attack altogether. However, identifying the optimal solution to address the problem is not simple. For example, changing either the two-hop routing lookup or the neighbor replacement policy will make it much harder to carry out the designed traceback attack. On the other hand, they will also likely affect the performance and adaptivity of Freenet.

Given that Freenet is only a semi-structured (instead of completely structured) peer to peer network, removing the two-hop routing lookup will likely greatly affect the performance of Freenet in locating requested files. Similarly, disallowing a Freenet node to replace an existing neighbor with a new node will prevent Freenet from re-organizing its topological structure, based on, for example, performance of the neighbors, and this will also make it hard for a new node to join Freenet. Another potential solution is to dynamically change the UID value of a request message along the forwarding path. However, this will likely affect the effectiveness of forwarding loop prevention in Freenet.

In addition, although such solutions may help address *the* developed traceback attack, they may not work for other potential traceback attacks. We note that, although the traceback attack presented in this paper only targets content request messages on Freenet, it can be potentially extended to trace back other types of messages on Freenet. Moreover, it may be generalized to attack other anonymous file storage and retrieval systems that are built using the similar techniques as Freenet. We need to develop a generic solution that can help address the fundamentals of this type of traceback attacks, instead of individual attacks.

We note that the key capability that the traceback attack relies on is being able to query a Freenet node to determine its state regarding the forwarding of a message (or rather the corresponding UID value). That is, by certain means, an attacker can distinguish a node that has forwarded a message from a node that has not forwarded a message. If this distinction can be somehow identified, an attacker can then determine the forwarding of a message, and a traceback attack can be potentially carried out. We therefore believe that, in order to prevent this and other traceback attacks on Freenet (and similar peer to peer anonymous networks), special attention should be paid to the design and development of the system so that, anyone external to a node should not be able to determine the message forwarding state at the node.

Following this simple yet powerful observation, a more proper solution to the traceback attack is to change the response of a Freenet node to an incoming request (or probe) message with a UID that has been observed by the node. Instead of sending back a *Reject with Loop* failure message to inform the requester this fact, a more general failure message should be sent back so that the requester cannot determine the message forwarding state of the node. Specific failure messages currently used in Freenet, such as *Reject with Loop*, *Route not Found*, and *Data not Found* should all be replaced by this general failure message. After receiving such a message, the requester knows that it cannot use the node to locate the requested file, but it cannot infer if the node has seen the corresponding request message before.

We caution that, given the complexity of the Freenet system, before such a solution can be adopted on Freenet, we need to thoroughly evaluate the potential impacts of this solution on the behavior and performance of Freenet. In particular, such a solution should not (greatly) affect any optimizations that the Freenet system may perform to re-organize its network structure to speed up the locating and forwarding of contents stored in the network.

Accidentally, the above insight in developing generic solutions to address traceback attacks on Freenet is similar to the behavioral equivalence, or the *weak bi-similarity* property of systems modeled using the π-*calculus* [11]. In essence, this property states that, if two processes are weakly bi-similar, we cannot distinguish the two processes based on the observations of the input to and output from the two processes. In the context of Freenet traceback attacks we have discussed, if two nodes are weakly bi-similar, an attacker cannot tell which of the two nodes (if any) have initiated or forwarded a message previously, after sending query messages to both nodes.

B. Related Work

In this subsection we briefly discuss the related work. A number of theoretical attack models on p2p anonymous networks have been developed, including the predecessor attack [16] and the eclipse attack [13]. They do not target any particular p2p anonymous networks; rather, they investigate the relationship between the number and coverage of attacking nodes and the likelihood that any message can be traced back. They were developed based on the fact that the members of p2p anonymous networks are dynamic. If a sufficient portion of a p2p anonymous network consists of attack nodes, it is likely that they can collaborate to identify the possible origin of a message.

Such studies provide us with some guideline on the deployment of attacking nodes on a p2p anonymous network. However, in this work we are more interested in attacks that exploit the operational features of a p2p anonymous network instead of attacks that rely on a large number of attack nodes to cover the critical regions of a p2p anonymous network. In addition, real-world p2p anonymous networks including Freenet have taken steps to prevent a large number of nodes in any individual network domain from joining the networks, so that it is becoming hard to launch such attacks from a single network domain. The traceback attack developed in this paper requires much less resources on the attacker compared to this type of attacks.

A number of watermarking techniques have been developed to trace back traffic on low-latency p2p anonymous networks such as Tor (see, for example, [9], [10], [17]). However, as we have discussed in Section I, existing watermarking-based traceback attacks on low-latency anonymous networks will not work well on anonymous content sharing systems such as Freenet.

An attack called *Pitch Black* has been developed on Freenet [6]. Pitch Black works in the Darknet mode of Freenet, which allows the swapping of locations between neighboring nodes, based on the distance to their respective neighbors. Pitch Black exploits the location swapping feature of Darknet to deteriorate the performance of Freenet in terms of network congestion and data loss. It does not target attacking or improving the user anonymity of Freenet. In addition, it only works in Darknet mode of Freenet.

Freenet project has also documented a few potential attacks on Opennet [8], including node harvesting on Freenet, mobile attacker source tracing, and routing table takeover. The attack developed in this paper is a complete, practical, and efficient attack on Freenet to trace back a content request message to its originating machine. Importantly, this work helps to illustrate the security issues in the design and development of Freenet, and to provide insights on how to improve Freenet and similar p2p anonymous networks to proactively defend against similar traceback attacks.

VI. Conclusion and Future Work

In this paper we have developed a practical and efficient traceback attack on Freenet by exploring a few design and development decisions made in Freenet. In addition, we have also performed Emulab-based experimental studies to confirm the feasibility and effectiveness of the developed traceback attack. As future work, we will explore opportunities to further investigate the performance of the developed traceback attack on larger-scale Freenet testbeds. The scale of our current experiments was constrained by the resources we can obtain from the Emulab system. We will extend the attack to trace back other types of request messages on Freenet, and explore the possibility to generalize the attack onto other peer to peer anonymous content sharing systems. In addition, we will fully investigate potential solutions to address the developed traceback attack.

VII. Acknowledgments

Guanyu Tian and Zhenhai Duan were supported in part by NSF Grant CNS-1041677. Todd Baumeister and Yingfei Dong were supported in part by NSF Grants CNS-1041739, CNS-1127875, and CNS-1219579. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of National Science Foundation.

References

[1] T. Baumeister, Y. Dong, Z. Duan, and G. Tian. A routing table insertion attack on Freenet. In *Proceedings of ASE International Conference on Cyber Security*, Washington D.C., USA, Dec. 2012.

[2] C. Callanan, H. Dries-Ziekenheiner, A. Escudero-Pascual, and R. Guerra. Leaping over the firewall: A review of censorship circumvention tools. Report by Freedom House, Apr. 2011.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop On Designing Privacy Enhancing Technologies: Design Issues In Anonymity And Unobservability*, 2001.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13 th Usenix Security Symposium*, 2004.

[5] Emulab. Network emulation testbed. http://www.emulab.net/.

[6] N. S. Evans, C. GauthierDickey, and C. Grothoff. Routing in the dark: Pitch black. In *Proceedings of ACSAC*, 2007.

[7] Freenet. https://freenetproject.org/.

[8] Freenet. Opennet attacks. https://wiki.freenetproject.org/Opennet_attacks/.

[9] J. Huang, X. Pan, X. Fu, and J. Wang. Long PN code based DSSS watermarking. In *Proc. IEEE INFOCOM*, 2011.

[10] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia. A new cell counter based attack against Tor. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2009.

[11] R. Milner. *Communicating and mobile systems - the π-calculus*. Cambridge University Press, 1999.

[12] O. Sandberg. Distributed routing in small-world networks. In *Proceedings of 8th Workshop on Algorithm Engineering and Experiments*, Jan. 2006.

[13] A. Singh, T. wan Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. IEEE INFOCOM*, 2006.

[14] G. Tian, Z. Duan, T. Baumeister, and Y. Dong. A traceback attack on freenet. Technical Report TR-130114, Department of Computer Science, Florida State University, Jan. 2013.

[15] Tor. https://www.torproject.org/.

[16] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security*, 7, 2004.

[17] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao. DSSS based flow marking technique for invisible traceback. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2007.