Forensic Investigation of the OneSwarm Anonymous Filesharing System

Swagatika Prusty Brian Neil Levine Marc Liberatore
Dept. of Computer Science, Univ. of Massachusetts Amherst
{swag, brian, liberato}@cs.umass.edu

ABSTRACT

OneSwarm is a system for anonymous p2p file sharing in use by thousands of peers. It aims to provide Onion Routing-like privacy and BitTorrent-like performance. We demonstrate several flaws in OneSwarm's design and implementation through three different attacks available to forensic investigators. First, we prove that the current design is vulnerable to a novel timing attack *that allows just two attackers attached to the same target to determine if it is the source of queried content. When attackers comprise 15% of OneSwarm peers, we expect over 90% of remaining peers will be attached to two attackers and therefore vulnerable. Thwarting the attack increases OneSwarm query response times, making them longer than the equivalent in Onion Routing. Second, we show that OneSwarm's vulnerability to traffic analysis by colluding attackers is much greater than was previously reported, and is much worse than Onion Routing. We show for this second attack that when investigators comprise 25% of peers, over 40% of the network can be investigated with 80% precision to find the sources of content. Our examination of the OneSwarm source code found differences with the technical paper that significantly reduce security. For the implementation in use by thousands of people, attackers that comprise 25% of the network can successfully use this second attack against 98% of remaining peers with 95% precision. Finally, we show that a novel application of a known TCP-based attack allows a single attacker to identify whether a neighbor is the source of data or a proxy for it. Users that turn off the default rate-limit setting are exposed. Each attack can be repeated as investigators leave and rejoin the network. All of our attacks are successful in a forensics context: Law enforcement can use them legally ahead of a warrant. Furthermore, private investigators, who have fewer restrictions on their behavior, can use them more easily in pursuit of evidence for such civil suits as copyright infringement.*

Categories and Subject Descriptors

C.2.4 [**Comp. Comm. Networks**]: Distributed Systems

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

General Terms

Security, Legal Aspects, Design

Keywords

Digital Forensics, Child Sexual Exploitation, P2P Networks

1. INTRODUCTION

OneSwarm [8, 9] is a peer-to-peer (p2p) system for anonymous file sharing that is actively used by thousands of peers and has been downloaded by hundreds of thousands of users. Like Gnutella, peers can query for content and download it from other users. The main design goal of OneSwarm is to resist monitoring and traffic analysis attacks that are easily performed on p2p file sharing networks such as Gnutella and BitTorrent [11]. In short, OneSwarm aims to provide Onion Routing-like privacy and BitTorrent-like performance. OneSwarm's architecture is based on a dense topology of peers to ensure availability of content, probabilistic forwarding of queries to neighbors to thwart traffic analysis, and application-level delays to thwart timing attacks.

OneSwarm provides privacy using a much different method than Onion Routing (OR), which is the architectural basis for Tor [6]. If a peer possesses queried content, it will not forward the query, and it will answer the query only after some delay; otherwise it forwards the query to each of its neighbors with probability p, again after some delay. Thus, as the original paper points out [8], colluding neighbors can determine the chance that a target possesses content when the query is not forwarded to them. For example, that analysis showed that with $p = 0.5$ and a population of 1,000 users, the chances are only 1% that 30 colluding attackers will succeed with 95% precision in determining that a specific, targeted peer is the source of a queried file. Users have downloaded and joined the system with these expectations.

The network is sure to receive the attention of many civil and government investigators. Our law enforcement partners were able to download images of the sexual abuse of prepubescent children through the OneSwarm system; our past work with law enforcement has demonstrated the same for Gnutella [11] and BitTorrent. We take the perspective of criminal forensic investigators that seek to meet standards of evidence for lawfully obtaining warrants for seizing and searching computers observed to be sharing child pornography. We note that, like all p2p networks, the vast majority of users are sharing copyrighted materials rather than child pornography, but as we explain below, investigations of copyright are easier than the criminal standard we hold to.

In this paper, we demonstrate several flaws in OneSwarm's design and implementation based on three different attacks that are available to investigators. First, we prove that OneSwarm is vulnerable to a novel timing attack that allows just two attackers attached to the same target to distinguish whether it is the source of queried content. When attackers comprise 15% of OneSwarm peers, we expect over 90% of remaining peers will be attached to two attackers and therefore vulnerable. The attack itself is simple: two attackers simultaneously query for the same content and they compare the *summed* response time. If the summed application delay is no greater than 600ms above the summed network roundtrip time to the target, then the neighbor target is the source. We derive the proper application-level delays that can deter this attack, but also show that the increased delays result in significantly higher path delays than are present in onion routing [20] architectures. Our attack does not distinguish whether a target is the source of a file or a trusted friend of the source, but as we discuss, both are crimes within our forensic goals.

Second, we show that OneSwarm's vulnerability to attackers that collude to compare probabilistically forwarded traffic is much greater than was previously reported. We use a corrected analysis that also considers the implications of content popularity and is based on a realistic legal model. As in papers that demonstrated flaws in the Vanish p2p system [7, 25], we show that the steep growth in effectiveness of the attack as the population of attackers grows was not considered in the original analysis of OneSwarm. We show for this second attack that when investigators comprise just 25% of peers, over 40% of the network can be investigated with 80% precision to find sources of content that is not popular. The attack's success increases with content popularity.

Third, we analyze the OneSwarm source code and find that differences with the technical paper significantly reduce the security of the system. Specifically, the probability of forwarding is hard-coded to $p = 0.95$ instead of 0.5. For the implementation in use by thousands of people, attackers that comprise 25% of the network can successfully use this second attack against 98% of remaining peers with 95% precision due to this value of p. In comparison, should attackers comprise 25% of an Onion Routing network, at most 8% of the circuits are vulnerable in an analogous attack.

Finally, we demonstrate a novel application of a known TCP-based attack, which allows a single attacker to identify whether a neighbor is the source of data or a proxy for it. Using optimistic acknowledgements of data not yet received, an attacker can drive up a target's TCP send rate. If the target is a proxy rather than a source, it will run out of data to send before the true source can supply it. Hence, sources and proxies are distinguishable. Since this requires a single attacker, the entire network can be attacked with a trivial number of attackers — however, only users that turn off the default rate-limit setting are exposed. Onion Routing is not vulnerable to this attack, due to chained proxies. For this and the other two attacks, investigators can increase attack success by repeatedly leaving and re-joining the network to expose a new set of peers.

While our attacks are based on OneSwarm's design, our results are applicable to broad design principles for anonymous communication systems. For example, our TCP-based attack works on anonymous systems that don't use onion-based encrypted layers when streaming data, and several such open-source systems exist, including MUTE [16] and RShare [17]. We revalidate the common wisdom that there is a tradeoff between privacy and performance, and we quantify OneSwarm's anonymity so that it can be compared directly to Onion Routing. We also develop a different attacker model, one based on a conservative set of legal restrictions.

2. OVERVIEW OF ONESWARM

OneSwarm is a popular p2p system for anonymous file sharing. There are North American (`http://oneswarm.cs.washington.edu`), French (`https://forum.oneswarm-fr.net`), and Russian (`http://oneswarm.ru`) communities each with many thousands of users. Below, we include details of only the OneSwarm mechanisms that are relevant to our analysis. We examined the source code available from `http://www.cs.washington.edu/homes/isdal/OneSwarm-20110115.tar.bz2`, which is version 0.7. We note any relevant differences between the technical paper and source code.

OneSwarm is based on a dense topology of peers that discover each other through a community server. Neighboring peers can be *trusted* or *untrusted*. Trust is assigned by the user, and trusted peers see none of the delays or other mechanisms that OneSwarm introduces, as if they were standard BitTorrent peers. The content shared by trusted friends is displayed explicitly in the OneSwarm GUI.

Topology Construction. Each peer has 26 neighbors, and they can be added from out-of-band methods (such as email or social networking sites) as trusted or untrusted friends, or assigned as untrusted friends by the community server. The simplest method of investigation is to be randomly assigned to peers by the community server, and that is the case we focus on here. The source code assigns peers between 22 and 39 neighbors; as peers quit, community server can assign more peers to clients. When a target does have a trusted friend, all privacy controls are turned off, and therefore becoming a trusted friend is an appealing method of investigation; we don't investigate such an approach in this paper. But we do quantify the affect of trusted friends on our attacks, and we summarize the legal implications, which favor law enforcement, in Section 4.3.

Neighboring peers communicate via SSL over TCP. Key exchange is based on an underlying DHT; we elide the details. The public/private keypair used by each peer does not change and is stored on the local computer. The public keys of neighbors are also stored on the user's local computer. The user's keys and neighbors' keys are never deleted (until the application is uninstalled) and are useful corroborating evidence.

Searching for Content. OneSwarm is strongly linked to BitTorrent, and peers can search for content by flooding a query containing a text string or by a unique BitTorrent *infohash*, which is a standard method of uniquely identifying a torrent. When content is found, peers indicate they have a path to the content, without disclosing whether they are the source of the content or just a proxy to it. Pieces of the torrent are then swarmed from all remote peers that provide a path.

When a OneSwarm peer possesses queried content, it will return a *search reply* message after some delay but not forward the query any further. In the OneSwarm paper, the delay is selected uniformly at random from 150–300ms. The choice is consistent (random but deterministic) for the match-

ing content (by info hash). Two neighbors that query a source for the same file will see the same delays. A peer that queries for two different files from the source will see different delays, but when a query is repeated it will see a consistent delay. In the source code, query replies are returned with a delay of 170–340ms. In the case of infohash searches, the delay is chosen on the basis of the infohash; for text searches, the delay is selected based on the content that matches the query. When intermediaries receive a search reply, they forward the message along the reverse path back to the original querier. The reply messages are forwarded by intermediaries with no delay in the source code, though the paper specifies that all OneSwarm protocol messages are delayed.

If the peer does not possess the queried content, it forwards the query to each of its neighbors with probability p; the choice to forward a specific query to a specific neighbor is random but deterministic. In the paper, forwarding of queries is delayed again by 150–300ms, a value chosen at random but again consistent for the specific query and the neighboring peer. In the software, query replies are forwarded without delay. In the paper, $p = 0.5$ is a suggested value, but it is stated that "privacy-conscious users are free to decrease their forwarding probability". However, the software follows a different design. p is set to a much higher value of 0.95 and there is not yet a user-visible method to change the value of p; users must edit and recompile the source. As we show, this setting greatly reduces the privacy of the system.

Once the querier receives a reply, the content is requested and relayed through the path of OneSwarm peers using the BitTorrent protocol. There are no direct downloads between peers unless they are neighbors, but peers cannot naively identify these direct connections.

OneSwarm messages have no time-to-live (TTL) fields, as they would allow attackers to determine if a neighbor is a source of a file by falsely setting the TTL of an outgoing query to 1. Without TTLs, queries might cause congestion collapse from unlimited traffic, and so OneSwarm uses another mechanism. As search replies are returned along the reverse path to the querier, *search cancel* messages are sent to any neighbor that received the original query. These cancel messages are sent without delay and are designed to catch up to and stop the propagating query.

Like BitTorrent, OneSwarm allows a form of parallel downloading (called *swarming*) that Onion Routing implementations, like Tor, do not support well. Our analysis of OneSwarm's privacy is so that users can evaluate if the performance benefits are worth the decline in security in comparison. The reason Onion Routing does not support swarming well is that a separate 3-proxy tunnel is needed to each peer offering part of a torrent.

Other details of OneSwarm's operation do not introduce vulnerabilities that we've discovered, and we do not describe them further here.

3. PROBLEM STATEMENT & MODEL

In this section, we provide a problem statement, attacker model, and our assumptions. Related work appears inline.

3.1 Problem Statement

OneSwarm was designed to thwart third parties from monitoring peers on its p2p network. For years, academic projects [3, 15] have measured p2p networks. Private companies such as DtecNet, Peer Media Technologies, and Media Defender have also monitored p2p networks to assist copyright holders in filing civil copyright infringement lawsuits.

For more than a decade [22], p2p networks have been an enormous venue for the distribution of child sexual abuse imagery, according to the US Dept. of Justice [21], past work by ourselves [11, 12], and others [5, 10]. Our work with law enforcement identified over 425,000 Gnutella users, as identified by application-level IDs, sharing known files of child pornography (CP) during 2010 [11]. Our focus is on US-based criminal investigations of OneSwarm peers for two reasons. First, criminal investigators are at a disadvantage compared to other civil investigators. The former are far more restricted by the legal process defined in the US for law enforcement. Accordingly, this process provides a lower bound on OneSwarm's vulnerabilities. Second, our law enforcement partners have verified there is CP shared on OneSwarm, and thus it is of interest to them.

Investigator Goals. The investigator is essentially an attacker, attempting to violate OneSwarm's privacy promises, though more limited in ability than is typically assumed [24]. Their goal is to *identify* a subset of all OneSwarm peers that are each sharing (or conspiring to share) one or more *files of interest*. The files are content that is known to be CP, and this represents a small fraction of files shared on the network. The overriding goal of the attacker is to gather evidence sufficient for a search warrant so that the physical location corresponding to a peer's IP address can be searched, and relevant physical evidence seized. Hence, we consider success only when evidence can be gathered that is sufficient at least for *probable cause*.

The OneSwarm threat model is designed to be resistant to the disclosure of user behavior to an attacker with control over a limited number of overlay nodes and that wishes to monitor millions of users. In contrast, our investigators want to quickly gather sufficient evidence for a subset of actors that are breaking the law, all while expending minimal resources. The investigator is not looking to catch everyone at once, but can repeat attacks over time.

Probable cause is a lower standard than the *beyond a reasonable doubt* standard needed for conviction. There is no quantitative standard for probable cause, and courts have defined it only qualitatively as a "fair probability" ; see *United States v. Sokolow*, 490 U.S. 1 (1989). Accordingly, we say a peer has been *identified* if the investigator's statistical confidence is above a level sufficient to serve as probable cause. While Isdal et al. analyze attacks requiring 95% precision, we believe that 80%, or perhaps even lower, is still a conservative quantification of this standard, though of course it depends on context. We evaluate different scenarios throughout this paper.

Evidence is *forensically valid* if gathered using techniques that are based on testable hypotheses, have a known error rate, are based on accepted scientific methods, and are peer reviewed; see *Daubert v. Merrell Dow Pharma.* 509 U.S. 579 (1993).

A basic tenet of criminal forensics is that investigators have limited resources. Like narcotics crimes, the number of persons arrested for CP possession is generally limited by investigator resources, not by the number of suspects. Each identified peer requires weeks of additional legal processing, and so maximizing the number identified is not necessary for success.

3.2 Model and Assumptions

The general approach adopted by law enforcement for investigating CP trafficking is based on a series of legal restrictions. We assume our attacker is in fact an investigator in the US and following these restrictions. The law places limits on how investigators can gather information prior to the receipt of a search warrant. The primary restriction is that investigators can only gather information in *plain view*. For p2p networks, any traffic is in plain view if either: *(i)* in the normal operation of the protocol the traffic can be observed by any peer on the normal path (for example, search queries that are broadcast to all peers, including the attacker); *(ii)* if the traffic is destined for the attacker (for example, a response to the attacker's search query). *U.S. v. Gabel*, 2010 WL 3927697 is a recent ruling that confirms this procedure's legality. We do not allow attackers in our model to analyze or compromise the contents of encrypted traffic. We do not allow attackers to seize or compromise peers through privilege escalation as that is outside the plain view standard.

Our attacker controls some fraction of all peers in the network. We detail three specific attacks on the anonymity of OneSwarm in the following sections, and describe the value of the evidence investigators discover from these attacks. Based on these attacks, an investigator can determine if a specific IP address is of interest. The first attack leverages timing information; the second leverages traffic information; the third leverages TCP information.

With a magistrate-ordered subpoena, the billing record of an IP address provides a specific geographic location. With the attack's evidence provided as part of a court-issued warrant, this location is searched, any computer systems and media are seized, and the media are examined for other evidence of the possession or distribution of CP. Such evidence includes CP burned to a DVD or other unassailable [2] evidence of intent to possess. Often the evidence gathered for the search warrant is not introduced at the criminal trial unless the search warrant is challenged; thus it is essential to investigations but must meet only the lower probable cause standard.

In comparison, a civil claim of copyright infringement need meet much lower standards. While a warrant cannot be issued by civil investigators, the billing records *and* computers at the location can be subpoenaed based on merely a *relevance* standard. Evidence is relevant if it "has a tendency to make any fact more probable or less probable than it would be without the evidence" [23]. As previous work has shown [14] this bar is very low, and the smallest success from our attacks would be sufficient for civil subpoena — precision of 95% confidence, as we can achieve, is for a civil subpoena extraordinarily high. Once at trial, only a *preponderance of evidence* standard must be met for a judgment to be awarded.

4. TIMING ATTACKS

In this section, we demonstrate that the current design of OneSwarm is subject to a novel timing attack. We also show that the deployed version of the system has this flaw. We then derive a configuration of OneSwarm delays that deters the attack; however, the new delays that OneSwarm enforces for responding to queries must be increased to 133%–400% of the delays imposed by Onion Routing. Further, the resultant search traffic on OneSwarm floods potentially thousands of nodes.

The basic design goal of OneSwarm is to prevent attackers from distinguishing between two cases for some targeted peer T and the source of some file (we use *file* and *content* synonymously):

- **Case A:** T **is** the source a particular file;
- **Case B:** T **is not** the source a particular file. Instead T is only a proxy on the path to some other peer it does not know, S, that is a source.

There is a major difference between the goals of the OneSwarm security model and the goals our forensic analysis. When the source of a file is a trusted friend, the OneSwarm design considers that as an example of Case B. In our analysis, we consider it as an example of Case A since the peer has explicit knowledge via the OneSwarm GUI of what its trusted friends are sharing on the network, and the peer is therefore a knowing source of the content as well; we return to this point in Section 4.3.

OneSwarm's design defeats a **Basic Timing Attack** where a single peer A issues a query to its neighbor T. The attacker compares the application-level response time to the network-based roundtrip time; if they are similar, then the attacker judges that T is the source.

To defeat the attack, when a peer *is* the source a requested file, it introduces an artificial delay of 150–300ms before answering queries, as described in Section 2. This response delay, r, is chosen randomly but deterministically for the specific file.

If a peer *does not* possess a queried file, it forwards the query on to its neighbors only after a delay, q, which is 150–300ms in the paper (but exactly 150ms in the current implementation). This added delay allows the original querier and intermediaries the chance to send a query *cancel message* that will catch up with the propagating query. Recall from Section 2 that the cancel messages are essential in preventing traffic explosions, since there are no TTL fields in OneSwarm messages.

OneSwarm peers do not keep a cache of previously answered queries. However, we discuss the effects of query and response caching on this attack later in the section and show that the attack is not substantively affected.

The attack makes use of two central variables:

- l is the one-way network layer delay between two peers. We assume such delays are symmetric and let $RTT = 2l$.
- δ is the application-level roundtrip delay in receiving OneSwarm search query responses from a source. Since OneSwarm enforces extra delays, this is not equal to the RTT.

4.1 The Twin Timing Attack

OneSwarm is vulnerable to a new attack using *simultaneous queries* from two attackers, which we dub the *twin timing attack*. The peers don't need synchronized clocks, but the two queries must be issued before the target changes from possessing the file to not possessing it, and before network-layer conditions change drastically. The complete attack appears in Algorithm 4.1 and refers to two attackers C_1 and C_2, both attached to a target peer T.

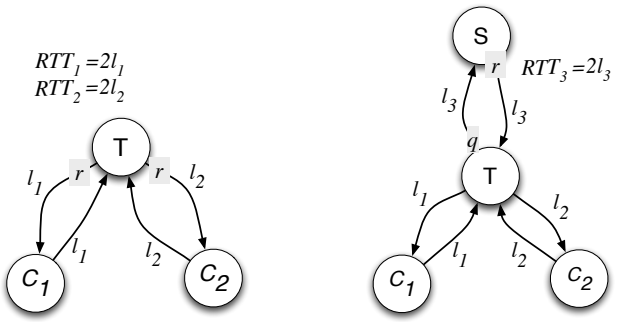

Figure 1: The attacker attempts to distinguish two scenarios. In Case A, peer T is the source of queried content. In Case B, peer S, one hop from T, is the source.

Algorithm 4.1: TwinTimingAttack(T)

1. C_1 and C_2 each measure their respective network roundtrip time to T as RTT_1 and RTT_2, respectively.
2. C_1 and C_2 simultaneously query T for the same content.
3. Let δ_1 and δ_2 be the total delays after which C_1 and C_2 receive replies, respectively.
4. If $(\delta_1 + \delta_2) - (RTT_1 + RTT_2) < 600$ ms, then T is the source.

THEOREM 1: If T is the source of the queried content, then $(\delta_1 + \delta_2) - (RTT_1 + RTT_2) < 600$ ms; else T is not the source.

PROOF: This proof uses the delays defined in the OneSwarm paper; the case for the delays defined in the software is discussed subsequently. **Case A** is illustrated in Fig. 1(left) where

- T is the source of a file;
- l_1 is the network delay between C_1 and T;
- l_2 is the network delay between C_2 and T;
- r is the response delay at T for a query, whether by C_1 or C_2.

The total delay for each attacker is

$$\delta_1 = 2l_1 + r \quad (1)$$

$$\delta_2 = 2l_2 + r \quad (2)$$

Let sum_A equal the sum of δ_1 and δ_2 for Case A.

$$sum_A = 2l_1 + r + 2l_2 + r \quad (3)$$

Because r is chosen from 150–300ms, we can bound sum_A as

$$sum_A \leq 600 + 2l_1 + 2l_2 \quad (4)$$

Case B is illustrated in Fig. 1(right) and adds the following variables.

- T is not the source S;
- l_3 is the network delay between T and S;
- q is the delay at T before it forwards the query to S;

In this case, the total delays are

$$\delta_1 = 2l_1 + q + 2l_3 + r \quad (5)$$

$$\delta_2 = 2l_2 + q + 2l_3 + r \quad (6)$$

Let sum_B equal the sum of δ_1 and δ_2 for Case B.

$$sum_B = 2l_1 + 2q + 2l_2 + 2r + 4l_3 \quad (7)$$

Because all application-level delays are chosen from 150–300ms, we can bound sum_B as

$$sum_B \geq 600 + 2l_1 + 2l_2 + 4l_3 \quad (8)$$

Let RTT_1 and RTT_2 be the measure network roundtrip time to T from C_1 and C_2, respectively. When T is not the source (Case B), the difference of the summed network RTTs and the total response delay summed is *at least*

$$sum_B - (RTT_1 + RTT_2) \geq 600 + 4l_3 \quad (9)$$

When T is the source (Case A), the difference of the summed network RTTs and the total response delay summed is *at most*:

$$sum_A - (RTT_1 + RTT_2) \leq 600 \quad (10)$$

In short, the attack works because the minimum value of sum_B is always larger than maximum value of sum_A.

Therefore, if the difference between $2(l_1 + l_2)$ and $\delta_1 + \delta_2$ is less than 600 ms, T is the source or T is knowingly conspiring with the source as per Section 4.3. □

Caching query results. OneSwarm does not cache queries or query responses at peers. Doing so would thwart the attack as it would not be clear if a response was from a cached result or the original source. To defeat this defense, the timing would instead be on the request for the actual content. We assert that requests for content must also have delays as queries for content (forwarding and response) otherwise the Basic Timing Attack and variants would be possible. To defeat the twin timing attack on the requests for content, OneSwarm peers would have to cache actual content that was recently transferred, which is an enormous change to the OneSwarm's design. It would increase the resources required of peers dramatically.

4.2 Software Vulnerability to the Attack

In the OneSwarm software, r appears to be chosen between 170–340ms for text queries. If the node does not have the file, it will forward the query after a delay of $q = 150$ms but it appears that there are undocumented delays as well. An added detail is that if a query is for a specific infohash, the operation is slightly different. First, the source chooses a value for r between 170–340ms. Then, the querier learns the torrent's meta-information after this delay, it requests specific pieces, and the responses to each are delayed by 20–40ms by the source. We analyze only the text query process in this paper. When responses to a query are forwarded to an intermediate peer there is never a delay in forwarding. With these settings in the software, the proof remains valid. However, the multi-threaded nature of the software makes it impossible for us to be sure of its actual operation. In fact, in examining the software, we found unintended delays unknown to the developers themselves, which were fixed in later versions that we did not test. An added delay of about 100 ms is unintended in the code according to the developers. We rely on experimentation to empirically validate the attack on the software since its exact operation is unclear.

We implemented the attack on a set of four machines within our building on separate networks, taking the roles

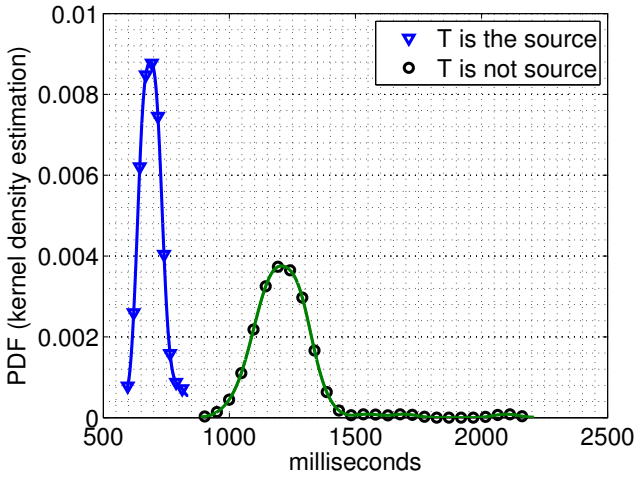

Figure 2: Results of the Twin Timing Attack implementation with 200 trials on a small network. The distribution of timing values is presented using a kernel density estimator. The two cases are from two distinctly different distributions.

shown in Figure 1. We did not attack any real users of OneSwarm because we did not have any ground truth for what they actually shared, so the experiments would have been impossible to evaluate. The attack fails when RTT variance is high and would be less successful across some Internet paths. However, if we had performed the attack on a node we controlled on the Internet, we would have not been able to isolate the aspects of OneSwarm's implementation that allow the attack. Moreover, we could not have asserted that the amount variance we observed would be typical of every path on the Internet. Stronger security is more often provided by a known design quality rather than relying on protean characteristics of Internet traffic that cannot be controlled or relied on, and we tested the former case here.

In our tests, each machine was running version 0.70 of OneSwarm; peers C_1 and C_2 were instrumented to display the relevant timing information, and network-level RTTs were determined by ICMP pings. We show the results of 200 trials of the attack in Figure 2 for 200 files, one trial per file. The results for each case is displayed using a kernel density estimator (KDE). The two cases, of possessing or proxying the file, are clearly and always differentiable as samples from distinct distributions. In our experiments, the maximum summed application-layer delay was 816 ms when T was not the source. The minimum application-layer delay when T was the source was 996 ms. The KDE representation of the distribution increases the maximum and minimum of each case, yet the distributions remain distinct. As we noted in the introduction, when attackers comprise 15% of OneSwarm network, we expect 90% of the remaining peers to be connected to two attackers and therefore vulnerable; see Eq. 20 in Section 5.

4.3 Trusted Neighbors

Trusted relationships are an important aspect of OneSwarm's design. When a neighbor is trusted, there are no delays for OneSwarm messaging. Accordingly, the Twin Timing Attack will not always distinguish between a target and its trusted friends as sources of a specific file, however, this is not a problem.

When peers act as a proxy for a trusted friend sharing child pornography, it is also a serious crime, and the attack's result is sufficient evidence of such criminal distribution. Filenames shared by trusted peers are shown in the target's GUI, and therefore a target of the Twin Timing Attack is distributing CP with knowing intent — the target can be charged with conspiracy to distribute; see 18 U.S.C. Sec. 2252A(b)(1). Furthermore, by setting a trusted relationship, the target gains a non-pecuniary benefit of better performance, which incurs a greater punishment; see US v. Schaefer, 472 F.3d 1219 sentencing memo, and USSG Sec. 2G2.2(b)(3)(B). Evidence found at the target could serve to support a search warrant of the trusted peer that is actually sourcing the child pornography, and so finding the proxy for a source of CP is a good strategy for investigators. In short, trusting a neighbor raises a significant criminal liability. While the system is designed to obscure whether a peer or its trusted neighbor is sharing a file, the distinction does not concern or thwart law enforcement investigations as in either case the target is breaking the law. The same conclusion can be drawn for civil investigations, where the bar of relevance is lower; see Section 3.2.

4.4 Deterring the Attack with Longer Delays

We now show that defeating the basic timing attack and the twin timing attack requires that the range of response delay r be larger than the forwarding delay q. However, the *search cancel* messages place a lower bound on the value of q. Under these constraints, finding content 1 hop away will require a delay of 4 times an expected Internet RTT. Finding content 3 hops away requires an additional delay of 12 times the expected RTT. While Onion Routing does not directly support search for content, its chain of 3 proxies (at a cost of 3 times the normal RTT) has much better performance when searching for torrents via Web sites, or when proxying the traffic of p2p systems that do support search.

First, we note that to defeat the Basic Timing Attack, OneSwarm enforces the following constraint for an expected RTT [8].

$$min(r) \geq RTT \tag{11}$$

With this minimum delay, a response from a target could appear to have come from a trusted peer, one hop away from the target. In the design, this value is chosen between 150–300ms to emulate the expected Internet delay of one to two hops.

THEOREM 2: To defeat the twin timing attack, the following constraint must hold

$$max(r) > min(q) - min(r) + RTT_3 \tag{12}$$

where RTT_3 is the roundtrip time between the Target and the Source as shown in Figure 1(right). We let $max(\cdot)$ and $min(\cdot)$ represent the maximum and minimum values of a given delay variable taken from uniform distributions.

PROOF: The twin timing attack is possible because of the difference between the summed total delay for Cases A and B. Based on logic similar to our proof of Theorem 1 (Eqs. 9 and 10), the attack is defeated if

$$min(sum_B) - max(sum_A) < 0 \tag{13}$$

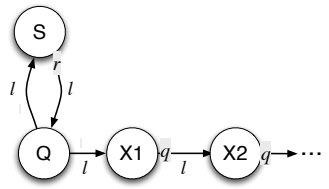

Figure 3: OneSwarm query and cancel message delays.

By first substituting Eqs. 3 and 7 (from the Proof of Theorem 1) into Eq. 13, it can be reduced to

$$min(2q + 2r + RTT_1 + RTT_2 + 2RTT_3) \\ -max(2r - (RTT_1 + RTT_2)) \quad < 0 \quad (14)$$

The target does not control the min or max values of the network-determined RTT, and thus must assume the worst case of $min(RTT) = max(RTT)$. Therefore, we have

$$min(q) + min(r) + RTT_3 - max(r) < 0$$

which is equivalent to Eq. 12. The Theorem holds, for example, when $q = 0$ and the range of r is greater than RTT_3. Since T can measure RTT_3 and it sets r, it can defeat the twin timing attack with correctly set delays. Moreover, it is not possible for the attacker to measure RTT_3.

□

However, when $q = 0$, cancel messages will not work at all.

We now prove the constraints that are necessary for cancel messages to work correctly. Unlimited propagation of query messages would prevent OneSwarm from scaling to large topologies: all nodes would receive all query messages, which is untenable.

Cancel messages are illustrated in Fig. 3.

- Q is the querier of content;
- S is the source of the content;
- $X1$ will forward the query to its neighbors if not stopped by Q;
- q is the query delay randomly chosen by $X1$ before it forwards the query to the rest of the network.
- r is the response delay chosen at random by S for a query by Q.

Once Q receives a response from S, it will immediately forward a cancel message to $X1$. In this scenario, $X1$ is representative of T's 37 other neighbors; $X2$ is representative of $X1$'s neighbors, each having 38 neighbors, and so on.

THEOREM 3: For a querier's cancel messages to stop all instances of the query message within h hops of Q, then

$$max(r) < min(q)h - 2l. \quad (15)$$

PROOF: We compare the latencies of the two messages as illustrated in the figure (dropping subscripts for link delays). For the cancel message to stop search queries within h hops, the following inequality must hold

$$max(r) + 2l + h\,l < h(min(q) + l) \quad (16)$$

which directly reduces to Eq. 15.

□

THEOREM 4: The constraints of the Basic Timing Attack, Twin Timing Attack, and search cancel messages imply that

$$min(q) > (6l)/(h - 1). \quad (17)$$

PROOF: Starting from Eq. 12, we substitute $max(r)$ with the bound from Eq. 15. By then substituting $min(r)$ with the bound from Eq. 11, we have the inequality stated in Eq. 17.

□

4.5 Performance Implications

Given that we have bounds on q and r that are required to defeat these attacks while ensuring cancel messages work, we can now determine the performance implications for OneSwarm. Specifically, we calculate the length of time to receive query responses from sources for these new delays. A large component delays in any implementation of Onion Routing come not from network latency, but rather from non-intentional application-layer queuing delays, and this is true in OneSwarm as well. Here, we perform a comparison of minimum delays imposed by the architecture of the systems that enables privacy, independent of traffic load or other external factors.

First, we determine for the existing OneSwarm implementation the number of hops queries can travel before being cancelled by a source one hop from a querier (as in Fig. 3. In the source code, $max(r) = 340$ms, $min(r) = 170$ms, and $q = 150$ms. Therefore, when $l \approx 75$ms, and from Eq. 15, we know that $h > (max(r) + 2l)/q = (340 + 150)/150 = 3.26$; therefore $h = 4$ since it can only take on an integer value. We can show that the earliest that messages can be stopped is $h > (min(r) + 2l)/q = (170 + 150)/150 = 2.13$; therefore, messages cannot be stopped before $h = 3$ hops from the querier. For smaller values of l, $2 \leq h \leq 3$, which we assume below as a conservative estimate.

Second, given this value, we can determine the number of peers that receive traffic for each query, even if answered by a direct neighbor. When $h = 2$, and $p = 0.95$ and with a full topology, each query will reach $(39 * 0.95)^2 = 1373$ peers. When $h = 3$ this value increases to 50,859 peers; the importance of limiting $h = 2$ is clear.

Third, given $h = 2$, we can now solve for values of q and r that protect OneSwarm users. From Eq. 17: $q > \frac{6l}{2-1} = 6l$; and from Eq. 11: $min(r) \geq 2l$; and from Eq. 12: $max(r) > 10l$.

Finally, these values allow us to compute, for a version of OneSwarm immune to timing attacks, the expected time t it takes to receive a response from a source x hops away is $E[t] = 8xl$. For x=1, $E(t) = 8l$; For x=3, $E(t) = 24l$.

In comparison, because Onion Routing always consists of a chain of 3 proxies, the delay in receiving data from a Torrent search engine is $E[t] = 6l$. Therefore, for any query, OneSwarm is always one RTT slower in terms of roundtrip delay; for many queries OneSwarm is about 4 times slower. The number of nodes that receive query traffic (in the thousands) is inefficient compared to contacting a single web server over an Onion Routing circuit.

5. COLLUSION ATTACK REVISITED

The *collusion attack* is a fundamental threat to OneSwarm peers as it based on only two aspects of the protocol's design: that queries are forwarded with probability p when the peer

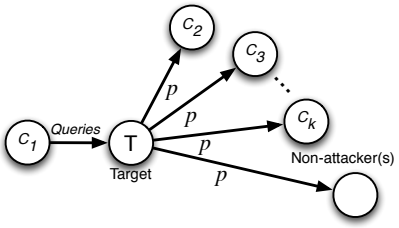

Figure 4: Setting for collusion attack. One attacker queries for content and the remaining attacker each have a probability p of receiving the query if T is not the source.

doesn't have requested content; and that each peer has up to 39 neighbors. If all other attacks we present in this paper are patched, the collusion attack remains. In this section, we demonstrate that Isdal et al. incorrectly calculate the chance of this attack in two ways. First, they overestimate the success of the attack by assuming attackers are chosen with replacement. Second, they underestimate the quick ramp up of the attack's success as the number of attackers grows. The collusion attack is significantly easier in OneSwarm than in OR. For example, we show the chances of success reach near 100% when attackers comprise 25% of the network, whereas success for OR would remain below 10%.

5.1 The Collusion Attack

As described in Section 2, if a peer receives a query and does not have the requested file, it forwards the query to only a subset of its neighbors. This subset is chosen based upon a forwarding probability p.[1] The decision to forward is decided independently for each piece of unique content and each neighbor and remains consistent.

The collusion attack is illustrated in Fig. 4. A set of k colluding attackers labeled $C_1, \ldots, C_k$ are directly connected to a target peer. When C_1 issues a query to target T that has the requested file, T does not forward the query on to its neighbors. Instead it sends back a reply after a randomly chosen delay. Therefore, if none of the other $k-1$ colluding attackers receive the query from T, there is a non-negligible probability that T may be sharing the file requested by C_1. This probability, which we call the attack's *precision*, is greater when k is larger. For law enforcement, higher precision better justifies having sufficient probable cause, as we discuss in Section 3.

Isdal et al. calculate that with $p = 0.5$, achieving 95% precision requires that at least $k = 6$ attackers (a querier and 5 colluders) to be directly connected to the target. The chance that a target that is not a source will forward the query to at least one of the $k-1$ colluders is [8]

$$1 - (1-p)^{k-1} \tag{18}$$

For example, we have $1 - (1-0.5)^5 = 97\%$ (which is greater than 95%). Isdal et al. state the chances that 6 or more colluders are attached to a particular peer when $C = 30$ of the $N = 1000$ peers are attackers is "much less than 1%" and "given by the binomial CDF" [8]. While they don't state a formula, to use the binomial, we let A be a random variable

[1]Here $p = 1 - p_f$, where p_f is Isdal et al.'s notation for the probability of *not* forwarding a query.

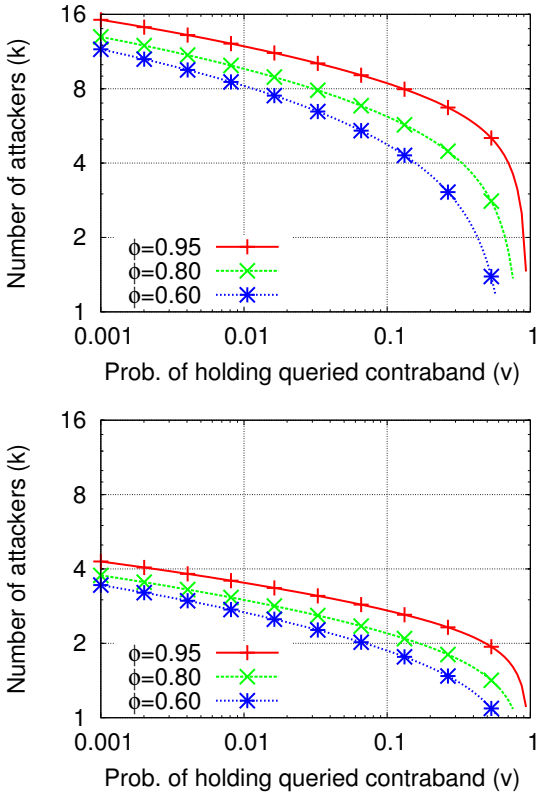

Figure 5: (top) For a fixed probability of forwarding $p = 0.5$ (the value suggested in the paper), the plot shows the required attackers k given the the popularity of content q from Eq. 23. Each line is a different precision require ϕ; (bottom) The same plot for fixed probability of forwarding $p = 0.95$ (the value hardcoded in software) again from Eq. 23. Both graphs use logscales.

that denotes the number of attackers assigned to a target. For N peers of which C are attackers, each with 26 neighbors, we have

$$P\{A \geq k\} = \sum_{i=k}^{26} \binom{26}{i} (\frac{C}{N})^i (1 - \frac{C}{N})^{26-i} \tag{19}$$

Unfortunately, the binomial is not entirely accurate, and the statement "much less than 1%" does not give a complete picture.

5.2 Re-Deriving the Attack

We re-derive the attack effectiveness with several changes. First, because community servers select neighbors for peers *without* replacement, the hypergeometric CDF is the correct model. Second, the paper states peers have 26 neighbors, but in the code, peers have at least 22 but up to 39 untrusted neighbors. We let U be the number of untrusted neighbors of a peer.

$$P\{A \geq k\} = \sum_{i=k}^{U} \frac{\binom{C}{i}\binom{N-C}{U-i}}{\binom{N}{U}} \tag{20}$$

Therefore, we have in the worst case that $U = 39$ for Eq. 20.

In the best case, $U = 0$. In reality, for each trusted neighbor, there is a probability that it is an undercover investigator, but it is not a scenario that we evaluate here.

The next step is to determine the value of k that defeats OneSwarm, and we do not use Eq. 18. Instead, we let X denote the event that the target T has content that was searched for, and let Y denote the event that none of the $k-1$ colluders were forwarded the search query issued by C_1. From Bayes' Theorem, we can define the precision of the collusion attack as

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y|X)P(X) + P(Y|\bar{X})P(\bar{X})} \tag{21}$$

We know that $P(Y|X) = 1$ because given that T has the content, the $k-1$ colluders will not be forwarded the search query. In general, not every peer will have the files of interest that the attacker queries for; to model this situation simply, we let $P(X) = v$, where $0 \leq v \leq 1$ interpreted as the popularity of some file of interest being queried. We know that $P(Y|\bar{X}) = (1-p)^{k-1}$. Finally, we let $\phi = P(X|Y)$, and substituting we have

$$\phi = \frac{v}{v + (1-v)(1-p)^{k-1}} \tag{22}$$

Solving for k we get

$$k \quad = 1 + \frac{\log\left((\frac{v}{1-v})(\frac{1-\phi}{\phi})\right)}{\log(1-p)} \tag{23}$$

Fig. 5(top) plots Eq. 23 for $p = 0.5$ (the value suggested in the paper) showing the minimum k value required for different precision levels and v as an independent variable. Fig. 5(bottom) shows the same equation for $p = 0.95$ (the value hardcoded in software). Each plot shows three lines corresponding to precision values of 95%, 80%, and 60%. The first is the value used in Isdal et al., the second is conservative for probable cause, and the last is weak evidence for probable cause. (Note that the value of k is independent of U.)

For example, for $p = 0.5, v = 0.1$, and $\phi = 0.95$, then the attack requires $k = 8$ attackers (rather than 6); however, the more important point is that the value of k varies quite a bit with v. Roughly, as content is an order of magnitude more popular, k typically halves in size. Comparison of the two plots makes the obvious point that p also has a strong influence on k. Releasing the software with a higher value for p than documented in the paper reduced the required number of attackers by 60–75% in all cases. Finally, when $p = 0.95$, the plots demonstrate that values of $k \geq 4$ are definitely sufficient for probable cause for content that is sourced at only one in a thousand peers.

The *false positive rate* for the collusion attack is the probability that the target does not forward the query to any of the colluding attackers given that it is not the source of the file of interest:

$$FPR = (1-p)^{k-1} \tag{24}$$

This quantity is less than 0.0025 for $k \geq 3$ and $p = 0.95$. Thus investigators are at a very low risk of falsely suspecting any peer in OneSwarm as deployed. When $p = 0.5$ the false positive rate is less than 1.6% only when $k \geq 7$; note that for files less popular than $v \leq 0.06$, the same bound of $k \geq 7$ holds for precision values of 95% (see Fig. 5(top)) and so the requirement of a low FPR is not an significant issue for the investigator. Querying for multiple files of interest will affect the FPR, as we discuss in Section 5.3.

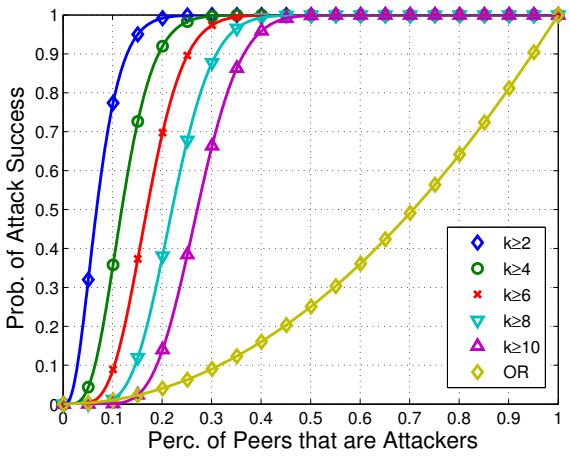

Figure 6: Plots of collusion attack success (Eq. 20 where $U = 39$) and first-and-last attack against OR (Eq. 25): The probability of success for the collusion attack on OneSwarm for a given required minimum value of k. The plot also shows the comparable attack success against OR (first and last peers on a circuit).

Comparison Against Onion Routing. Isdal et al. do not quantitatively compare OneSwarm directly against any other privacy mechanism. As a basic comparison we analyze the following simple OR attack. Peers hiding behind a OR circuit can be deanonymized if attackers are selected at random to be in the first and last positions. Selection of these nodes occurs without replacement. Once in these positions, attackers can use a well-known attack of sending a specific sequence of duplicate packets to determine if they are on the same path [20], essentially with a precision of 1 and FPR of 0. Therefore, the chances of a circuit being compromised in a OR network of N peers where C are attackers is

$$P\{A = 2\} = (\tfrac{C}{N})(\tfrac{C-1}{N-1}). \tag{25}$$

Mechanisms such as *guard nodes* [26,27] can make this attack on OR more difficult. However, in general the simplicity of the model in Eq. 25 prevents us from comparing OneSwarm to Tor directly, instead of Onion Routing.

We use this passive attack because it's a tractable, conceptually simple presentation of the collusion attack on the OR architecture, allowing a straightforward comparison with OneSwarm's architecture. A more accurate model would consider the active attacks that are available against Tor, which are more effective than Eq. 25. This would require updating Bauer et al. [1] with the latest Tor path selection algorithm and measurements of Tor, to model weighted path selection. It wouldn't change our analysis of OneSwarm, and we expect it wouldn't change the relative comparison, barring a gross bias in the Tor route selection algorithm.

Fig. 6 compares effectiveness of these attacks against OneSwarm and OR. The plot is independent of a chosen value for p; to determine the required value of k, we first choose a value of p and consult Fig. 5(top or bottom). There are a number of implications to note. First, there is a non-linear,

sharp increase in attacker success as they increase their proportion of the OneSwarm network. When the attack requires only that $k \geq 4$, the chances of success are 98% as attackers comprise 25% of peers in the network. Note that $k \geq 4$ is sufficient for even unpopular content when $p = 0.95$, which is the hard coded value in the released software. However, even if $p = 0.5$, requiring larger values of k, the effectiveness of the attack on OneSwarm grows far more quickly than the simple attack on OR.

All of the previous analysis assumes that the attackers join the network once. As such, they can only investigate peers to whom they've attached sufficient colluders. But attackers can repeatedly quit and rejoin the system with new identities, thereby investigating more and more peers over time. OneSwarm offers no Sybil attack protections.

Trusted Neighbors. Our attacks are worst case in that we assume the peer has 39 neighbors. To give a point of comparison, if the node has only $U = 20$ neighbors from the community server (and 19 trusted friends) then the investigator's job is harder. For example, the $k \geq 4$ line in Fig. 6 shifts roughly to almost where $k \geq 8$ is in the figure; these values can be compared more directly by re-plotting Eq. 20. Again, attackers can execute this attack as many times as necessary — churn in the OneSwarm population will force the community server to assign and re-assign new untrusted relationships.

5.3 Practical Considerations

The Multiple Comparison Problem. A naive version of the collusion attack would see investigators joining the network as described above. The investigator would then test each connected peer for possession of all known files of interest, and claim probable cause for each peer which was determined to have at least one file. This naive version is susceptible to the well-known *multiple comparison problem* of testing sets of hypothesis at once. We elide the analysis here, but we note the false positive rate quickly grows unusably high as the number of files tested per peer increases.

There are several ways to limit the effect of this problem, each of which centers around limiting the number of different files investigators check a peer for possession of. One such method is as follows. First, investigators survey the network for files of interest by searching for well-known keywords. In other p2p networks such as Gnutella and eDonkey2000, these files are typically named with explicit and descriptive names; our law enforcement partners confirm the same appears to be true of OneSwarm. The files are rank ordered by their apparent popularity in the system. Investigators then sweep through the network by performing the collusion attack for the top n files, where n is chosen to keep the FPR acceptably low. Only once these peers have been fully investigated, including offline searches and the subsequent legal process, do investigators repeat their survey to determine the next rank-ordered list to investigate.

Post-Warrant Confirmation. OneSwarm peers connect to one another using SSL bootstrapped by their RSA key pairs. A OneSwarm peer stores keys of neighbors and its own key on the local file system. Once a search warrant is executed and a machine seized, the investigator can tie that machine to specific network traffic based on the recovered key file. Even if the content has been deleted locally, the mechanism can confirm with cryptographic precision if this machine is the machine that transferred it. BitTorrent clients

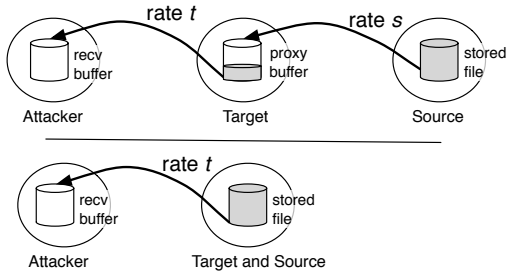

Figure 7: An attacker distinguishes two scenarios. In the first, the target is a proxy for the real source of a file. The goal is to increase the rate of transfer, t, from the target so that it is greater than the rate of transfer from the source, s. The buffer at the proxy should empty before the transfer is completed. In the second case, the target is the source, and therefore, for any value of t, the transfer should complete.

are subject to *forensic tagging* [12], in which investigators covertly ask remote machines to store nonces. These nonces can later be recovered to confirm that the correct machine was seized. However, OneSwarm makes this task redundant since all outgoing traffic is signed and the keys are stored persistently.

6. TCP-BASED ATTACKS

In this section, we demonstrate a novel adaptation of a known TCP-based attack [18] that can identify whether a OneSwarm peer is the source of data or a proxy. Peers that do not rate limit outgoing traffic are vulnerable. OneSwarm happens to turn on rate limiting by default to 80% of a test transfer to a central server, but nothing prevents eager users from turning off the rate limiting. A more robust defense without rate limiting is to probabilistically drop outgoing packets and audit incoming selective acks. In this section, we detail the attack and its limitations, show experimental results for a simplified implementation of the attack executed on a simple non-OneSwarm transfer, and we discuss defenses to the attack.

The attack leverages *optimistic acking* [18], where a receiver sends TCP acknowledgements for data before it is received, increasing throughput. Sherwood et al. [19] leverage the same mechanism to perform denial-of-service attacks against a server. Our contribution is in showing that the same mechanisms can be used to distinguish proxies from sources. The attack is not specific to OneSwarm; anonymous file-sharing protocol designers should be aware of the attack.

Fig. 7 illustrates our TCP attack scenario. An attacker requests a file from a target. The attacker induces a higher-bandwidth connection between itself and target than between the target and a potential source of data. If t can be made greater than any potential s and the target is not the source, it will stall out. The stall occurs because the target's application level buffers will run out before the actual source can fill them. Our tests show that it is possible to induce a sending rate that is higher than is typical even between nodes within the same building.

In our tests, we show that the attack can succeed in practice. In sum, OneSwarm is vulnerable (when rate-limiting is

Algorithm 6.1: DETECTPROXY($maxExtra, rate$)

```
local p, ack, extra, lastReceived, maxReceived
while (download is not complete)
  do { p = RECEIVEPKT(timeoutValue)
       if timeout occurred
         then { ack = lastReceived
                extra = 0
                if too many timeouts
                  then return ("proxy")
         else { lastReceived = SEQNUM(p)
                if SEQNUM(p) > maxReceived
                  then maxReceived = SEQNUM(p)
                extra += rate
                extra = MIN(extra, maxExtra)
                ack = maxReceived + extra
       SENDACK(ack)
return ("sender")
```

off) because it defends against only application-level timing and traffic attacks, and does not defend against an attacker breaking the underlying network abstraction.

Trusted Peers. When a peer has trusted peers (that are not undercover investigators), they reduce the chances that an investigator has an opportunity to execute the TCP attack, but do not affect the success of the attack itself. Vulnerable peers are those that have at least one untrusted relationship to an investigator; i.e., set $k = 1$ in Eq. 20.

6.1 Attack Details

We assume that targets of the attack follow TCP specifications properly, but we place no limit on the connection bandwidth between peers. The attacker does not follow TCP rules for receivers, by incorrectly acking packets that were lost and optimistically acking packets that have not yet been received. The receiver also always advertises a large TCP flow control window so as not to inhibit the sender.

The idealized attack algorithm is presented as pseudocode in Algorithm 6.1. The pseduocode assumes a TCP connection has been set up and a download requested via application-level messaging. The input to the algorithm is the rate at which the optimistic acking increases and the maximum value it can reach. Reasonable values can be found through heuristic hill-climbing.

Whenever a packet is received, the attacker sends an ack for the highest sequence number ever received from the sender, regardless of whether earlier bytes in the TCP flow were lost. Because acks are cumulative in TCP, the skipped over byte sequences are not a concern of the sender. Additionally, the receiver optimistically acks *extra* segments it has yet to receive but are likely on the way. The value of *extra* starts at 0 and the receiver increases it by *rate* bytes (and rounded to segment sizes) for each packet received, regardless of loss or duplication, until a given *maxExtra* value is reached.

The reason to grow this optimistic acking slowly is that the sender will silently drop acks for packets beyond what it has sent. If a sender receives overly optimistic acks, it will not close the connection, as TCP was designed to manage the occasional odd error. Accordingly, thoughtless optimistic acking by the attacker will have no effect, and the attacker must grow the window commensurately with the sender's values. As the window grows, heavy packet loss will occur, and the attack is partly blind in that sense.

If attacker grows the sender's window too aggressively, the RTT calculation at the sender can become mis-estimated as a very small duration, making a timeout exceedingly easy. When a timeout occurs, the sender will back off, cutting the rate drastically, and will resend old data, which is very bad for the attack. Therefore, if the receiver doesn't receive data by some timeout (250ms in our implementation), an ack packet is sent to the sender for the *latest* packet received, rather than the highest received. In our tests, this quickly re-initiates the data flow, and the attacker can return to acking the highest byte sent.

Pipelining. OneSwarm is BitTorrent based, and therefore requests for a file are piecewise. There are seven BitTorrent piece sizes in common use, and the most common are 256kB, 512kB, and 1MB. Our tests show that for the two smaller size, the sender barely gets out of slow start before the attack is complete. Inside of slow start, the congestion window is so small that the attack cannot achieve significant gains. However, pipelining of piece requests has been a feature since the protocol was introduced in 2003 to avoid reseting TCP's bandwidth algorithm and causing "disastrous" transfer rates [4]; this pipelining prevents the attack from stalling.

6.2 Attack Experimentation

We implemented the attack in about 700 lines of C++ to test its feasibility. The attack implementation and details of our measurement experiments (including packet header logs) are available from `http://traces.cs.umass.edu`. Our prototype implementation attacks the HTTP protocol rather than the BitTorrent protocol, as HTTP has fewer implementation details to manage. In principle, the same attack will work on pipelined BitTorrent requests. Further, our implementation does not implement the SSL handshake between peers that OneSwarm requires.

Methodology. We configured a network as shown in Fig. 7. We could not use PlanetLab for this experiment since it enforces a bandwidth cap, which is splits among all virtual hosts on each node [13]. The attacking machines were located at UMass, Wesleyan (CT), and UT Arlington (TX); the targets were a superset of the attacking machines and also included UCSB (CA); and the proxied sources at Central Michigan University, Harvey Mudd College (CA) and Mojohost, Inc. (FL). We chose these proxies to model relatively close (UCSB and HMC are both in CA) and distant (UCSB to Mojohost crosses the country) peers. We served the file directly through an httpd daemon from the targets, and we used netcat to act as an application-level proxy to the other sources. Our target file was a 10MB ISO image. We used wget as the well-behaved requester. We fixed the parameters of our attack implementation at relatively conservative values. We set the *maxExtra* value to 50 TCP segments for all connections, though in our experience this value should vary per bandwidth of the connection to the target. We report the bandwidth speedup of the attack, rather than seeking a *maxExtra* value that causes a timeout (as per Alg. 6.1). We ran measurement experiments across all paths (27 in total), varying type of retrieval (wget or tcp attack) and potential source (direct from UCSB, or proxied to either HMC or Mojohost). Each measurement was repeated five times.

Path			Target Possesses File		Target Proxies File	
Attacker	**Target**	**Proxied Source**	**wget**	**attack (relative)**	**wget**	**attack (relative)**
Wesleyan	UCSB	CMich	1.29 MB/s	*	686 KB/s	*
		HMC	1.25 MB/s	*	646 KB/s	*
		Mojohost	686 KB/s	*	714 KB/s	*
	UMass	CMich	3.66 MB/s	1.43	1.09 MB/s	*
		HMC	3.56 MB/s	1.96	1.04 MB/s	*
		Mojohost	3.57 MB/s	1.44	933 KB/s	*
	UTA	CMich	350 kB/s	1.90	311 KB/s	1.07
		HMC	385 KB/s	1.46	329 KB/s	1.05
		Mojohost	344 KB/s	1.26	304 KB/s	1.21
UMass	Wesleyan	CMich	281 KB/s	13.5	274 KB/s	4.42
		HMC	277 KB/s	13.6	292 KB/s	3.94
		Mojohost	270 KB/s	14.0	280 KB/s	3.74
	UCSB	CMich	446 KB/s	2.84	407 KB/s	1.75
		HMC	413 KB/s	3.13	475 KB/s	1.48
		Mojohost	402 KB/s	3.12	342 KB/s	2.06
	UTA	CMich	158 KB/s	10.4	168 KB/s	8.48
		HMC	183 KB/s	10.2	178 KB/s	7.95
		Mojohost	200 KB/s	8.76	209 KB/s	7.15
UTA	Wesleyan	CMich	94.7 KB/s	16.8	117 KB/s	11.3
		HMC	86.6 KB/s	19.2	115 KB/s	11.2
		Mojohost	93.7 KB/s	18.0	116 KB/s	12.2
	UCSB	CMich	1.61 MB/s	0.861	681 KB/s	0.976
		HMC	1.69 MB/s	0.793	659 KB/s	1.01
		Mojohost	1.41 MB/s	0.903	707 KB/s	0.920
	UMass	CMich	2.02 MB/s	0.750	1.25 MB/s	0.850
		HMC	3.58 MB/s	0.458	1.04 MB/s	1.02
		Mojohost	1.98 MB/s	0.678	1.03 MB/s	1.05

Table 1: Results of experiments that test our implementation of the TCP-based attack. Reading left-to-right, the first three columns denote a path, corresponding to the setup in Fig 7. The next pair of columns show, for the case where the target possesses the file, the mean throughput of the well-behaved downloader across five runs, and the relative mean throughput of the attack (for example, a value of 2.0 indicates the attack had twice the throughput of wget). The final pair of columns show the mean throughputs for the case where the target is proxying the file. The attack succeeds when the speedup for a possessed file is greater than the speedup of a proxied file (including timeouts). An asterisk indicates that all measurements timed out.

Results. Our results, shown in Table 1, show that the attack can differentiate between files served directly by the target, and files proxied by the target. Among all files made available by the target, those served at the highest rate are possessed by the target; those served at lower rates (or that result in timeouts) are proxied from a different source. The attack succeeds when the relative speedup of the attack (versus wget) is higher when the target is the source compared to when the target is the proxy; this is true in most cases even though $maxExtra$ is fixed. While preliminary, these results indicate that the attack can succeed in practice, though we do not estimate error rates for this attack in general given these limited tests.

Limitations. We note that the attack depends upon the attacker being able to force the target to serve files faster than they can be retrieved from potential sources when proxied. This assumption does not always hold. The attacker could have poor connectivity to the target; this is easily detected in advance, and attackers can acquire high-bandwidth connections. Less avoidably, there could be rate limiting or traffic shaping in place, or the target, acting as a proxy, may have an extremely high bandwidth connection to the true source. In the latter case, the machines may be co-located, which is acceptable for a search warrant, or the owner of the machine may choose to cooperate with law enforcement in investigation of a crime. This type of cooperation is typical in practice, such as when investigations of non-anonymous systems lead to an innocent owner of an open Wi-Fi base station whose neighbor is using the connection illicitly.

Defenses. To detect this attack, nodes can purposefully drop outgoing packets from their TCP stream and determine if the remote peer requests the missing data or acknowledges receiving it. There is a patch[2] to an old version of the Linux kernel that addresses this vulnerability, but any such fix results in a non-standard TCP implementation and is unlikely to be deployed on a wide scale. To defend against the attack without detecting it, OneSwarm can force a bandwidth cap on peers that can't be turned off, which isn't done currently. Since OneSwarm is an open source project, investigators will know if and when such defenses are deployed.

7. CONCLUSIONS

OneSwarm is in use by thousands of peers and is of interest to criminal and civil investigators. We have detailed three attacks on the system that are available for use by investigators using only *plain view* data. We have quantified the

[2]See `https://www.kb.cert.org/vuls/id/102014`.

precision and error rates of two of these attacks so that they are applicable in legal context. Each attack can be repeated for increased success when investigators leave and re-join the network, exposing a different set of peers.

Our novel *timing attack* is successful with only two attackers as we show by proof and experimentation with the released software on a small network. We have described a defense and shown that it places delays on OneSwarm querying that is slower than the use of Onion Routing. We have re-analyzed the collusion attack, showing the vulnerability is much greater than was previously reported, and is much worse than OR's. Our revised model accounts for probabilistic forwarding, content popularity, precision, and details from OneSwarm's source code. We show for this second attack that when investigators comprise just 25% of peers, over 40% of the network can be investigated with 80% precision to find sources of content that is not popular. For the implementation in use by thousands of people, attackers that comprise 25% of the network can successfully use this second attack against 98% of remaining peers with 95% precision. Finally, we show that OneSwarm is vulnerable to a novel application of a known TCP-based attack, which allows a single attacker to identify whether a neighbor is the source of data or a proxy for it. An implementation of the attack succeeds given that the attacker's traffic is not shaped by third-party routers. Only users that turn off the default rate limit setting are exposed.

We provided the details of our attacks and results to the OneSwarm developers in May 2011. In August 2011, they reported that the following changes to OneSwarm have been made: the default value of p is set to 0.5, unintended forwarding latency has been decreased from about 100ms to less than 10ms, and the discrepancy between text search and hash search delays have been fixed.

Acknowledgements. This work was supported in part by NSF award CNS-1018615. We thank Cpl. Robert Erdely of the Pennsylvania State Police for verifying the presence of images of child sexual exploitation on OneSwarm. We thank Elizabeth Belding, Norman Danner, Danny Krizanc, Clay Shields, and Matthew Wright for providing resources for our TCP experiments. We thank Robert Walls and Brian Lynn for their insightful comments on the work.

8. REFERENCES

[1] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-Resource Routing Attacks Against Tor. In *Proc. ACM WPES*, pages 11–20, 2007.

[2] S. Brenner, B. Carrier, and J. Henninger. The Trojan Horse Defense in Cybercrime Cases. *Santa Clara Computer and High Technology Law Journal*, 21(1), 2004.

[3] J. Chu, K. Labonte, and B. N. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *Proc. ITCom: Scalability and Traffic Control in IP Networks II Conference*, volume SPIE 4868, pages 310–321, July 2002.

[4] B. Cohen. Incentives Build Robustness in BitTorrent. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, February 2003.

[5] L. Denoyer and M. Latapy. Measurement and Analysis of P2P Activity Against Paedophile. *Journée Données et Apprentissage Artificiel (DAPA)*, March 2009.

[6] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. USENIX Security Symposium*, August 2004.

[7] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *Proc. USENIX Security Symposium*, 2009.

[8] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving P2P data sharing with OneSwarm. In *Proc. ACM SIGCOMM*, pages 111–122, August 2010.

[9] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving P2P data sharing with OneSwarm. Technical report, Dept. CS and Eng., Univ. of Washington, 2010. `http://oneswarm.cs.washington.edu/f2f_tr.pdf`.

[10] M. Latapy, C. Magnien, and R. Fournier. Quantifying paedophile activity in a large P2P system. In *IEEE Infocom Mini-Conference*. `http://antipaedo.lip6.fr`, April 2011.

[11] M. Liberatore, R. Erdely, T. Kerle, B. N. Levine, and C. Shields. Forensic Investigation of Peer-to-Peer File Sharing Networks. In *Proc. DFRWS Annual Digital Forensics Research Conference*, August 2010.

[12] M. Liberatore, B. N. Levine, and C. Shields. Strengthening Forensic Investigations of Child Pornography on P2P Networks. In *Proc. ACM Conference on Future Networking Technologies (CoNEXT)*, November 2010.

[13] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *Proc. USENIX OSDI*, pages 351–366, 2006.

[14] M. Piatek, T. Kohno, and A. Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks. In *Proc. USENIX Hot Topics in Security (HotSec)*, pages 12:1–12:7, July 2008.

[15] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network. *IEEE Internet Computing Journal*, 6(1):50–57, 2002.

[16] J. Rorher. MUTE. `http://mute-net.sourceforge.net`.

[17] RShare. `http://www.stealthnet.de/en\_index.php` and `http://www.planetpeer.de/wiki/index.php/RShare\_documentation\_\%28English\%29`.

[18] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. In *Proc. ACM SIGCOMM*, pages 71–78, October 1999.

[19] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP receivers can cause Internet-wide congestion collapse. In *Proc. ACM CCS*, pages 383–392, Oct. 2005.

[20] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Intl. Wkshp. on Designing Privacy Enhancing Technologies*, pages 96–114, July 2000.

[21] U.S. Dept. of Justice. National Strategy for Child Exploitation Prevention and Interdiction: A Report to Congress. `http://www.projectsafechildhood.gov/docs/natstrategyreport.pdf`, August 2010.

[22] U.S. General Accounting Office. File-Sharing Programs. Child Pornography Is Readily Accessible over Peer-to-Peer Networks. GAO-03-537T. Statement Before Congress; Linda D. Koontz, Information Management Issues, March 2003.

[23] U.S. Government. Federal Rules of Evidence. Rule 401. `http://www.law.cornell.edu/rules/fre/rules.htm`.

[24] R. J. Walls, B. N. Levine, M. Liberatore, and C. Shields. Effective Digital Forensics Research is Investigator-Centric. In *Proc. USENIX Workshop on Hot Topics in Security (HotSec)*, August 2011.

[25] S. Wolchok, O. Hofmann, N. Heninger, E. Felten, J. Halderman, C. Rossbach, B. Waters, and E. Witchel. Defeating Vanish with Low-Cost Sybil Attacks Against Large DHTs. In *Proc. ISOC Symposium Network and Distributed System Security (NDSS)*, February 2010.

[26] M. Wright, M. Adler, B. N. Levine, and C. Shields. An Analysis of the Degradation of Anonymous Protocols. In *Proc. ISOC Symposium Network and Distributed System Security (NDSS)*, pages 38–50, February 2002.

[27] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communication Against Passive Logging Attacks. In *Proc. IEEE Symposium on Security & Privacy*, pages 28–41, May 2003.