Denial-of-service attacks in Tor: Taxonomy and defenses

Nick Mathewson
The Tor Project
nickm@torproject.org

Tor Tech Report 2015-10-001
October 29, 2015

Abstract

This document surveys what we know about potential denial-of-service attacks against Tor, in order to focus our efforts over the coming years to increase Tor's resistance against these attacks.

Introduction

This document surveys what we know about potential denial-of-service attacks against Tor, in order to focus our efforts over the coming years to increase Tor's resistance against these attacks.

Why? We spend a lot of time thinking about anonymity defenses and security defenses, and less about DoS defenses. And while actual DoS attacks up to present have been (with a couple of exceptions) comparatively unsophisticated, we should not let our approach to them become reactive: instead we should systematize our approach to defending Tor against these attacks, and take a proactive stance towards them.

Denial of service attacks are a special problem for anonymity networks, since many of them can be adapted to weaken anonymity. Bugs that would simply be annoying resource exhaustion issues against other network programs can be a real threat to user security.

Here we provide an overview of denial-of-service vectors against anonymity networks in general and Tor in particular, and discuss how best to prioritize responses against them.

We'll conclude below that our best focus in each area is to look for categorical defenses and protocol improvements, to better resist more categories of attacks before they occur.

This work was partially supported by the Open Technology Fund, and by the National Science Foundation under Grant No. CNS-1111539, CNS-1314637, and CNS-1520552. 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 the Open Technology Fund or the National Science Foundation.

Methodology

There are quite a few ways to classify DoS attacks: by the exhausted resource, by the kind of service targeted, by the required attacker resources, by the required attacker expertise, etc.

Here we are considering each attack in terms of how well the attack achieves the attacker's goals. For each attack we will consider:

- How difficult the attack is to perform.
- How difficult it is to exploit the attack to achieve particular outcomes.
- How repeatable the attack is over time. (How hard it is to detect; how hard it is to prevent once detected.)
- The difficulty of mitigating this sort of attack.

Results and Taxonomy

Attacks by resource

CPU consumption attacks Onion routing servers perform expensive cryptographic computations on behalf of their users. Additionally, some computations internal to the Tor process use inefficient algorithms (like linear searches), which might prove problematic if those algorithms' inputs prove large.

An attacker could mount CPU consumption attacks by running a large number of legitimate clients and shoving traffic through them. That's comparatively simple to do, but comparatively expensive.

Worse, an attacker could construct bogus cryptographic requests that require relays to perform a lot of computation before realizing that the request was spurious. An attacker can also mount CPU consumption attacks by finding an inefficient algorithm in and provoking it to be used far more than it would be in ordinary operation. These attacks need a little engineering and programming skill, but create a higher impact for a given amount of computation.

CPU exhaustion attacks can reduce the CPU resources available for legitimate operations. They can also interfere with other traffic, thereby creating patterns in users' streams that can be observed elsewhere in the network.

CPU exhaustion attacks are usually noticeable only when done blatantly. If a large number of nodes suddenly see their CPU load spike, or if usage slows to a crawl across the whole network, we'll get reports of it– but small interference with targeted relays or nodes is harder to detect.

To defend against CPU exhaustion attacks, our best responses are:

- To improve the efficiency of our implementation in places where it uses inefficient algorithms.
- To move more work from the main thread into other threads, thereby increasing other threads.
- To modify our protocols so that an attacker cannot spend a small amount of CPU in order to force the server to use a lot.

- To radically improve our work- and cell-scheduling architectures so that it is harder for one user's traffic to interfere with another's.
- Possibly, to instrument our code so that we can better detect strange patterns in CPU usage. We'd need privacy-preserving ways to do this.

RAM consumption attacks Tor servers will, for many purposes, allocate memory in order to fulfil un-trusted requests. Operations that allocate memory are below. Ones that Tor currently handles in its OOM-handler (see below) are marked with HANDLED.

- Queueing cells on a circuit for transmission. (HANDLED)
- Queueing data to be sent on an edge connection. (HANDLED)
- Storing hidden service directory entries. (HANDLED)
- Queueing directory data for compression. (HANDLED)
- Internal OpenSSL buffers. (HANDLED.)
- Storing uploaded descriptors as a directory authority. (!)
- Key-pinning entries on directory authorities. (!)
- Cached DNS replies. (!)
- Replay caching on hidden services.
- Storing votes from other directory authorities.
- Storing statistics as a statistics-collecting node.
- Per-circuit storage.
- Per-connection storage.
- Per-stream storage.
- Queued circuit-creation requests.

For cases when Tor knows how to handle memory pressure in a given area, it keeps track of the amount allocated, and responds to memory pressure by killing off long-unused connections and circuits.

When Tor has not been taught to handle memory pressure in a particular area, it runs a risk of running out of memory in response to attacks. Whens the Tor process uses too much memory, the OS will respond in one of two ways:

1. On some operating systems, memory allocation functions will return NULL. Tor responds to this currently by aborting the process.

2. On Linux, the kernel kills processes when the system runs low on memory, using a set of heuristics that pleases nobody.

Memory exhaustion attacks per se are comparatively inexpensive: the most usual types require the attacker to transmit only one encrypted byte per byte consumed on the target. (Tor already takes steps to prevent the even cheaper versions, where the attacker uploads a compressed zlib bomb.)

The impact of a memory exhaustion attack can be blatant (crashing the Tor process, see below), or subtle (triggering the Tor process's internal OOM handler and causing *other* circuits or connections to get closed, or information to get dropped). We'll focus on subtle exhaustion here; for analysis on crashes, see the next section.

Memory exhaustion attacks usually require a little programming skill to exploit, and more if the goal is more subtle than shutting down a server.

The most promising ways to resist memory-exhaustion DoS attacks are:

- Include more types of memory (see list above) among those that the OOM handler is able to free on demand. Those marked with (!) seem most valuable to do first.
- Extend our OOM handler to be triggered by raw memory allocation or by detectable OOM events. This would allow it to *respond* to every kind of RAM exhaustion attack, though it would not necessarily be able to correct every time.

More difficult methods would include:

- Improve our OOM handler to offload offending information to disk rather than deleting it entirely.
- Analyze memory leak bugs discovered in the past, looking for patterns. Preliminary analysis suggests that our biggest antipatterns here are:
  - Forgetting to call free;
  - Using any kind of memory management more complicated than alloc/free pairs;
  - Forgetting to run Coverity right before a release.

Crash attacks Due to programming errors, there may be as-yet-unknown ways to shut down Tor and/or other programs remotely, either with a memory violation, with an assertion failure, or with RAM exhaustion.

Generally, these require that the attacker discover a programming flaw in the program. So long as Tor is actively maintained, these attacks can't be used too often without being detected, since they usually leave a trace of why servers are crashing–either as a stack trace in the log, or an assertion failure message, or both. Thus, they require an up-front investment with a limited shelf life.

Nevertheless, these attacks are problematic when they have the ability to force a user to change guards (see the Sniper Attack paper by Jansen et al.); to mount traffic confirmation attacks; or to attack anonymity in other ways as well.

To resist these attacks better we could:

- Further improve our guard node selection algorithms to resist multiple induced guard failures.
- Consider designs that would allow circuits and their traffic to migrate across different nodes, so as to better survive the collapse of a single node. (HARD!)
- Get stack trace support working on Windows, so we can better detect patterns in Windows crashes.
- Use fewer assertions, and more circuit-level or connection-level error handling.
  - Introduce patterns for handling unexpected bugs more gracefully, possibly modeling them on the BUG_ON() etc macros used by the Linux kernel.
- Analyze memory crash bugs discovered in the past, and search for trends to identify more. Preliminary analysis suggests that our most crash-prone code has historically been:
  - Parsing text-based formats.
  - Parsing binary formats.
  - Failing to enforce checks in one part of the code that are later enforced with an assertion.
  - Hand-coded state machines with unexpected transition modes.
  - Extending the type of an existing field (especially IPv4->IPv6) and elsewhere asserting that the field is the original type.
  - High-complexity functions, especially ones with "you must call A after B" dependencies.
  - Code not currently tested by our integration tests.

Disk space exhaustion attacks Attacks of this kind are most threatening to systems with extremely low disk capacity. To carry one out, an attacker needs a way to force the target to store an unbounded amount of information.

Directory authorities are also a concerning target for disk exhaustion attacks, since they store published relay descriptors on request. To mount a disk exhaustion attack against a non-authority Tor process, an attacker would need either to compromise enough authorities to make the directory grow without bound, to run enough legitimate-looking nodes to make the directory grow without bound, to find an unsuspected bug in Tor's code, or to fill up the disk with log messages.

Our best defenses here are:

- Work harder to make directory authorities throw away descriptors if they become low on disk space.
- Ensure that all of our packages use log rotation.
- Respond to low disk space by diminishing the amount of storage we use.

Bandwidth consumption attacks These attacks are among the most fundamental. A Tor server's job, after all, is to receive bytes, transform them, and retransmit them. Simply by sending data to a Tor server, an attacker forces that server to receive and retransmit the data.

Servers defend themselves against some of these attacks already: by declining to accept data on circuits whose buffers are already too full, and by using fairness metrics to ensure that quieter circuits receive priority service.

The potential impact of one of these attacks can be degraded service from a targeted server. But the most dangerous aspect is that it can interfere with other traffic, introducing patterns to it that can be observed remotely.

Mounting these attacks requires only a botnet running traffic generation code. Exploiting interference is a little trickier, and requires a bit of engineering.

The highest-impact bandwidth consumption attacks are those with a bandwidth multiplier: that is, those in which an attacker can spend only a little bandwidth (and minimal CPU) in order to force the server to spend a lot. These include directory fetches, HTTP requests, and any other traffic pattern where a small request generates a large response.

These attacks are difficult to detect in practice.

Our most promising ways to prevent or mitigate these attacks include:

- Possibly, when bandwidth is scarce, consider prioritizing 1:1 traffic over asymmetric traffic.
- Implement KIST (or something like it) to become more responsive to TCP congestion.
- Implement a scheme like DonnchaC's "split introduction/ rendezvous" proposal (#255) in order to make it harder to congest a hidden service's bandwidth.
- Improve TorFlow (or another bandwidth scanning system) to follow less predictable patterns for bandwidth scanning.
- As above, consider schemes to allow busy circuits to migrate from one path to another.

Socket/port usage attacks TCP reserves around 64000 usable ports per IP address. In practice, the operating system will often have us use many fewer sockets than this. By exhausting the available port count, an attacker can prevent a server from opening new connections to other servers, prevent an exit server from making new exit connections, or prevent a hidden service from serving additional users.

These attacks are less easy to turn into a deanonymization vector than others, except that by reducing service at a guard or exit server, an attacker can force its users to use another. The attacker does not control the next server chosen, though, so unless the client retries indefinitely, the attack has low odds of success.

These attacks require a little engineering to perform optimally, but a less skilled attacker can make a skill/resource tradeoff.

These attacks can be noticed, but are hard to distinguish (currently) from legitimate port/socket exhaustion.

To defend against these attacks, we have a few options:

- Respond to socket/port exhaustion as we currently respond to memory exhaustion: not only refusing to accept new connections, but by looking for long-idle connections that might need to get closed.

  - When looking for connections to close, we could look for suspicious patterns (many connections to the same address, many outbound connections to relays not listed in the consensus, etc). This could be a bad resource use, however, since these patterns are easy to suppress in practice.

- Make better use of systems where we have multiple interfaces to use for outbound connections.

- Make better use of multiple IPv6/IPv4 addresses on a single relay, to increase the number of CPU ports we can use.

- Possibly, deploy some sort of UDP-based transport, to prevent socket exhaustion from interfering with client-relay or relay-relay connections.

Attacks by target

In parallel to considering DoS attacks based on the resource exhausted, we can also distinguish them based on the sort of system that they target.

Attacking clients Performing resource-based DoS attacks against Tor clients serves a couple of possible purposes:

- Traffic confirmation, by forcing a suspected client to degrade performance or to leave the network entirely, and looking to see whether other observed circuits degrade in a corresponding way.

- Traffic shifting, by encouraging users to stop using Tor, thereby receiving less cover themselves and providing less cover than others.

Attacking applications Performing a denial-of-service attack against a (client- or server-side) application can serve an attacker's goals by:

- Encouraging the client to shift to a less-anonymized application

- Gaining a resource-multiplier effect by forcing the client to introduce itself to more nodes on the Tor network.

Attacking hidden services There are a few key opportunities to perform denial-of-service attacks against hidden services:

- Attacking (or becoming) hidden service directories, so clients can't find how to contact a targetted HS.
- Attacking (or becoming) introduction points, so clients can't make contact with a targetted HS
- Attacking a hidden service itself.

Similarly, these attacks can be used either to degrade a HS's performance, or to attempt to mount traffic confirmation against it.

Attacking relays Attacking a relay is at its most powerful when that relay is known to be the guard node for a user of interest, and at its next most powerful when an attacker is running a large number of their own relays and wants to shift traffic onto them.

Attacking directory authorities The directory authorities, thanks to their special position in the Tor network, are an especially concerning attack for DoS attacks. Remove a few authorities, and the voting process becomes less robust; remove enough of them, and the entire network becomes inoperable.

Our best approach here is separating authority functions and isolating their most sensitive duties from their most expensive. (Proposal 257 has more ideas on this topic.)

Attacking other services via Tor By using the Tor network as a vector for performing denial-of-service attacks on other services, an attacker can encourage those services to block or restrict Tor users over time, thereby discouraging Tor usage. This attack can also make it more difficult to host exit nodes.

The best known methodology for resisting these attacks is:

- Developing technology to restrict abusive Tor users without blocking non-abusive ones, and encouraging the use of this technology. (The present author is a fan of various anonymous blacklistable credential systems, but they have yet to be used in practice, and might prove too unwieldy when actually deployed. We should also consider other approaches.)

Attacking project infrastructure All attacks become stronger when engineers can't respond to them. If, instead of targetting the Tor network, an attacker targets the infrastructure that Tor developers use to coordinate, any other attack will be more powerful and successful.

Our best response here is likely to ensure that we have backup means of coordination and communication prepared *before* we need them.

Similarly, an attack against user-facing project infrastructure could prevent us from communicating with users during an emergency, or prevent us from distributing updates effectively.

To resist this kind of attack, we should grow our mirror capacity, and build an emergency response plan for a temporary DoS against our current software distribution methods.

Recommendations

In general, we should favor defenses that close multiple attack vectors simultaneously over defenses that simply close one at a time. Remember: An attacker needs only one attack that works, while we need to prevent *every* good attack.

Therefore, we should look for general solutions and process improvements, rather than piecemeal responses to a single discovered attack at a time.

Solutions with the potential to close many attack vectors at once include:

- Changes to the Tor network's structure to prevent single-server attacks from aiding traffic confirmation attacks. These seem particularly difficult, but quite powerful if we can come up with a design that works.
- Changes to the directory authority infrastructure to remove them (in part or entirely) as a viable DoS target.
- Changes in our coding practices to make it more difficult for programming errors to propagate into large-scale DoS vectors.
- Changes in the code used to handle resource exhaustion should look at the consumed resource directly rather than at indirect counts of it. For example, our OOM-handler should look at the operating system's view of our memory consumption, rather than simply counting an incomplete set of memory consumers.
- Future revisions of our public-key algorithms should consider resource-exhaustion attacks and resource asymmetries.
- Look for areas that have been historically problematic for programming errors, and investigate ways to reduce future errors of this kind.
- Isolate parts of the Tor process to different modules, and try to make crashes or resource exhaustion in any particular part as survivable as possible.