The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network

Rob Jansen* Florian Tschorsch‡ Aaron Johnson* Björn Scheuermann‡

* U.S. Naval Research Laboratory, Washington, DC
{rob.g.jansen, aaron.m.johnson}@nrl.navy.mil

‡ Humboldt University of Berlin, Germany
{tschorsch, scheuermann}@informatik.hu-berlin.de

***Abstract*—Tor is a distributed onion-routing network used for achieving anonymity and resisting censorship online. Because of Tor's growing popularity, it is attracting increasingly larger threats against which it was not securely designed. In this paper, we present the Sniper Attack, an extremely low cost but highly destructive denial of service attack against Tor that an adversary may use to anonymously disable *arbitrary* Tor relays. The attack utilizes *valid protocol messages* to boundlessly consume memory by exploiting Tor's end-to-end reliable data transport. We design and evaluate a prototype of the attack to show its feasibility and efficiency: our experiments show that an adversary may consume a victim relay's memory by as much as 2187 KiB/s while using at most only 92 KiB/s of upstream bandwidth. We extend our experimental results to estimate the threat against the live Tor network and find that a strategic adversary could disable all of the top 20 exit relays in only 29 minutes, thereby reducing Tor's bandwidth capacity by 35 percent. We also show how the attack enables the deanonymization of hidden services through selective denial of service by forcing them to choose guard nodes in control of the adversary. Finally, we discuss defenses against the Sniper Attack that provably render the attack ineffective, and suggest defenses against deanonymization by denial-of-service attacks in general that significantly mitigate the threat.**

I. Introduction

Large scale Internet censorship by state-level authorities [44] has spurred the development of new privacy enhancing technologies that circumvent the censor, followed by new techniques to recognize and block these circumvention tools [22]. As this censorship arms race proceeds, more resilient circumvention technologies will be developed in order to increase the cost of detection using traditional methods. We argue that as these circumvention technologies improve and the cost of detection increases, alternative techniques for disruption will become increasingly viable. As such, understanding these alternatives is paramount not only to the successful design of future technologies, but also to the security of existing networks and systems.

Tor [19] is the most popular deployed system for fighting censorship and online privacy encroachments, currently supporting several hundreds of thousands of users daily and transferring roughly 3 GiB/s in aggregate [8]. Tor uses onion routing [25] to route *clients'* traffic through a *circuit* of geographically diverse *relays*, preventing any single relay from linking the client to its Internet destination. This work focuses on Tor due to its practical relevance as an adversarial target.

In this paper, we present, analyze, and evaluate a novel and destructive denial of service (DoS) attack against Tor that may be used to *anonymously and selectively disable arbitrary Tor relays* with very low cost to the attacker: our attack is efficient to the extent that an adversary interested in censorship could *disable* instead of *block* Tor by simply disabling all relays or intelligently targeting crucial subsets of relays, e.g., those providing high network throughput or authoritative directory services. The attack may be undetectably carried out on any machine with moderate computational and memory resources and presents severe security implications for Tor and its users. In addition to threatening network availability, we show how the attack can be used to deanonymize hidden services by selectively disabling relays, heavily influencing paths to those in control of the adversary. Our attack thus imposes real, significant threats to Tor's users,[1] and we believe it constitutes the most devastating attack against the Tor network to date.

The attack, which we call the *Sniper Attack* since the attacker remains hidden while disabling relays in a targeted manner, works by utilizing Tor's application level congestion and flow control mechanisms to cause a Tor relay to buffer an arbitrary amount of data in application queues. In particular, an adversarial client builds a normal Tor circuit using the target relay as the entry, commands the exit to start downloading a large file through the circuit, and then continuously sends `SENDME` cells to the exit *without reading from the target entry*. The `SENDME` cells signal the exit to increase its congestion windows, after which it will continue to pull data from the external data source and push it into the circuit. This process may be repeated in parallel on many circuits using the same target entry for each. The remote Tor process on the target relay will queue the data and eventually exhaust its host's memory, resulting in termination by its operating system's memory manager (e.g. the oom-killer on Linux [1]).

Using Shadow [28], we demonstrate the destructiveness of the Sniper Attack and the effectiveness of our defenses by evaluating them in a safe, private, simulated Tor network. The evaluation of our attack prototype indicates that an adversary may consume a target relay's memory by as

NDSS '14, 23-26 February 2014, San Diego, CA, USA
 ISBN 1-891562-35-5
http://dx.doi.org/10.14722/ndss.2014.23288

[1] We disclosed our attack to The Tor Project [6] in February 2013. We have worked with them to develop and deploy a short term defense [17], and continue to work with them in developing long term solutions [32]. As a result, Tor is no longer vulnerable since version 0.2.4.14-alpha.

much as 2187 KiB/s (903 KiB/s in the median), while the adversarial bandwidth costs are at most 92 KiB/s upstream and 39 KiB/s downstream, (46 KiB/s upstream and 14 KiB/s downstream in the medians). Using these results, we estimate that sequentially disabling each of the fastest 20 exit relays takes a cumulative total of only 29 minutes. In addition, we explore using Tor to anonymously disable relays by utilizing a separate anonymous tunnel through which we launch our attacks, and find that doing so does not increase the adversarial bandwidth requirements.

We analyze the security threat that the Sniper Attack poses and present novel techniques for deanonymizing hidden services. We utilize the Sniper Attack's ability to kill arbitrary relays in a selective denial of service attack against the guard relays of hidden services, influencing the paths chosen by the hidden services to those in control of the adversary. We find that it enables the complete deanonymization of hidden services within days by an adversary with only modest resources or within hours by a more powerful adversary.

This paper also explores defense strategies against the Sniper Attack. We discuss how simple hard-coded queue size limits and end-to-end authenticated signals affect the adversary's attack strategy, but do not completely prevent the attack. We then present an algorithm that adaptively reacts to high memory pressure indicative of the attack. Our adaptive defense utilizes queuing delay as a metric to identify and kill malicious circuits in order to prevent the process from being killed. We derive resource bounds with our defense mechanism in place, showing that it cannot reasonably be leveraged by attackers to cause relays to destroy honest clients' circuits. Our evaluation shows that our adaptive circuit killing defense detects and stops the Sniper Attack with no false positives.

Finally, we present and analyze path restrictions that mitigate the threat of DoS deanonymization. By restricting the relays it uses for sensitive circuit positions, a client will *fail closed* to an unavailable but safe state instead of an avaliable but potentially compromised one. We analyze the security and availability cost of such changes under a variety of parameters and find acceptable security/availability trade-offs.

Our main contributions may be summarized as follows:

- a dangerous and destructive DoS attack capable of disabling arbitrary Tor relays (Section II);
- an evaluation of a prototype of the attack and our defenses in a safe, virtual Tor network (Section III);
- a security analysis showing how the attack may be used to deanonymize hidden services (Section IV);
- practical defenses against the Sniper Attack that reduce Tor's vulnerability to attacks that exploit Tor's queuing mechanisms (Section V); and
- practical defenses against DoS-based deanonymization attacks that improve security by limiting network exposure (Section VI).

II. The Sniper Attack

In this section, we develop a DoS attack against the Tor network that can be used to anonymously disable arbitrary Tor relays by killing the Tor process on its host machine. To facilitate an understanding of the exploited protocol features, we first describe two basic attack variants that require the adversary to run both a Tor client and either a Tor exit relay or an Internet service. We then describe a more efficient variant that only requires a Tor client and therefore significantly reduces the resources required by the adversary. Finally, we discuss strategies that disguise the adversary's identity.

A. Background

Tor is an application-level overlay network enabling anonymous communication between clients and arbitrary Internet destinations. Tor clients are responsible for path selection at the overlay layer, and form virtual *circuits* through the overlay network by selecting three relays from a public list for each: an entry; a middle; and an exit. Once a circuit is established, the client creates *streams* through the circuit by instructing the exit to connect to the desired external Internet destinations. Each pair of relays communicate over a single onion routing *connection* that is built using the Transmission Control Protocol (TCP). The application layer protocols rely on this underlying TCP connection to guarantee reliability and in-order delivery of application data, called *cells*, between each relay. As a result of using hop-by-hop TCP at the network layer, Tor does not allow relays to drop or re-order cells at the application layer. Streams are multiplexed over circuits, which themselves are multiplexed over connections.

Tor implements an end-to-end sliding window mechanism to control the amount of data directed into the network. For every circuit, each edge node (i.e. client and exit) manages a *package* window counter that is initialized to 1000 and decremented by one for every data cell it directs *into* the circuit, and a *delivery* window counter that is initialized to 100 and decremented by one for every data cell it *removes* from the circuit. Analogous counters also exist at the stream level, respectively initialized to 500 and 50. The packaging edge (PE) of a circuit will stop injecting cells from any multiplexed stream whose package window reaches zero, and will stop injecting cells from all multiplexed streams when the circuit packaging window reaches zero. The delivery edge (DE) of a circuit will send a feedback signal, called a `SENDME` cell, to the PE whenever the circuit delivery window or any stream delivery window reaches zero. These `SENDME` cells cause the DE to increment the associated delivery window by its initialized value, and the PE to increment its packaging window by the same amount.[2] Thus, there will not be more than 500 data cells in flight on a stream, and not more than 1000 on a circuit.

B. Basic Attacks

The Sniper Attack exploits Tor's reliable application-level queuing. Our assertion is that a DE that *stops reading* from a connection will cause the next hop node to buffer a full package window worth of data (1000 cells) from the PE for every active circuit multiplexed over the connection, under the assumptions that there are at least two streams multiplexed on each circuit and that the streams transfer enough data in aggregate to reduce the PE's circuit package window to zero. When a DE with incoming data stops reading from its TCP socket on the connection to an adjacent relay, the DE's TCP receive buffer will fill, its TCP flow control window will empty, and it will announce a zero window to the other end of the TCP connection. The adjacent relay will then no longer be able to forward cells to the DE, causing its TCP send buffer

[2] In practice, circuit and stream delivery windows are respectively initialized to 1000 and 500. When they reach 900 and 450, `SENDME`s are sent and they are incremented by 100 and 50. Therefore, the delivery windows will not fall below 900 and 450 under normal operation.

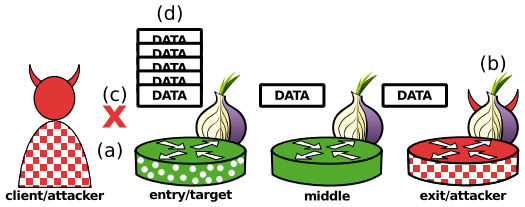


Fig. 1: In the basic version 1 of the Sniper Attack, the adversary controls the client and the exit. (a) The client creates a circuit using the target as the entry. (b) The exit generates, packages, and sends data through the circuit, ignoring package window limits. (c) The client stops reading from the TCP connection to the target entry. (d) The target entry buffers the data until the Tor process is terminated by the OS.

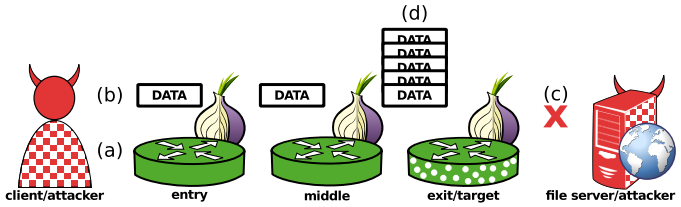


Fig. 2: In the basic version 2 of the Sniper Attack, the adversary controls the client and the server. (a) The client creates a circuit using the target as the exit, and connects to a colluding server. (b) The client generates, packages, and sends data through the circuit, ignoring package window limits. (c) The server stops reading from the TCP connection to the target exit. (d) The target exit buffers the data until the Tor process is terminated by the OS.

to fill. With a full TCP send buffer, the adjacent relay will buffer cells in the application layer circuit queue (recall that Tor does not allow relays to drop cells in the application layer) until the PE's stream or circuit package window reaches zero. The PE will then stop sending data into the circuit, and stop reading from the data source.

Using the mechanism described above, an adversary that controls a client and a relay may attack a target relay as shown in Figure 1. The adversarial client constructs a circuit by selecting the target relay as the entry and the adversarial relay as the exit.[3] The client signals the exit to start the attack by issuing an arbitrary request over the custom attack circuit, and then stops reading from the TCP connection to the target entry. The exit simply ignores the empty package windows and continuously sends data it arbitrarily generates, increasing the amount of memory consumed by the entry to queue the cells. Note that it is not necessary for the malicious exit to produce correctly encrypted Tor cells since they will never be fully decrypted by the client (though correct circuit IDs are required). Eventually, the Tor process on the entry node depletes all of the available memory resources and is terminated by the operating system. On Linux systems, this job is handled by the out-of-memory (oom) killer [1].

A variation of the basic attack described above is shown in Figure 2. In this variant, the adversary controls a client and a file server. The client generates arbitrary data and packages it for delivery to the target exit. The adversarial server avoids reading from the TCP connection to the target exit, again resulting in memory exhaustion and death of the Tor process on the target relay's host machine. Note that the cells must be encrypted in this attack variant because they will be decrypted by a machine which is not under the adversary's control.

Note that the adversary may choose any relay as its target entry in version 1 of the basic attack, and should choose the file server's port according to the exit relay's exit policy in version 2. However, choosing relays without the `Guard` flag for a circuit's entry position will raise suspicion since Tor's default path selection algorithm will not choose entries in that manner. Alternatively, basic versions 1 and 2 may be slightly modified to target any middle node: in version 1 the adversary may additionally run an adversarial entry relay that stops reading from the connection to a target middle relay; in version 2 the adversary may run an adversarial exit that stops reading from the connection to a target middle relay instead of running an external file server.

[3]The Tor software provides parameters, `EntryNodes` and `ExitNodes`, to specify a list of nodes for the respective roles; one could also use the Tor control protocol [4] to build custom circuits.

We assert that the TCP connection from the client to the target must remain open from the victim's perspective to prevent the attack circuit from being closed and its queue cleared, but the cost of doing so is insignificant (and it can be done without maintaining state [26]). Also, the adversary may slightly reduce the required bandwidth by minimizing the size of its TCP receive buffer, e.g., by using `setsockopt`.

C. *Efficient Attack*

We now describe an efficient Sniper Attack that eliminates the necessity of generating and uploading data, thereby significantly reducing resource demands. This efficient version of the Sniper Attack exploits Tor's end-to-end flow control signals. Our assertion is that the `SENDME` flow signals expected by the PE (so that it may continue packaging data and sending it into the circuit) only *imply* that a DE received data and a DE may send `SENDME`s to the PE without actually receiving any data.

The efficient sniper attack works by combining the *`SENDME` signal* mechanism described above with the *stop reading* mechanism from the basic versions of the attack. As shown in Figure 3, the adversary must only control a single malicious client. This client first builds a custom circuit by selecting the target as the circuit entry, and then initiates the download of two large files (e.g., large Linux distributions) over the circuit to ensure that the two streams will empty the exit's circuit package window. The client then stops reading from the connection to the target entry, and begins maliciously sending `SENDME`s to the exit to ensure that the exit's package window does not reach zero and it continues injecting packaged data into the circuit. These packaged cells will continue to flow to and be buffered by the entry in its application queue, continuously consuming memory until the entry's Tor process is selected and killed by the OS.

1) Avoiding Detection: To launch a successful Sniper Attack, the adversary must circumvent a protective mechanism that Tor employs to prevent protocol violations, e.g., by clients who try to cheat by sending more `SENDME` cells to get more data earlier. When the exit relay receives a `SENDME` that causes its circuit window to go above 1000 cells, it detects the violation, closes the circuit, and sends a `DESTROY` cell backwards. The middle hop converts the link-level `DESTROY` cell into a `RELAY` cell of type *truncate* and sends it to the entry, who just passes it back to the client. When the client extracts the `DESTROY` cell (that originated at the exit) from the `RELAY` cell, it closes the circuit and sends a `DESTROY` cell forward to the entry. The entry closes the circuit (clearing the circuit queue) and forwards the `DESTROY` cell to the middle, who also closes the circuit.

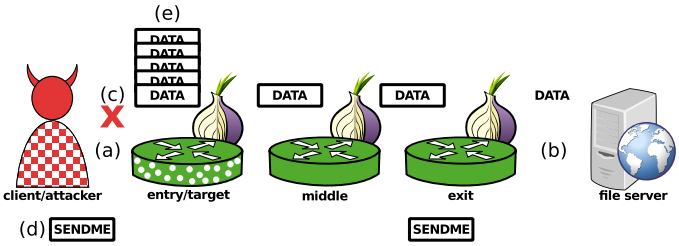


Fig. 3: In the efficient version of the Sniper Attack, the adversary controls a client only and repeats the following several times in parallel. (a) The client creates a circuit using the target as the entry. (b) The client initiates a large file download from an external file server through the circuit. (c) The client stops reading from the TCP connection to the target entry. (d) The client sends SENDME cells to the exit, causing it to continue sending data through the circuit. The rate of SENDMEs is low enough to avoid exceeding the exit's package window size. (e) The target entry buffers the data until the Tor process is terminated by the OS.

In order for the attack to succeed, the adversary ideally would (a) prevent the exit's package window length from exceeding its size; and (b) in case it does, the client would avoid sending out the final DESTROY cell to ensure the entry does not clear its queue. Note that since the malicious client will not be reading from the target entry, the adversary will not be able to determine if (a) occurred, and therefore does not need to handle (b) in practice. However, we note it here for completeness. Also note that, as will be discussed in the next section, even if the adversary fails at (a) and the exit detects a protocol violation, the attack circuit will continue to consume the target's memory until the TCP connection is destroyed.

The adversary may avoid the exit's protective mechanism by sending SENDMEs to the exit at a rate low enough so that the exit's package window never exceeds 1000 cells. One approach to estimating such a rate is to consult the Tor metrics portal [8] and use recent relay byte histories to estimate the throughput of the custom circuit. However, given the dynamics of the Tor network and its usage, this approach would likely result in a high failure rate. Instead, a malicious client may account for real time congestion by performing file download probes through the same nodes that were chosen for the target circuit. If each probe downloads σ KiB in Δ seconds, then we can estimate the circuit throughput as σ/Δ KiB/s, or $2\sigma/\Delta$ cells/s (all Tor cells are 512 bytes in size). Now recall that stream and circuit level SENDMEs are sent for each 50 and 100 downloaded cells, respectively. Thus, using our probe we estimate that stream and circuit level SENDMEs be sent every $T_{ss} = 25\Delta/\sigma$ seconds and $T_{cs} = 50\Delta/\sigma$ seconds, respectively. The malicious client may update Δ by periodically performing an additional probe, and larger values of σ are more costly but will produce more accurate estimates over time. Probing requires additional adversarial bandwidth, but this cost may be significantly reduced.

2) Parallelizing the Attack: Recall that the exit will close a circuit if the package window exceeds its size, and this circuit closure will be undetectable by the client once it stops reading from the target entry. Although a circuit closed by the exit will not cause the target entry to clear its application queue (and therefore free any memory consumed by that circuit), the circuit may no longer be utilized to *increase* memory consumed by the target entry. This situation may occur even if the adversary probes the circuit to find a good SENDME rate, since relay congestion and path throughput are highly dynamic.

To improve the attack's resilience to circuit closures while at the same time speeding up the rate at which the target's memory is consumed, the adversary may parallelize the attack by using multiple *teams* of multiple circuits. One circuit in each team is assigned throughput probing duties (in order to measure Δ as described in the previous section), while the remaining circuits are assigned SENDME sending duties (to cause the exit to push data toward the target). The Δ computed by a team's probing circuit is used to dynamically inform the rate at which that team's sending circuits send SENDMEs. Each team is assigned a Tor path using the target as the entry relay and uses that path to build each of its circuits.

We now consider how these circuits are constructed. Recall that once the attack begins and the adversary has stopped reading from the onion-routing TCP connection to the target, it will be unable to determine which circuits on that connection have closed and which ones have not, and will also be unable to create new circuits over that connection. Since a separate connection is required for the probing circuits (because it must communicate bi-directionally), the adversary will need at least two connections to the entry for each team if the attack is to be successful. With this in mind, we consider three viable attack strategies: 1) use one Tor client instance for each circuit of each team; 2) use one Tor client instance per team that creates a new onion-routing connection to the target whenever one is needed; and 3) use two Tor client instances per team: one that controls the probing circuit and one that controls the sending circuits. Note that unique onion-routing connections are guaranteed by using separate Tor client instances. Although each of the above strategies are viable, we reject 1) because there is a high resource cost associated with running many Tor instances, and we reject 2) because multiple connections from a single Tor client instance would be easy for the entry to detect and would require significant code changes. Therefore, we assume the adversary uses strategy 3) where all circuits are operating in parallel.

The use of multiple circuits within each team will increase the throughput acheived by that team from its assigned path due to the circuit scheduling policies employed at each relay and will prevent a single sending circuit failure from stalling the attack. Using a consistent path within each team ensures that the sending rate Δ is accurate for all of that team's members. Assigning middle and exit relays indepently for each team further utilizes Tor's distributed resources by reducing the effect of throughput bottlenecks while also increasing the robustness to node failures. Finally, as there is no circuit feedback, the adversary may also pause the attack on existing teams and rotate to new ones over time to ensure that the target entry's memory consumption continues to increase.

3) Hiding the Sniper: For simplicity, we have thus far discussed the Sniper Attack as if the adversary is directly connecting to the target entry. Here, $\mathcal{C}$ denotes client, $\mathcal{G}$ denotes entry, $\mathcal{M}$ denotes middle, $\mathcal{E}$ denotes exit, and $\mathcal{S}$ denotes server, while the subscripts A and V denote adversary and victim, respectively. The path of the attack as previously described may then be represented as:

$$\mathcal{C}_A \leftrightarrow \mathcal{G}_V \leftrightarrow \mathcal{M} \leftrightarrow \mathcal{E} \leftrightarrow \mathcal{S}$$

In this situation, the victim $\mathcal{G}_V$ knows the adversary $\mathcal{C}_A$'s IP address since they are directly connected. $\mathcal{G}_V$ may have enough information to blame $\mathcal{C}_A$, either during or after the attack, because of the anomalous behavior. Extra protections may be desired to avoid this exposure.

a) Stealth with Tor: Tor itself is a useful tool to provide such protections. One way the adversary could use Tor is by also running a Tor exit node:

$$\mathcal{E}_A\mathcal{C}_A \leftrightarrow \mathcal{G}_V \leftrightarrow \mathcal{M} \leftrightarrow \mathcal{E} \leftrightarrow \mathcal{S}$$

This situation provides the adversary *plausible deniability*: $\mathcal{G}_V$ will not be able to distinguish an attack by $\mathcal{C}_A$ from one launched through a circuit in which $\mathcal{E}_A$ is merely serving as the exit.[4] However, drawbacks to this approach are that $\mathcal{E}_A$ will need to serve as an honest exit, which consumes far more resources than required by the attack and also results in the adversary appearing in the public Tor directory. The adversary then has to ensure that $\mathcal{E}_A$ has the characteristics of other honest exits (has the right consensus flags for its activities, has the right amount of traffic for its consensus weight, etc). Further, $\mathcal{G}_V$ will still know the IP address and may use it as a starting point when looking for someone to blame.

Alternatively, the adversary may use a full Tor circuit:

$$\mathcal{C}_A^2\mathcal{C}_A^1 \Leftrightarrow \mathcal{G}^1 \Leftrightarrow \mathcal{M}^1 \Leftrightarrow \mathcal{E}^1 \leftrightarrow \mathcal{G}_V^2 \leftrightarrow \mathcal{M}^2 \leftrightarrow \mathcal{E}^2 \leftrightarrow \mathcal{S}$$

This provides the adversary *anonymity*. It will prevent A's IP address from being known by anyone except $\mathcal{G}^1$, who will be oblivious to the attack. In this scenario, $\mathcal{C}_A^1$ stops reading on the connection to $\mathcal{G}^1$ but $\mathcal{C}_A^2$ sends `SENDME`s to $\mathcal{E}^2$ through the $\mathcal{C}_A^1$ proxy tunnel. A drawback to using a separate circuit in this way is that it may slightly increase the latency and length of the attack, because $\mathcal{G}_V^2$ will not start depleting its memory resources until $\mathcal{E}^1$'s package window reaches zero. It may also be more difficult to estimate a good `SENDME` rate when concatenating two circuits, and the adversary must now run twice as many Tor client instances to ensure that each team has two anonymous tunnels. Finally, a circuit that exits back into Tor may draw unwanted suspicion.

b) Stealth without Tor: Alternatives to using Tor to hide include using public open wireless access points, briefly renting a small botnet, or using a cloud computing system. However, more entities will then know about the adversary's actions, increasing the risk of discovery: access points and cloud services will be collecting logs; and some number of bots could be part of a honeypot. The adversary may want to connect to these services through Tor anyway to remain anonymous to them, and the composition of services will make it easier to make a mistake. By using Tor as described above, the adversary does not need knowledge of botnets or cloud systems, drastically simplifying the attack.

III. Evaluation

We implemented a prototype of the Sniper Attack in order to evaluate its feasibility and efficacy. We evaluated it using Shadow [28], a discrete event network simulator that runs Tor code in a private Tor network, after testing its functionality in a minimal private Tor network in our lab. Shadow enables a *safe* development and evaluation environment that does not harm the security and privacy of the operational Tor network or its users, while also providing *realistic* results since it runs authentic Tor code. In this section, we detail our private Tor network configuration, describe our prototype implementation, evaluate the attack's efficiency and resource costs, and analyze our results in the context of the live Tor network.

A. Private Tor Network

Tor nodes running in Shadow communicate over a simulated network. Therefore, Shadow requires models of downstream and upstream node bandwidths as well as link latency, jitter, and packet loss rates. The Shadow distribution [3] includes these models, and also includes tools to generate private Tor network configurations for running Shadow simulations. Using these tools and real network data published by Tor[5] [8], we configure a private Tor network consisting of 4 directory authorities, 400 relays, 500 file servers, and 2800 clients. This private network consumes roughly 60 GiB of memory on our Linux host during each experiment. The clients generate background traffic during the experiments by downloading variously sized files from the servers through our private Tor, causing congestion and performance characteristics indicative of conditions in the live Tor network. All of these nodes run in the Shadow simulator and communicate only with one another. Our configuration follows the methodologies from recently published and validated research on modeling private Tor networks [27], which describes in detail the modeling choices made by Shadow's configuration generation tool.

B. The Sniper Attack Prototype

We implemented the parallel version of the efficient Sniper Attack as described in Section II-C, including multiple parallel circuits but without the rotating circuits enhancement. In our `C` prototype implementation, a *manager* manages all *workers*, each of which use the Tor control protocol [4] to command and control the associated Tor client instance and its circuits. The workers run a modified Tor client instance, based on stable release `0.2.3.25`, that adds: a `STOPREADING` controller command which instructs Tor to stop reading from the onion routing connection to the target; `SENDSTREAMSENDME` and `SENDCIRCUITSENDME` commands which instructs Tor to send a stream-level and circuit-level `SENDME`s on the specified streams and circuits; and an `IGNOREPACKAGEWINDOW` command that instructs the client to ignore package windows when sending data upstream.

We implemented both *direct* and *anonymous* Sniper Attack modes. In *direct* mode, each worker connects to the Tor client over the controller port, waits for it to become fully bootstrapped into the Tor network, and builds its custom Tor circuits using the same path as the other workers on its team. Once the attack circuits are ready, the probing workers begin circuit measurement probes by downloading files through their attack circuit; the remaining workers request an extremely large file through the attack circuit, command Tor to stop reading, and send two stream `SENDME`s and one circuit `SENDME` for every completed probe download. In *anonymous* mode (see Section II-C3a), each worker runs two Tor client instances instead of one: the first is used to create an anonymous tunnel through Tor; the second is used as in direct mode, except that all communication with relays is done over the anonymous tunnel using the `Socks4Proxy` Tor configuration option. Note that the client instances that create the anonymous tunnels ignore their package windows using the `IGNOREPACKAGEWINDOW` command, because otherwise

[4] $\mathcal{G}_V$ can distinguish `CREATE` cells from `EXTEND` cells, but it is plausible that a `CREATE` cell originated from some client in a separate circuit terminating at $\mathcal{E}_A$ rather than from $\mathcal{C}_A$, e.g., if that client is using Tor's `Socks4Proxy` or `Socks5Proxy` options.

[5] We use the server descriptors and extra info documents from 2013-06, and the Tor consensus from 2013-06-30-23-00-00

TABLE I: Sniper Resource Usage

Config.	Max RAM (MiB) and Mean BW (KiB/s)					
	Direct			Anonymous		
	RAM	Tx	Rx	RAM	Tx	Rx
10 Teams, 100 Circs	283	56.0	20.9	564	56.6	17.0
5 Teams, 50 Circs	141	30.0	9.5	283	27.7	8.5
1 Teams, 10 Circs	28	6.1	2.6	57	9.4	2.1
1 Team, 5 Circs	28	4.0	2.3	56	3.6	1.8

the SENDMEs that are being forwarded from the attack circuits upstream through the tunnel will evetually drain the windows and stall the attack (the tunnel's normal downstream SENDMEs which increment the package window will not be received because of the stop reading attack behavior). The sniper manager and worker logic was packaged as a Shadow plug-in consisting of 1416 lines of code, while our Tor modifications included 253 lines of code.

C. Experiments and Results

We experiment with our prototype implementation of the Sniper Attack to explore the target memory consumption and sniper resource tradeoffs when conducting the attack against target relays of various capacities. Our Tor network model is configured as described above, with the addition of an adversarial sniper node that runs the Tor clients and the sniper manager that controls the attack. Unless otherwise specified, our experiments use 100 circuits configured as 10 teams of 10 circuits each, while each probing circuit downloads σ=50 KiB files, pausing for 60 seconds between each probe. Every team uses a unique Tor path for their circuits chosen following Tor's weighted path selection algorithm. Our sniper is configured with a 100 MiB/s symmetric bandwidth access link so as not to result in a bandwidth bottleneck during the experiment for measurement purposes. Each experiment runs for 60 virtual minutes, during the first 30 of which we allow the network to bootstrap and during the last 30 of which the attack is executed. We run one attack against a single target relay in each experiment, and measure the RAM used over time by the target and the sniper as well as the bandwidth consumed by the sniper.

We tested the feasibility of the Sniper Attack, arbitrarily choosing the highest weighted non-authoritative, non-exit guard node in our network as the target. This node had a 9 MiB/s symmetric bandwidth access link and otherwise served as an ordinary relay in our experiment. We tested the Sniper Attack with each of 100, 50, 10, and 5 attack circuits. As can be seen in Figure 4a, the number of circuits directly affects the rate at which the sniper is able to consume the target's memory. While the target's memory consumed in each scenario increases approximately linearly, there is a dramatic difference between 10 and 50 circuits: the 10 circuit experiment was configured with 1 team, meaning that all 10 circuits are configured with the same path through Tor; the 50 circuit experiment was configured with 5 teams, meaning that there are 5 paths chosen through Tor. Choosing multiple paths in this way more effectively utilizes Tor's capacity and prevents the attack from stalling due to a poorly chosen circuit that may contain a tight bottleneck.

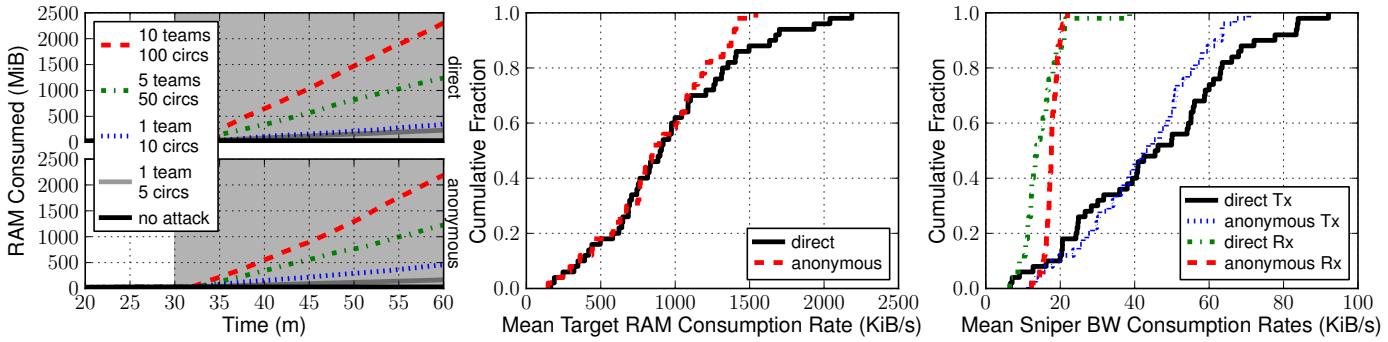
The memory and bandwidth requirements for the sniper in our feasibility experiments can be seen in Table I. Shown are the maximum total memory (RAM, in MiB) used by the sniper at any point during the attacks and the mean total bandwidth (BW, in KiB/s) consumption, for both the *direct* and *anonymous* experiments. The RAM used by the sniper depends almost entirely on the number of Tor instances being used: in all cases, the mean RAM used per Tor client instance was approximately 14 MiB. As expected, the *anonymous* attack consumes roughly twice as much memory as the *direct* attack since it is using twice as many Tor client instances. The resource requirements for our prototype are quite reasonable: the maximum memory required in any of our experiments was less than 600 MiB and the maximum upstream and downstream bandwidth required was 56.6 and 20.9 KiB/s, respectively. Further, the sniper's 60 second bandwidth burst remained below 500 KiB/s throughout the experiment. We expect an adversary willing to launch this type of attack can easily satisfy these requirements. Note that probing less often may further reduce the bandwidth costs.

Our feasibility experiments tested the Sniper Attack against an arbitrarily chosen relay. We expanded this evaluation to determine how the Sniper Attack performs against a variety of relays with unique congestion, load, and bandwidth capacities. To do this, we chose a set of 50 relays from our private network, again using Tor's weighted path selection algorithm. Using the default settings outlined above, we ran our prototype Sniper Attack against each relay twice: once in *direct* mode and once in *anonymous* mode. We measured the memory consumed by the target and the bandwidth consumed by the sniper during each experiment.

We computed the mean target memory consumption rate and mean sniper bandwidth consumption rate achieved during each experiment (recall that each experiment targets a different relay). Figures 4b and 4c show the cumulative distribution of these rates for each mode over the 50 experiments; each experiment produces one data point in each of the two figures. As shown in Figure 4b, the median computed mean target RAM consumption rate was 903.3 KiB/s in the *direct* attack and 849.9 KiB/s in the *anonymous* attack. Further, the *direct* mode of the attack was only slightly more effective in our experiments: in the maximum the sniper was able to consume the target's RAM at 2186.8 KiB/s, roughly 1.4 times as fast as the maximum of 1541.8 KiB/s in *anonymous* mode. Although this difference is only seen in the tail of the CDF, the reason is likely due to the additional length of the attack circuit path in *anonymous* mode (the cells must traverse 6 Tor relays in this case), which may lead to less accurate probing and extra latency when sending the SENDME cells through the anonymous Tor tunnel to the the opposite edge of the attack circuit. Further, the longer path increases the chance that a bottleneck exists on the path which may cause some of the attack circuits to fail. Figure 4c shows that the bandwidth requirements in both modes are similar: the mean upstream bandwidth measured was 45.9 and 43.0 KiB/s in the median for the *direct* and *anonymous* attacks, while the mean downstream bandwidth was respectively 13.6 and 17.6 KiB/s in the median. Our experiments show that the Sniper Attack enables the adversary to relatively easily trade its bandwidth resources for a victim relay's memory resources.

D. Analysis

We now analyze the practical effect the Sniper Attack has on the operational Tor network by considering how realistic adversaries might choose to disable relays. The adversary may prioritize as targets the relays with low RAM but high



(a) Target Memory over Time (b) Target Memory Consumption (c) Sniper Bandwidth Consumption

Fig. 4: The Sniper Attack resource consumption. Shown in (a) is the target relay's memory usage over time in *direct* and *anonymous* attack modes. Compared are attacks with 1 team of 5 and 10 circuits, 5 teams of 10 circuits each (50 circuits total), 10 teams of 10 circuits each (100 circuits total), and no attack. The shaded area indicates the time during which the attack is active. Shown in (b) and (c) are the distributions of the mean consumption rate of the target relay's RAM per experiment and mean sniper's bandwidth cost per experiment, respectively, over 50 experiments each of *direct* and *anonymous* Sniper Attacks. The sniper in each experiment is configured to use 10 teams of 10 circuits each (100 circuits total).

consensus weights: this will have the largest impact on users since Tor's load balancing algorithm is tuned so that the probability that a client chooses a relay is in proportion to the bandwidth capacity that relay contributes to the network. However, since relay memory resources are not public, we consider an adversary that chooses relays based on the consensus weight alone and explore the time to disable them according to various potential memory configurations. Because of the load balancing algorithm and the fact that currently the relays with the top 100 weights constitute 40 percent of the selection probability, the adversary may have significant impact on the network by disabling a relatively small group of relays.

We utilize the results from our 100 experiments discussed above to estimate memory consumption rates that an adversary may achieve on live Tor network relays. To do this, we compute the correlation between the observed mean memory consumption rate of each target relay in our experiments and that relay's consensus weight using a linear regression. This results in parameters that we use to estimate the memory consumption rate of any relay for which we have a consensus weight. Negative rate estimates were replaced with the minimum observed rate. We then use these rates to compute the time to disable various groups of relays: we consider the top, median, and bottom guard and exit relay by the probability of selection by clients out of those with the `FAST` flag, as relays without the `FAST` flag are only selected if no `FAST` relays are available. We also consider the top 5 and 20 of both guards and exits as those relays will be selected most often by clients and represent the most attractive targets for the adversary. We consider the 10 directory authorities as the final group, as the network will not function over time without the authoritative documents they collectively produce and distribute.

Shown in Table II is the total selection probability for each relay group, and the estimated total length of time to disable all relays in the group when the Sniper Attack is synchronously launched on a single relay at a time. We consider memory consumption rates for both *direct* and *anonymous* attacks, and consider the length of time to disable relays with 1 and 8 GiB of RAM as examples of relay memory capacities. Note that these results scale linearly to other RAM sizes. Also note that

TABLE II: Combined Path Selection Probability of and Expected Time to Disable Selected Groups of Relays

			Time (H:M) to Consume RAM			
			Direct		Anonymous	
		Sel %	1 GiB	8 GiB	1 GiB	8 GiB
Relay Groups	Top `FAST` Guard	1.7	0:01	0:18	0:02	0:14
	Median `FAST` Guard	0.025	0:23	3:07	0:23	3:07
	Bottom `FAST` Guard	1.9e-4	1:45	14:03	1:45	13:58
	Top `FAST` Exit	3.2	0:01	0:08	0:01	0:12
	Median `FAST` Exit	0.01	1:45	14:03	1:22	10:53
	Bottom `FAST` Exit	6e-5	1:45	14:03	1:48	14:20
	Top 5 Guards	6.5	0:08	1:03	0:12	1:37
	Top 20 Guards	19	0:45	5:58	1:07	8:56
	Top 5 Exits	13	0:05	0:37	0:07	0:57
	Top 20 Exits	35	0:29	3:50	0:44	5:52
	All Dir Auths	N/A	17:34	140:32	17:44	141:49

although the linear regression did not result in a strong correlation (*direct*: r^2=0.164, *anonymous*: r^2=0.237), we believe it provides a reasonable prediction of RAM consumption for analysis purposes as we expect the actual time to disable the groups of relays given in Table II to fall somewhere the times given in the 1 GiB and 8 GiB columns.

Our analysis shows that the fastest guard and fastest exit with 1 GiB of RAM can be disabled in just one minute when using the *direct* attack, thereby disabling an expected 1.7 and 3.5 percent of paths in the Tor network, respectively. When allotting 8 GiB of RAM for these relays, they can be disabled in under 20 minutes in both attack modes. Perhaps more strikingly, the entire group of the fastest 20 exits can be disabled in just 29 minutes if each relay has only 1 GiB of RAM, and in just under 4 hours if each relay has 8 GiB of RAM. (The *anonymous* attack takes slightly longer in both cases.) This would be extremely disruptful to the Tor network, causing roughly 35 percent of all paths to fail and increasing load and congestion on the remaining relays. Similarly, the group of the fastest 20 guards can be disabled in just 45 minutes if allotting 1 GiB of RAM for each relay, and just under 6 hours if allotting 8 GiB of RAM for each (again, the *anonymous* attack takes slightly longer). This would cause 19 percent of Tor paths to fail. Finally, the attack takes significantly longer on the group of directory authorities, since their lower bandwidth weights result in lower RAM

consumption rates than the fastest relay groups. Note that relays will likely be rebooted by their operators some time after going down, however, all circuits they were carrying will be lost and the attack could be relaunched against a relay as soon it is available. This may effectively cause a relay to be marked as unstable and not chosen by clients for their circuits.

IV. Deanonymization in Tor

The Sniper Attack is more than just a threat to Tor's availability: it can also be used as an attack on *anonymity*. Because Tor accepts any willing relay into the network, an adversary that runs relays can deanonymize a victim by controlling the entry and exit relays and correlating the observed timing and volume of a user's traffic entering the network with that leaving the network shortly afterwards [11], [42].

To prevent an adversary running relays from eventually being chosen for these positions, a user chooses a small set of *entry guards* (Tor defaults to 3 guards), and begins all circuits at one of these guards. This protects the user from being directly observed as long as adversarial relays are not chosen as guards. A guard is used for 30–60 days, at which point a replacement is selected [21].

Thus a user's guards are an attractive target for the Sniper Attack. If few enough of a user's guards are responsive (at most 1 in Tor), the user will select new guards as replacements. By disabling the user's guards, the adversary can cause the user to choose new guards and hope that an adversarial relay is among them. This process can be repeated until the adversary succeeds.

This attack requires the adversary to identify the target's guards and to force her to choose new ones as soon as the old ones are disabled. Doing so is particularly easy with hidden services [34] because they create circuits on demand. Therefore, we will describe and analyze the attack applied to hidden services.

Deanonymizing Tor clients using the Sniper Attack is less straightforward because they generally do not respond on demand. However, in some significant cases guards could be identified and guard reselection initiated. For example, a user downloading a large file could give the adversary enough time to discover the guard using a congestion side channel [23], [24], [33]. Furthermore, download managers and BitTorrent clients generally automatically restart an interrupted download, which would prompt guard reselection by Tor.

Finally, we note that in addition to deanonymization, the adversary could use the Sniper Attack to attack Tor privacy in other ways. For example, he could attack the exits of long-lived circuits, such as IRC connections, in order to be chosen as the replacement exit and discover the destination. He could also attack exit relays that allow connections to certain ports in order for adversarial relays to observe a larger fraction of exit traffic to such ports.

A. Deanonymizing Hidden Services

Hidden services provide *responder anonymity* for a persistent service. Users are able to connect to the service through Tor without knowing its location. Let H be a hidden service and C be a client. H chooses a set I of Tor relays as *introduction points* and creates persistant circuits to them. The protocol for using a hidden service is (1) C chooses a Tor relay R to serve as a *rendezvous point* and creates a circuit to it; (2) C chooses an introduction point I, creates a Tor circuit to it, and sends R to H through I; (3) H creates a circuit to R; and (4) C and H communicate to each other over their respective circuits to R.

To perform the anonymity attack on a targeted hidden service, the adversary will need to control at least one relay that can serve as a guard, and he will need to control another relay that can serve as a rendezvous point. For an adversary's relay to be used as a guard, it must satisfy minimum uptime and bandwidth requirements (roughly, its uptime must be at least that of the median relay, and its bandwidth must be at least the minimum of the median bandwidth and 250 KB/s [7]). Any relay in the Tor network can serve as a rendezvous point.

The deanonymization attack proceeds in three phases:

1) Identify the guards of the hidden service;
2) Disable the guards with the Sniper Attack; and
3) Test if the hidden service selected an adversarial relay as a replacement guard, and repeat from 1) if not.

To describe these phases in detail, let G_A be the adversarial relay that can be used as a guard, R_A be the adversarial relay intended to be used as a rendezvous point, C_A be an adversarial Tor client, and H be the target hidden service.

Phase 1 (Identify Guards): A user can force H to select a new circuit by requesting a new connection through a rendezvous point. H chooses the circuit's relays other than the guard roughly at random weighted by bandwidth. Thus, by requesting enough connections, the adversary will eventually cause H to choose circuits such that, for every guard of H, in some of those circuits the adversarial relay G_A is the hop after that guard. For these circuits, the adversary can directly observe the guards' identities, although he may not realize it.

Biryukov et al. describe an efficient technique for the adversary to recognize when he is in such a situation [12]. The rendezvous point R_A sends a pattern of 50 `PADDING` cells to H down the rendezvous circuit followed by a `DESTROY` cell. If G_A observes a pattern of 2 cells on a rendezvous circuit from a hidden service and 52 cells on the same circuit to the hidden service (the cells in excess of 50 are from circuit construction), followed by a `DESTROY` cell shortly after one is sent by R_A, it concludes that the relay one hop closer to H on the circuit is a guard of H. During experiments on the live Tor network, Biryukov et al. observed no false identifications using this method. They also note that the attack could be performed without an adversarial rendezvous point, although it would slow the attack because the rendezvous circuit must extend to C_A.

Using this method, the adversary can quickly, efficiently, and perfectly identify all guards of H. Moreover, the discovery process looks fairly innocuous to H, which only sees a series of normal rendezvous requests. Of course, all such requests are to the same rendezvous point, the connections may appear abnormally fast, and no data is ever carried on the circuits. If stealthiness is a goal, the attack could be mounted from C_A with normal rendezvous point selection, at a slower rate, and including some typical data requests as cover. This would come at the cost of some speed and efficiency. Note also that Øverlier and Syverson [34] describe a less-efficient method of guard identification that depends on performing traffic correlation that is less precise but is more robust to countermeasures.

Phase 2 (Disable Guards): Once H's guards have been identified, the adversary can use the Sniper Attack to cause the Tor process of each guard to be killed. The attack can be run against all guards in parallel to reduce the time of the

attack to the time to kill just one guard. Moreover, attacking the guards at the same time increases the length of time that the guards remain simultaneously unavailable. Eventually, we would expect the relay operator to notice that the Tor process was killed and restart it.

Phase 3 (Test for Guard Selection): Once the hidden service's guards are disabled, the adversary can easily cause new ones to be selected simply by connecting normally to the service. Then he can determine if his guard G_A was selected by H using techniques very similar to those used to identify guards in Phase 1. A difference is that he would look on the circuits of G_A for those with 3 cells from the circuit origin and 53 cells towards it before destruction. This step requires only enough circuits that any given guard of H is sufficiently likely to be used for at least one (*e.g.* with 35 circuits, the probability of such a selection failing to occur is at most $(2/3)^{35} \leq 10^{-6}$).

B. Evaluation

The DoS Deanonymization Attack executes a number of three-phase *rounds* until it succeeds. To estimate the time to complete round i of the attack on hidden service H, let t_1^i be the time to identify the guards of H (Phase 1), t_2^i be the time to disable the guards of H (Phase 2), and t_3^i be the time to test if H selected a malicious relay as a guard (Phase 3). Let r be the number of rounds needed for the attack. Then the total attack time t can be expressed as $t = \sum_{i=1}^{r} t_1^i + t_2^i + t_3^i$. We estimate t for the actual Tor network and various sizes of the adversary, and use real Tor network data from Tor Metrics [8].

1) Time for Phase 1 (t_1^i): To identify the guards of H, the adversary runs a malicious relay M_A and creates enough connections to H such that, for each guard G, a resulting circuit from H to the rendezvous point uses M_A as the middle relay and G as the guard. The connections to H can be created in parallel to speed up this phase. Let t_c be the time from initiating a connection to H until M_A observes the cell pattern that indicates its presence on a rendezvous circuit of H. Let c^i be the number of connections that are needed for M_A to observe all guards of H. Let ℓ be the number of connections created in parallel. The time for this phase is then $t_1^i = t_c c^i / \ell$.

To estimate t_c and c^i, we ran a hidden service experiment in Shadow. During the experiment, a client repeatedly created new connections to a hidden service and sent the 50-cell traffic signature used to recognize relays on the resulting circuit. Note that we used the version of the attack in which the client sends these cells rather than the rendezvous point. We recorded the paths of these circuits and the time from initiation of the connection until the service received the traffic signature. Our experiments were performed in two sessions, each with 10 client-server pairs run in parallel and with background traffic.

During these experiments, 8319 connections to hidden services were created. The average time between starting a connection at the client and receiving the inserted cell pattern at the server was 10.69s. The minimum time was 1.45s and the maximum time was 319.87s. Thus we expect that $t_c = 10.69$.

Our expectation for c^i depends on the bandwidth of M_A. The higher the bandwidth is, the more likely that M_A is selected in a circuit and the lower that c^i is. Thus we consider a range of capacities for M_A. Table III shows the average number of connections that clients had to make to identify the guards of H when we consider relays of different sizes to be the malicious relay M_A. The relays we select were chosen middle relays with probabilities that range from 0.0026 to 0.030. We estimate the bandwidth a Tor relay would need to be to be selected with those probabilities using a linear regression on the consensus weights and the estimated relay capacity. The regression is on network data from 6/30/13. We can see that for relays with bandwidth in the range of 8–100 MiB/s, the average number of connections c^* needed to identify all guards ranges from 598 to 118. c^* is a good estimate for c^1, and as the attack progresses through additional rounds the expectation for c^i only decreases as relays are disabled and the malicious relay becomes a larger fraction of the active network. Thus we can conservatively use c^* as the estimate for all c^i.

TABLE III: Speed of Guard Identification

Selection Prob. as Middle	Tor BW (MiB/s)	Avg # of Cxns to Identify All Guards	t_1^* (min), $\ell = 10$
0.0026	8.41	598.00	10.65
0.0052	16.65	357.33	6.37
0.010	31.97	227.94	4.06
0.021	66.04	141.74	2.53
0.030	96.61	118.40	2.11

We can then use $t_1^* = t_c c^* / \ell$ as a conservative estimate for the time t_1^i to complete Phase 1 in round i. Table III shows this time for $\ell = 10$ parallel connections. We use this value of ℓ because our experiments consisted of 10 clients simultaneously connecting to hidden services. However, we expect that many more connections could be created in parallel without increasing the connection time t_c much because the time is dominated by network latency and creating a connection uses relatively little network bandwidth. This could potentially decrease t_1^* to as little as $t_c = 10.96s$.

2) Time for Phase 2 (t_2^i): During the ith round of a given attack, the relay will have selected a set of guards (Tor uses at most 3). We suppose that the Sniper Attack can be run in parallel on all of these, and thus the time t_2^i to disable all of them is the longest time it takes to disable any one of them. Given a set of guards, we can use the linear regression of Section III to estimate the memory consumption rate from the Tor consensus weight. Then we can consider the time to fill each guard's memory for varying memory capacities.

3) Time for Phase 3 (t_3^i): During Phase 3, the adversary creates enough connections to H that if G_A has been chosen as a guard of H, it will be detected on a resulting rendezvous circuit. We suppose that the adversary creates 35 connections so that if G_A is a guard it fails to be used as a guard on one of the resulting rendezvous connections with probability less than 6.87×10^{-7}. We use our previous estimate for the expected circuit construction time of 10.69s and suppose that the adversary makes 10 parallel circuits. We thus estimate that $t_3^i = 4 * 10.69 = 42.76$s for all i.

4) Time for DoS Deanonymization Attack (t): To provide an estimate for t, we simulate the selection of guards by H during the attack using the TorPS tool [5]. As input to TorPS, we use a Tor consensus and server descriptors from 6/30/13. We perform 10,000 simulations of the attack. During each simulation, guards are selected by H in each round until G_A is chosen. We estimate the total time t for a simulation by adding the given phase estimates in each round.

Table IV shows the results of these simulations. For each bandwidth capacity of the malicious guard G_A, we can see the resulting probability p of being chosen during an individual guard selection. This directly affects the expected number of rounds needed for deanonymization, which we can see ranges

TABLE IV: Speed of DoS Deanonymization Attack

G_A BW (MiBps)	Guard Prob.	Avg # Rounds	Avg # Sniped	Avg Time 1 GiB (H:M)	Avg Time 8 GiB (H:M)
8.41	0.0048	65.66	132.33	45:30	278:14
16.65	0.0097	38.55	78.09	22:27	148:34
31.97	0.019	23.03	47.05	12:12	83:49
66.04	0.038	12.33	25.67	5:57	43:12
96.61	0.054	8.75	18.50	4:08	30:22

from 8.75 to 65.66. These values can in general be roughly estimated as $1/(2p)$ because Tor only replaces the sniped guards in each round with two new guards. The number of guard sniped during the attack, shown next, ranges from 18.50 to 132.33 and is also simply $2r + 1$. The total time t for the attack has a range of 4 hours and 8 minutes to 45 hours and 30 minutes if all Tor relays have 1 GiB of free memory and a range of 30 hours and 22 minutes to 278 hours and 14 minutes if Tor relays have 8 GiB free. Clearly, the time to snipe the guards dominates t, and so we can approximate it simply with $t \approx rt_2^1$. Thus, the adversary can significantly reduce t by running a guard or guards with a large amount of total bandwidth, which decreases r in expectation.

Finally, we note that it is quite possible that some guard operators become aware that their guards have crashed and restart them while the attack is still executing. Tor will go back to using such guards once they become available again. Thus, it may be necessary during the attack to snipe guards multiple times to keep them down.

V. Defenses Against Sniper Attacks

The Sniper Attack exploits two fundamental problems with Tor's design: a lack of enforcement to follow the flow control protocol; and unchecked, unbounded application queues. In this section, we address these problems by exploring defense strategies (summarized in Table V) and their costs, limitations, and practical operational deployment issues.

A. Authenticated SENDMEs

One problem exploited by the Sniper Attack is that the packaging edges are unable to verify that the delivery edges actually received any cells. One solution to this problem is adding a challenge-response puzzle to every 100th cell. Each packaged cell currently includes a hash digest of its contents so that bit errors may be detected by the client. A package edge can require that the digest of each packaged cell be included in the corresponding SENDME feedback signal cell. To prevent the delivery edge from pre-computing this digest when downloading a known file, the package edge could include a 1 byte nonce in every 100th cell. This nonce will randomize the digest that must be returned in the SENDME, and can only be guessed with probability 1/256. If the response digest doesn't match the challenge, the exit can drop the circuit. Authenticated SENDMEs prevent clients from subverting the 1000 cell in-flight limit, including those who attempt to "cheat" by preemptively sending SENDMEs to the exit in order to download data faster.

This defense provides an elegant solution to detecting protocol violations. It defends against a single client using the efficient version of the Sniper Attack. However, using this approach alone has some limitations. First, it does not completely stop the attack: each circuit will still be able to cause the target to queue 1000 cells (500 KiB), and so the target can still be taken down using the parallel attack from Section II-C2. Second, relays are relying on honest circuit members to perform the authentication protocol correctly, and therefore this defense does not protect against either of the basic versions of the Sniper Attack where the packaging edge is malicious. We could improve the situation by allowing intermediate relays to read and detect unexpected SENDME cells and destroy the circuit, but we note that a self-defense strategy is preferred to one that relies on other circuit members. Finally, this approach has a longer transition phase, since all clients and at least all exit relays need to be aware of the authentication protocol.

TABLE V: Defense Capabilities

Defenses \ Attacks	Basic V1	Basic V2	Efficient	Parallel
Authentication	No	No	Yes	No
Length Limit	Yes	Yes	Yes	No
Circuit Killer	Yes	Yes	Yes	Yes

B. Queue Length Limit

Another problem exploited by the Sniper Attack is that Tor's application queues may grow without interference by the relay. Therefore, a simple defense is for each relay to enforce a maximum queue size to limit the amount of memory each circuit may consume. If the queue length becomes greater than the allowed size, then the relay may assume a protocol violation and destroy the circuit to inhibit malicious activities.

To find a good candidate queue size, we consider that Tor's flow control algorithm already enforces a limit on the number of cells that may be in transit (1000, plus some tolerance for control messages). One approach would be to use a similar limit as a queue length limit, which provides a self-defense mechanism while also protecting against adversaries who control multiple nodes in a circuit. However, as with the authenticated SENDMEs defense, a queue length limit does not prevent an adversary that uses the parallel Sniper Attack from circumventing the memory limitations, since memory consumption from its multiple circuits in aggregate can still crash the relay with relatively low overhead. Further, a maximum queue length would obstruct future development. Considering that the Tor Project anticipates custom transport protocols with dynamic feedback mechanisms, a hard threshold on the queue length may complicate migrations. Finally, we note that the queue length limit defense enables a new attack in which webservers could inject page objects that require new streams and cause benign circuit queues to grow beyond the limit and therefore be destroyed [18].

C. Adaptive Circuit Killing

To overcome the limitations of the previous defenses and protect against the parallel Sniper Attack, we now develop a more sophisticated, adaptive mechanism which is incrementally deployable and has strong security properties. A clever attacker against both of the previous defenses can use a sufficiently high number of parallel circuits, each with a short queue, to exhaust a relay's memory. To prevent memory exhaustion, a relay can begin and continue to kill circuits while the *total* memory consumption remains above a critical memory threshold. This technique will guarantee that a relay process will not terminate due to an out-of-memory condition.

1) Selecting Circuits: The central question to be solved is to decide which circuit should be killed if memory becomes scarce. This question is not as simple to answer as it might

seem at a first glance. For instance, the most straightforward approach would be to kill the circuit with the longest queue. This, however, can be leveraged for a new attack: an adversary could set up a large number of circuits with relatively short queues on a given relay, so that this relay's memory consumption is very close to critical. Whenever a benign circuit temporarily builds up a long queue, the threshold will be exceeded and a *benign* circuit will be killed, while the adversary's (shorter) circuits will remain in place. The relay is therefore manipulated in such a way that it will regularly kill benign circuits—without any need for the attacker to spend resources beyond initially setting up the circuits. While the relay will not crash due to running out of memory, this is still highly undesirable.

We must therefore aim for a decision criterion which cannot be abused by an attacker to make a relay kill benign circuits. Here, we propose to use the time of arrival of the frontmost cell in the queue as the basis for our decision: if memory becomes scarce, the circuit killing mechanism will kill the circuit with the currently oldest cell at the front of its queue. We require that each incoming cell be tagged with a timestamp upon arrival at a relay, but note that this already happens in the current versions of Tor in order to compute cell delay statistics. Therefore, this mechanism is almost trivial to implement. In the remainder of this section, we will argue why it is also effective.

To gain an intuitive understanding, observe that an attacker—in order to avoid that his circuit is killed when memory becomes scarce—will have to keep the frontmost cell in the circuit's queue "fresh". Since Tor circuit queues are strict FIFO queues, the frontmost cell in any given circuit queue will have spent more time in this queue than any other cell. The attacker is therefore forced to continuously read from all his circuits; otherwise, the cell at the attack circuit's head will soon be older than the frontmost cells in the queues of benign circuits. Thus, by deriving bounds on the share of the relay's available bandwidth that is required in order to make a relay kill a benign circuit, we will be able to prove the effectiveness of the defense strategy.

2) Proof Sketch: Consider a specific relay which offers a total bandwidth B for relaying Tor circuits. We assume that B is available both in incoming and in outgoing direction (substantially imbalanced incoming and outgoing bandwidths do not make sense for a relay which essentially forwards all incoming data). Furthermore, assume that this relay is currently used by a total of n active circuits. We define an active circuit as a circuit which currently has at least one cell in its queue.

If the outgoing bandwidth of the relay were assigned to the active circuits in a perfectly fair manner, then each circuit would experience an outgoing data rate of

$$r_{\mathit{fair}} = \frac{B}{n}. \tag{1}$$

Of course, in practice, the distribution will not be perfectly fair; in fact, there are certain known artifacts with respect to inter-circuit fairness [43]. But Tor relays include mechanisms which will still result in bandwidth allocations to circuits that are not arbitrarily unfair: there is a round-robin scheduler which picks cells from circuit queues for transmission. Moreover, circuits are carried over TCP connections, and TCP, too, strives for a fair distribution of available bandwidth to multiple connections. Both of these mechanisms are controlled by the relay and are thus outside the sphere of influence of an attacker. We will discuss the case of an attacker who is able to claim a huge fraction of the relay bandwidth for himself later. For now, we may reasonably assume that there is a fairness factor $0 < \alpha \leq 1$ such that each active circuit receives a bandwidth share of at least

$$r \geq \alpha \frac{B}{n}. \tag{2}$$

As we will see, the exact value of α is not critical for our scheme, as long as an active circuit's bandwidth share does not become arbitrarily small for a longer period of time.

Now observe that benign circuits will typically have queues which are bounded above by a relatively small size Q. Q is, as discussed before, in the order of 1000 cells in the current Tor protocol. Even if possible future protocol versions do not enforce a hard upper limit, observe that high values of Q imply long queues in the relays and thus poor circuit performance. In practice, any reasonable future protocol design will therefore also result in reasonable queue lengths. Note that while we assume that such an upper bound Q exists in our analysis, its value need not be known and is not used to decide which circuit to kill. The exact value is thus much less critical than in the previously discussed queue length defense.

Based on these assumptions we make a central observation for our argument: if a benign circuit's queue length does not exceed Q and its mean rate is at least r, then the maximum time for which a cell can remain queued is bounded above by

$$d_{\max} = \frac{Q}{r} = \frac{Qn}{\alpha B}. \tag{3}$$

Therefore, if t_{now} is the current point in time, the cells at the heads of all benign circuits' queues will have a timestamp later than $t_{\mathit{now}} - d_{\max}$.

Note that an attacker using a single circuit will thus have to make sure that the cell at the front of the queue does not become older than $d_{\max}$, i. e., the cell must have arrived at a point in time later than $t_{\mathit{now}} - d_{\max}$. Only then can the attacker hope that a benign circuit will be killed instead of the attacker's circuit. If the attacker uses multiple circuits in parallel, the same criterion must hold for all these circuits. Consequently, all the cells in the attacker's circuits must have arrived within a time interval of length $d_{\max}$.

Let the amount of free memory at the relay be denoted by M. The attacker must (roughly) build up queues with a total size of M bytes in order to make the relay kill circuits. Since, as seen before, the attacker must inject all these cells within a time span of length $d_{\max}$, the attacker needs to send cells into the relay at a mean rate of at least

$$r_a = \frac{M}{d_{\max}} = \frac{M}{Q} \cdot \alpha \frac{B}{n} = \frac{M}{Q} \cdot r. \tag{4}$$

This is a factor of M/Q higher than the minimum outgoing rate r which we assumed for benign circuits above in (2). Observe that M/Q can easily be made a very large number if sufficient memory is provided. We recommend an order of magnitude of a few hundred megabytes, which is not a problem on today's relays (also on machines with a 32 bit address space) and results in a factor M/Q in the order of 1000.

The attacker would therefore have to claim the incoming relay bandwidth virtually entirely for himself in order to mount a successful attack that results in a benign circuit being killed. Although such an attack is possible if an adversary has enough

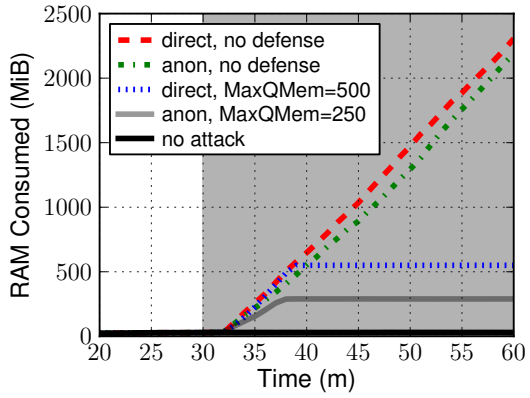


Fig. 5: The circuit killer renders the Sniper Attack ineffective.

bandwidth, we consider it practically unrealistic for two key reasons: first, fairness mechanisms are in place also on the incoming side of a relay, making it very hard to achieve this in the first place; and second (and much more important), observe that consuming almost all of a relay's bandwidth constitutes by itself a far more devastating attack on the relay. An adversary with enough bandwidth to succeed in this attack and cause a relay to drop a few benign circuits would do more damage using its bandwidth in a classic DoS attack, or in a selective DoS attack [13] launched while running malicious relays. (A bandwidth attack on a relay may in fact kill benign circuits anyway, e. g., due to TCP connections timing out.)

3) Evaluation: We implemented the described out-of-memory (oom) circuit killing as a Tor software patch. It introduces a new configuration parameter `MaxQMem`, which specifies the maximum amount of memory usable by Tor's circuit queues. Every second the algorithm checks for violations to this threshold and kills a circuit if necessary. We re-run the experiments from Section III (the results of which are shown in Figure 4a) with the oom circuit killer deployed on all relays, using a `MaxQMem` of 500 MiB for the direct and 250 MiB for the anonymous Sniper Attack (we chose different values solely for a clearer presentation). The results in Figure 5 contrast the memory consumption with and without our defense. With our defense in place, it depicts a horizontal line around the configured `MaxQMem` during the attack, showing that the consumed memory is bounded by our new parameter. Closer examination shows a microscopic oscillation around the threshold, i.e. first surpassing it, then freeing memory, and then rising again due to the other sniper circuits. It successfully protects the process from arbitrarily increasing the memory consumption and thus from being killed. During the experiments of the direct and the anonymous attack the circuit killer intervened 43 and 32 times respectively, and in all cases only attacking circuits were killed. Thus this defense resulted in a 100% identification rate with no false positives.

The above results reveal insights into the interplay between fairness and the robustness against the Sniper Attack when such a mechanism is in place. An attacker needs a lower rate and thus fewer resources either if the queues of benign circuits become longer (higher value of Q) or if the distribution of relay bandwidth to the circuits becomes less fair (smaller value of α). Approaches that bound the queue lengths based on per-link feedback [10], or improve transmission scheduling in Tor [43] would therefore complement this defense strategy.

In summary, we believe that adaptive circuit killing based on the queuing duration of the cells at the heads of the queues constitutes a strong defense. Not only does it prevent a relay from crashing due to insufficient memory, it is also very resilient against being abused to make relays kill benign circuits. It is simple to implement and easily deployable: the mechanism need only be implemented on the relays, and it is immediately effective on all relays where it is deployed.

VI. Defense Against DoS Deanonymization

Our proposed defenses against the Sniper Attack protect against memory exhaustion but do not protect against brute force network or CPU overloading. In addition, other DoS attacks on Tor continue to be discovered [31], [36], and a Tor relay is vulnerable to all DoS attacks on the host platform. The Deanonymization DoS Attack can be performed using *any* DoS attack on a Tor relay and thus is still a serious problem.

As a defense against it, we suggest that the Tor client limit the number of relays that it chooses for the most sensitive positions in its circuits. In the following we describe this proposal in detail, and we evaluate its security and its cost in terms of network performance.

A. Limiting Network Exposure

The key vulnerability in Tor path selection that we exploit is that a client is willing to choose an unlimited number of entry guards in a period of time. We propose the simple fix of limiting the rate at which clients will add relays to their entry guard list. In addition, hidden services make guard discovery fast for the adversary by selecting new circuits for each connection. To slow this down, we suggest that hidden services use two levels of guards for their rendezvous circuits.

Our first proposed change to the path-selection algorithm limits entry-guard selection. This change applies to any new circuit, including exit circuits, rendezvous circuits, and introduction circuits. It tries to maintain a certain number a_g of *active* guards, that is, guards that are currently responding. For improved security, though, it puts a hard limit r on the number of *recent* guards, that is, guards selected more recently than t time ago. Specifically, the algorithm for circuit guard selection is as follows: if there are at least a_g active guards, return a random one; else if there are fewer than r recent guards, select new guards until either a_g active guards or r recent guards exist and then return a random active guard; else if there are any active guards, return a random one; else return an error. Note that guard *expiration*, that is, removal from the guard list, is a separate procedure handled on a time schedule as Tor currently does [21].

If no active guards are available but the rate limit has been reached, circuit selection cannot proceed. There are a couple of reasonable options for handling this at the user level: (*i*) indicate to the user that Tor will be temporarily unavailable to prevent a possible security attack but allow a manual override, or (*ii*) use a configuration setting for desired security level to determine if circuit construction should be halted.

This algorithm isn't a strict generalization of Tor's current procedure. However, a close approximation is that currently Tor uses $a_g = 2$ and infinite r (Tor prefers 3 guards if possible but only enforces that 2 are active). We consider a range of parameter values in our evaluation. Of course, it only makes sense to have the recent time t less than the expiration time.

Our second proposed change is for hidden services to use *middle guards* for their entry guards when creating rendezvous circuits. A hidden service H maintains a middle-guard set of

relays $\mathcal{M}_G$ for each of its guards G. After choosing a guard G for a new rendezvous circuit, H adds relays to $\mathcal{M}_G$ as necessary to maintain a_m active relays in the set. Then H chooses an active middle guard randomly from $\mathcal{M}_G$ to use in the circuit. Middle guards expire either after some time chosen uniformly from $[e_0, e_1]$ or when their guard expires, whichever occurs first.

The purpose of these middle guards is to increase the time and effort needed for the discovery of hidden-service entry guards, which is otherwise quite fast and cheap. Hiding the identity of the guard helps prevent any DoS-based deanonymization attack. In addition, it frustrates other guard-based attacks. For example, currently a Tor adversary can very easily monitor guard use by a targeted hidden service and notice the use of a guard—even for just a short time—run by an entity or in a jurisdiction for which the adversary can easily set up surveillance or obtain logs.

The design choices in our middle-guard algorithm are made specifically to prevent such attacks. We do not suggest applying rate-limiting to middle-guard selection, as an adversary could then achieve the effect of the entry-guard DoS just by attacking the a_m middle guards. We rather force him to attack enough middle guards to observe the entry guard directly and then be forced to attack it as well. We also do not suggest extending the guard concept beyond the second circuit hop. This would further reduce load balancing, and because guard identification can be achieved via attacks other than DoS [23], [24], [33], there is a limit to the benefit of raising its cost. Finally, we only apply middle guards to rendezvous circuits because client and introduction circuits already have longer lifetimes, and middle guards increase circuit linkability, which is especially a concern for client circuits. Thus we conservatively limit this proposal to where it seems like an unambiguous improvement.

B. Analysis

Both entry-guard rate-limiting and the use of middle guards sets up a tradeoff between security and performance. We consider this tradeoff separately for each defense.

1) Entry-Guard Rate-Limiting: The main performance cost from rate limiting entry guards is that a client may be left at times with no active guards, even just due to benign guard failure. Depending on how the limiting is implemented, this could mean that Tor is unavailable to the user for a period of time or that the user must consider if he is willing to allow a less restrictive rate-limiting than is recommended by default. To evaluate how often this might happen, we simulated rate-limited guard selection. Our simulator selected a new guard using the same constraints and weighting as Tor [6], and it used data from Tor Metrics [8] to determine past network states. We required that the desired number of active guards be available at *every* time instant, a conservative approximation—especially for individual users, who likely only use Tor intermittently.

Table VI shows results of simulations from February to March 2013 (after two months all guards will have expired), where each row represents 5000 simulations. For each setting of a_g and r, we include the largest t for which the fraction of simulations with *any* period of guard unavailability was at most 0.001, if any. The table includes this probability of a period of "down time" as well as the median length among such periods. We can see that with even fairly strict rate limiting, a client almost never experiences down time, and when it does it can recover as fast as within half a day. Guard reselection due to expiration happens at a rate of 1 relay every 15 days, and so we could, for example, set a limit at double this existing rate by setting $a_g = 1$, $r = 4$, and $t = 28$ and still obtain very high availability.

[6]The simulator is based on Tor 0.2.3.25.

TABLE VI: Unavailability from Rate-Limiting Entry Guards

a_g	r	t (days)	Prob. Down	Med. Down Time (H:M)
1	3	7	0.0004	77:12
1	4	28	0.0008	840:30
1	5	28	0	N/A
2	4	14	0.0004	10:54
2	5	28	0.0004	224:24

With rate-limiting in place, it becomes much more difficult for an adversary to push a target client into using a malicious guard using a DoS attack. At most the client can be forced to choose another r guards every t time. Suppose that the malicious guards have probability p of being selected as a guard. During the Deanonymization DoS Attack, the probability that target client chooses a malicious relay within in the first i periods of time t is $1-(1-p)^{ri} \approx rip$, ignoring that p increases slightly as the targeted relays are taken down. For example, consider $r = 4$, $t = 28$ days, and suppose that $p = 0.017$ (this is the top guard probability on 6/30/13). Let the adversary run a DoS every t days against all observed guards of the target that may be unexpired (the maximum expiration time is public). Note that the DoS need only last briefly, and the attack works even if relays come back online shortly afterwards; thus this is not an implausible attack. The probability that the target client selects the malicious guard with 3 months is 0.19. This compares to virtual certainty without guard rate-limiting and to a probability of 0.10 over the same time period without a DoS attack just due to guard expiration.

Clearly, however, over time the DoS Deanonymization Attack will eventually succeed. For users facing a persistant adversary, the only option may be to limit guards to a deliberately chosen set using the relevant Tor configuration options. We can also imagine a more sophisticated automated defense scheme in which multiple guard failures in a short time period are handled with increasing suspicion, but we leave such an improvement to future work.

2) Middle Guards: A potentially significant performance issue with the use of middle guards is that traffic will not be load balanced as effectively because: (*i*) the capacity of an entry guard is higher than the capacity of its middle guards; (*ii*) traffic from different services gets concentrated by chance on middle-relay "hot spots"; or (*iii*) relays join the network but aren't chosen quickly as middle guards.

(*i*) can be mitigated by setting a_m large enough. By looking at the recent 6/30/2013 consensus as an example, we observe that the observed bandwidth of relays weighted by their probability of being selected as an entry guard is 9669.89 KiB/s, and the observed relay bandwidth weighted by the probability of selection as a middle relay is 7721.50 KiB/s. Therefore we could prevent middle-guard bottlenecks in expectation by setting $a_m \geq 2$. In addition, load from all traffic other than hidden services would be load balancing as usual, making the capacities available to hidden-service traffic even more similar between guard and middle relays.

(*ii*) and (*iii*) can both be mitigated by making the average middle-guard expiration short enough that hot spots don't

develop and new relays are quickly used. Middle guards need only slow down guard discovery to the speed of other known methods for identifying guards, such as throughput fingerprinting [33] or the long-path congestion attack [23], which are each effective within hours. Their complexity and resource cost is significantly higher than passive observation at a relay, however, and so the speeds of the attacks need not be equalized. Moreover, because most Tor traffic continues to be load-balanced effectively, the net imbalance from middle guards seems likely to be small.

The defense offered by middle guards is that an adversary running malicious relays cannot quickly discover hidden-service entry guards by sending the service many rendezvous requests. Instead, an adversary trying to directly observe the entry guard must wait to be selected either as the entry guard itself or as a middle guard. With a_m middle guards and an average expiration of $e = (e_0 + e_1)/2$ days, an adversary with a probability p of being selected as a relay will expect to wait $(1/(1-(1-p)^{a_g a_m}) - 1)e$ days until being selected as the middle guard of *some* entry guard. Suppose that $a_g = 1$, $a_m = 2$, $e_0 = 30$ and $e_1 = 60$ (i.e. middle-guard expiration is the same as current entry-guard expiration), and $p = 0.021$ (the largest middle-relay selection probability on 6/30/13). Then the expected time for the adversary to be selected as a middle guard is 1037.79 days.

VII. Related Work

Internet DoS attacks, those that make an Internet service unavailable for longer than the intended waiting time [45], have been extensively studied in the literature. Although unique in this space, the Sniper Attack is most closely related to *low rate* and *slow read* DoS attacks, which are variants of the well-known SYN flood DoS attack [20], [40]. The goal of these attacks is to exhaust resources in order to prevent the victim from processing new incoming connection requests. Transport layer *low rate* attacks [30] exploit TCP's retransmission timeout (RTO) dynamics. An attacker repeatedly sends short high-rate packet bursts, which produce packet losses (i. e., timeouts) and thus make the victim double the RTO of other TCP connections [37]. Transport layer *slow read* attacks [26] send legitimate data requests, advertise a small TCP receive window, and then slowly empty the receive buffer. As a result, the victim's send buffer remains full over a long time span, thus blocking resources. Similar *low rate* and *slow read* techniques have been described to exploit web server weaknesses on the application layer [2], [14], [38]: sending partial HTTP requests or slowly reading responses to requests will prolong the HTTP session and extends the time in which the availability of the web server's connection pool is reduced.

Although the Sniper Attack shares the general goal of preventing new incoming connections with *low rate* and *slow read* attacks, it is achieved as a byproduct of the more direct goal of exhausting system memory resources. In particular, we consume memory from the application layer using valid overlay network protocol messages *without reading from the victim*. Therefore, our attack may be characterized as a *no read* attack. Another important distinction is that, unlike the attacks described above, our attack does not require several simultaneous connections to the target and continued effort in order to maintain the effect of the attack. Finally, our attack destroys existing established connections in addition to preventing new ones.

The Sniper Attack may also be categorized as a *permanent* DoS attack, as it exploits application layer overlay network protocol semantics to consume system memory and crash the process. It is distinguished from similar attacks, such as the Ping of Death [29], in that it utilizes valid messages to exploit the protocol design. Fixing it is therefore not simply a matter of correcting a broken protocol implementation.

Our attack is also similar to those that rely on misbehaving receivers and optimistic ACKs to bypass flow control protocol mechanisms [9], [39], [41]. In particular, the opt-ACK attack [41] is similarly challenged to adjust a feedback signal rate in such a way that it still appears legitimate to the communication partner. Our attack differs in that we target *application* layer protocols of overlay networks in order to exhaust the available *memory*, rather than targeting *network* layer protocols for the purposes of consuming the available *bandwidth*. As such, the Sniper Attack is a *no read memory exhaustion* attack.

DoS attacks against the Tor overlay network have been studied before, building upon a fundamental observation first made by Syverson *et al.* [42]: if the first and the last relay along a Tor path are compromised, an adversary can link the source and destination by correlating traffic patterns. Øverlier and Syverson first demonstrated how an adversary could lie about the available bandwidth of compromised relays in order to inflate the probability of being selected for a hidden service circuit [34], and Bauer *et al.* extended the attack to increase the probability of end-to-end compromise of general purpose circuits [11]. Borisov *et al.* [13] describe a selective DoS attack on Tor where malicious relays terminate circuits of which they are a part but do not control both ends. This forces clients to re-build circuits and similarly increases the probability of end-to-end compromise by the adversary. Danner *et al.* show how selective DoS attacks can be provably detected by exhaustively probing potential paths [15], while Das and Borisov reduce the cost of detection using probabilistic inference [16].

Resource consumption attacks that may also be used to increase an adversary's probability of end-to-end circuit compromise include the Packet Spinning attack [36] and the CellFlood attack [31]. In the Packet Spinning attack, the adversary crafts special packets that cause them to continuously "spin" through circular circuits composed of the target relays. In the CellFlood attack, the adversary uses special handshake packets to efficiently build a large number of circuits through the target relays. Both of these attacks effectively make relays appear busy by forcing them to spend resources doing unnecessary work. Honest clients' circuits through these relays will then be more likely to time out, causing them to choose new circuits containing malicious relays with higher probability. The Sniper Attack also causes relays to perform unnecessary work, but focuses on consuming memory resource rather than bandwidth or computational resources.

Our hidden service deanonymization attack builds upon techniques developed in previous work. In particular, Øverlier and Syverson first identified that hidden services could be located quickly and easily [34] because rendezvous circuits are created on demand using new relays for each circuit. The adversary could therefore continue to make new connections to a hidden server until traffic correlation indicated that the hidden server built a circuit that directly connected to one of the adversary's nodes. They further described how a hidden server using guard nodes would still be insecure against an adversary using the attack to identify the hidden server's

guards and then DoS them: this process could be repeated until one of the adversary's nodes was chosen as a new guard. They outlined layered guards, or guard nodes for the guard nodes, to help defend against such an attack. Biryukov *et al.* showed how the adversary may detect its position on a rendezvous circuit by simply counting cells instead of performing traffic correlation [12]. Finally, Øverlier and Syverson introduced Valet Service nodes to improve the resilience of introduction points against DoS attacks [35].

VIII. Conclusions and Future Work

In this paper we presented a novel and destructive DoS attack against Tor that may be used to anonymously disable arbitrary Tor relays by exploiting the protocol's reliable end-to-end data transport. We outlined several ways to carry out the Sniper Attack and assessed its resource and time profiles in large scale simulations. We performed an in-depth security analysis, showing how the attack may be used to deanonymize hidden services. We developed a defense that identifies and kills malicious circuits in out-of-memory (oom) situations and showed that it renders the attack ineffective. Finally, we suggested alternative guard and path selection policies that enhance Tor users' security.

Although the Sniper Attack is tuned for Tor, our mechanisms may generalize to systems that do hop-by-hop reliability and end-to-end flow control. We leave it to future work to analyze the extent to which this generalization applies. Further, although our defenses prevent memory exhaustion, they do not stop the Sniper Attack from consuming a large amount of Tor's bandwidth capacity at low cost. Future work should consider this and other bandwidth consumption attacks, as well as defenses against them.

Acknowledgments

We thank the anonymous reviewers for their feedback and suggestions, Damon McCoy for discussions about misbehaving receivers and authenticated signals, and Roger Dingledine for discussions about attack and defense variations. Aaron Johnson was supported by the Office of Naval Research (ONR) and DARPA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or ONR. We are also grateful for the DFG grant supporting this work.

References

[1] "OOM Killer," http://linux-mm.org/OOM_Killer.
[2] "R-U-Dead-Yet (RUDY)," https://code.google.com/p/r-u-dead-yet/.
[3] "Shadow Source Code," https://github.com/shadow/shadow.
[4] "TC: A Tor control protocol (Version 1)," https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=control-spec.txt, Acc. June 2013.
[5] "The Tor Path Simulator," http://torps.github.io/.
[6] "The Tor Project," https://www.torproject.org/.
[7] "Tor directory protocol, version 3," https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt, Acc. July 2013.
[8] "Tor Metrics Portal," https://metrics.torproject.org.
[9] F. Adamsky, S. A. Khayam, R. Jager, and M. Rajarajan, "Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver," in *CyberC '12*, Oct. 2012.
[10] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. Voelker, "DefenestraTor: Throwing out Windows in Tor," in *PETS '11*, Jul. 2011.
[11] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-Resource Routing Attacks Against Tor," in *WPES '07*, Oct. 2007.
[12] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization," in *SP '13*, May 2013.
[13] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of Service or Denial of Security?" in *CCS '07*, Oct. 2007.
[14] T. Brenann, "OWASP HTTP Post Tool," https://www.owasp.org/index.php/OWASP_HTTP_Post_Tool.
[15] N. Danner, S. Defabbia-Kane, D. Krizanc, and M. Liberatore, "Effectiveness and Detection of Denial-of-Service Attacks in Tor," *ACM TISSEC*, vol. 15, no. 3, Nov. 2012.
[16] A. Das and N. Borisov, "Securing Anonymous Communication Channels under the Selective DoS Attack," in *FC '13*.
[17] R. Dingledine, "#6252 didn't go far enough," https://trac.torproject.org/projects/tor/ticket/9063, June 2013.
[18] ——, "#9063 enables Guard discovery in about an hour by websites," https://trac.torproject.org/projects/tor/ticket/9072, June 2013.
[19] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security '04*, Aug. 2004.
[20] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," RFC 4987, IETF, Aug. 2007.
[21] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg, "Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor," in *WPES '12*, Oct. 2012.
[22] T. Elahi and I. Goldberg, "CORDON–A Taxonomy of Internet Censorship Resistance Strategies," University of Waterloo CACR 2012-33, Tech. Rep., 2012.
[23] N. S. Evans, R. Dingledine, and C. Grothoff, "A Practical Congestion Attack on Tor Using Long Paths," in *USENIX Security '09*, Aug. 2009.
[24] J. Geddes, R. Jansen, and N. Hopper, "How Low Can You Go: Balancing Performance with Anonymity in Tor," in *PETS '13*, Jul. 2013.
[25] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding Routing Information," in *IHW '01*, May 1996.
[26] ithilgore, "Exploiting TCP and the Persist Timer Infiniteness," *Phrack Magazine*, vol. 0x0d, no. 0x42, Jun. 2009.
[27] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine, "Methodically Modeling the Tor Network," in *CSET '12*, Aug. 2012.
[28] R. Jansen and N. Hopper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation," in *NDSS '12*, Feb. 2012.
[29] M. Kenney, "Ping of Death," http://insecure.org/sploits/ping-o-death.html.
[30] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted Denial of Service Attacks and Counter Strategies," *IEEE/ACM TON*, vol. 14, no. 4, 2006.
[31] V. P. Marco Valerio Barbera, Vasileios P. Kemerlis and A. Keromytis, "CellFlood: Attacking Tor Onion Routers on the Cheap," in *ESORICS '13*, Sep. 2013.
[32] N. Mathewson, "We should have better, fairer OOM handling," https://trac.torproject.org/projects/tor/ticket/9093, June 2013.
[33] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting," in *CCS '11*, Oct. 2011.
[34] L. Øverlier and P. Syverson, "Locating Hidden Servers," in *SP '06*, May 2006.
[35] ——, "Valet Services: Improving Hidden Servers with a Personal Touch," in *PETS '06*, Jun. 2006.
[36] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos, "Compromising Anonymity Using Packet Spinning," in *ISC 08*, Sep. 2008.
[37] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, IETF, Jun. 2011.
[38] RSnake, "Slowloris HTTP DoS," http://ha.ckers.org/slowloris/.
[39] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver," *ACM SIGCOMM CCR*, vol. 29, no. 5, 1999.
[40] S. Shalunov, "Netkill – generic remote DoS attack," http://seclists.org/bugtraq/2000/Apr/152, 2000.
[41] R. Sherwood, B. Bhattacharjee, and R. Braud, "Misbehaving TCP Receivers Can Cause Internet-wide Congestion Collapse," in *CCS '05*, Nov. 2005.
[42] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an Analysis of Onion Routing Security," in *DIAU '00*, Jul. 2000.
[43] F. Tschorsch and B. Scheuermann, "Tor is unfair – and what to do about it," in *LCN '11*, Oct. 2011.
[44] P. Winter and S. Lindskog, "How the Great Firewall of China is blocking Tor," in *FOCI '12*, Aug. 2012.
[45] C.-F. Yu and V. D. Gligor, "A Formal Specification and Verification Method for the Prevention of Denial of Service," in *SP '88*, May 1988.