Frank DENIS random thoughts.

On dnscache poisoning and how SipHash can help

Without DNSSEC, the security of the DNS protocol mainly depends on one thing: an attacker shouldn't be able to predict a port number, a 2-byte transaction ID, the destination address, and a question, or else a reply for that question can easily be spoofed.

Spoofing a reply from an authoritative server to a resolver is much more interesting than hijacking a query sent to a client, as the result of the spoofed reply can then stay in the resolver cache for a very long time.

Guessing a question a resolver is going to send to an authoritative server is obviously not an issue if an attacker has the ability to send queries to a recursive resolver.

Guessing what servers this question is going to be sent to is not an issue either, since it's bound to be the authoritative servers for that name, maybe in addition to all required authoritative servers down to root servers.

A port number + transaction ID tuple only offers 32 bit entropy. Actually even less, since port numbers don't use the full 16 bit range. Even if these are truly random, it only takes around $N * 2^{16}$ blind attempts to win the game, N being the number of possible authoritative servers for the target zone.

In order to quickly poison a DNS cache, an attacker needs to force it to send some specific query to authoritative servers.

Names with a very low, and even 0 TTL, are just asking for being hijacked. Unless a DNS resolver has specific defense against this, it only takes a couple minutes to poison a cache so that malicious records are served for a 0 TTL zone. Implementing per-IP rate limiting and enforcing a minimum TTL doesn't defeat this kind of attack, but makes it more time consuming.

Forget about names with a low TTL for a minute. TLDs, in particular, are the best possible target, but they have a decent TTL. Fortunately, we can force dnscache to ignore it.

More than DoS from hash table collisions

Triggering a denial of service attack by causing collisions in a hash table is a problem as old as hash tables themselves. After being under the spotlight last year, tons of applications have switched to some randomized function in order to select a bucket for an entry in a hash table.

In order to defend against a hashdos, a cache can just stop traversing a list after too many unsucessful matches. This is exactly what dnscache has implemented, way before the hashdos issue was under the spotlights.

Once the correct bucket has been found, lookups are only performed on the 100 first entries. It after having scanned 100 slots, no matches have been found, the key is considered nonexistent. Which is totally acceptable for a cache, since this information can be fetched again when needed.

Keys are hashed using the classic, totally deterministic djb33 hash function, which makes it easy to build collisions.

A query for an uncached name requires knowing the IP address of authoritative servers. A query for example.to requires resolving to.cctld.authdns.ripe.net, ns1.iafrica.com, colo.to, tonic.to, and auth02.ns.uu.net. These would usually be present in the cache.

However, adding 100 entries to the same list as tonic.to is enough to force dnscache to issue a query to authoritative servers again. A domain with a wildcard zone is a simple tool to do it. A custom authoritative server can also serve names with glue records leading to the same bucket in order to fill 16 slots in one shot. And generating collisions for the djb33 function is straightforward. For example, all of these names will all hit the

same list as tonic.to no matter what the cache size is, forcing the resolver to send a query to root servers in order to resolve it again as soon as a query for a .to name will be received.

By exploiting a hash table collision, an attacker has no way to trigger a DoS, but he can actually do something way more interesting: force the resolver to send the same query for the same TLD, over and over again, always to the same set of servers, no matter what the intended TTL is and no matter what the cache size is.

And suddenly, poisoining dnscache with a malicious TLD much, much, much easier and faster.

SipHash to the rescue

SipHash is a new pseudo-random function designed by Jean-Philippe Aumasson and Daniel J. Bernstein, optimized for short inputs.

Aumasson and Bernstein propose that hash tables switch to SipHash as a hash function.

SipHash is fast. Almost as fast as CityHash and SpookyHash, but with the huge advantage of being a pseudorandom function.

SipHash uses a 128 bit key and outputs 64 bit values, which seems small and obviously not collision-resistant, but it is more than enough in order to pick a hash table bucket. And the keying obviously protects against intentional hash flooding.

Replacing a deterministic hash function with SipHash requires minor code changes, has minimal overhead, and protects against hash DoS without side effects of other mitigation techniques.

So, check it out, use it and abuse it.

Also check out the siphash patch for djbdns-1.05, a trivial change to let dnscache take avantage of SipHash.

Jump back to the home page

Latest tweets:

- 08 Apr React v15.5.0 https://t.co/VTHF3knoVp
- 07 Apr RT **@RustCrates**: pq (0.1.2) https://t.co/2RiWGmWl6p jq for protobuf
- 07 Apr **@pathcl** Start here -> https://t.co/WMVHoX1Ghw
- 07 Apr RT **@nostarch**: The Rust Programming Language is now in Early Access! https://t.co/ns8KzUWOW8
- 07 Apr **@clementd** Good job! :)
- 07 Apr Spreading love for Rust at OpenDNS: done. At OVH: done. At Fastly: person next to me is reading the Rust book as we speaki :)
- 07 Apr RT **@KingstonTime**: If you use **@firefox** container tabs. Let us know how you browse: https://t.co/wlr7h2YGZQ We ❤ feedback.
- 07 Apr RT **@DefuseSec**: GoFundMe to audit PHP port of libsodium: https://t.co/vj9qhman0Q This port should significantly improve the state of things…
- 07 Apr *#golang* 1.8.1 released https://t.co/y4cM87SgU1
- 07 Apr RT **@jpmens**: Kea DHCP server 1.2.0-beta has been released https://t.co/z4bdcn6FqY

- Download area
- Github zone
- Google me
- Pure-FTPd & friends

Frank Denis (Jedi/Sector One)
blog at pureftpd.org

Follow @jedisct1 on Twitter