Hashed RPZ

Jeroen Massar <jeroen@massar.ch>

New law in Switzerland: ISP block CSAM domains

- Switzerland introduced a new law[1][2][3], active per 2021-01-01, that Access Providers need to block Internet content as per direction of the Swiss Federal Police (FedPol) for the purpose of blocking CSAM.
- The blocking is domain based, the list of domains is distributed from FedPol by fetching from a not-in-DNS password-based SFTP server, multiple lists each in a password protected RAR file.

[1] https://www.bakom.admin.ch/bakom/de/home/das-bakom/organisation/rechtliche-grundlagen/bundesgesetze/fmg-revision-2017/revision-fmg-verordnungen.html
[2] https://www.fedlex.admin.ch/eli/cc/2007/166/en
[3] https://www.fedlex.admin.ch/eli/cc/1997/2187_2187_2187/en

Standard Solution: RPZ

- RPZ ("Response Policy Zones") aka DNS Firewalls by Paul Vixie and Vernon Schryver[1], implemented in most DNS recursor server software.

- Used for blocking of unwanted content (ads, malware, etc.), by encoding hostnames into a normal DNS zone with actions on the right hand.

- Examples (as left hand sides inside a zone, for instance blocklist.rpz.example.org):

  - `www.bad.example.net CNAME .` => causes NXDOMAIN

  - `www.wall.example.com CNAME wall.example.org.` => causes a faking of A/AAAA and thus for instance for HTTP going to the new address (but with HTTPS being prevalent typically a SSL/TLS error)

- Thus for instance for the Swiss "Casino Law"[2] we create a RPZ zone (as ESBK/COMLOT do not provide one) and configure recursors to block using that.
  People going to https://casino.example.com get redirected to the blocking page which is served with a wrong SSL cert as we do not have that one.
  Bypass is also easy by configuring different DNS server, but >95% use ours[3].

[1] https://dnsrpz.info
[2] https://www.fedlex.admin.ch/eli/cc/2018/795/de
[3] https://stats.labs.apnic.net/rvrs

Problems with access to cleartext lists of illegal content

- Having access to the list, which one has a copy of due to AXFR means you would know the URLs of the illegal content. And in this case content that is socially very controversial and where one want to remain far away from.

- For debugging, troubleshooting or maintenance, one could accidentally already stumble upon the content of the list.

- Backups of systems would also create a backup of these zones and thus store them in another location, more possibility to exposure.

What I want to avoid...

- I don't want to cause a random engineer who maybe just does maintenance or some troubleshooting and thus not knows about the type of content, be accidentally exposed to the contents of the list due to possible legal and personal consequences.

- "They have this link on their computer, they must have been looking at it!"

- .... one does not want to show up in the news cycle for that, it destroys careers and people.

Hashed RPZ

- Instead of having the left hand sides in clear text, we hash them. To preserve the possibility of wildcard matching we hash per component.

- The algorithm is thus relatively simple:

    - Split the left hand side by label.

    - Then for each component from right to left:

        - If the component is a wildcard (*), keep it verbatim (unhashed).

        - Hash with BLAKE3 keyed, but as a complete domain up to that point; hashing size is 4, 8 or 16 bytes depending on input length (this to keep output length short, as base32hex causes the length to double thus some left hands sides might not fit).

    - Output the hash using base32hex lowercase (RFC4648)

- For `www.example.com` would be: `hash(www.example.com) + '.' + hash(example.com) + '.' + hash(com)`

- PS: This is not a new idea: with PowerDNS Recursor one can use Lua to do MD5 over the left hand sides as has been used by many providers.

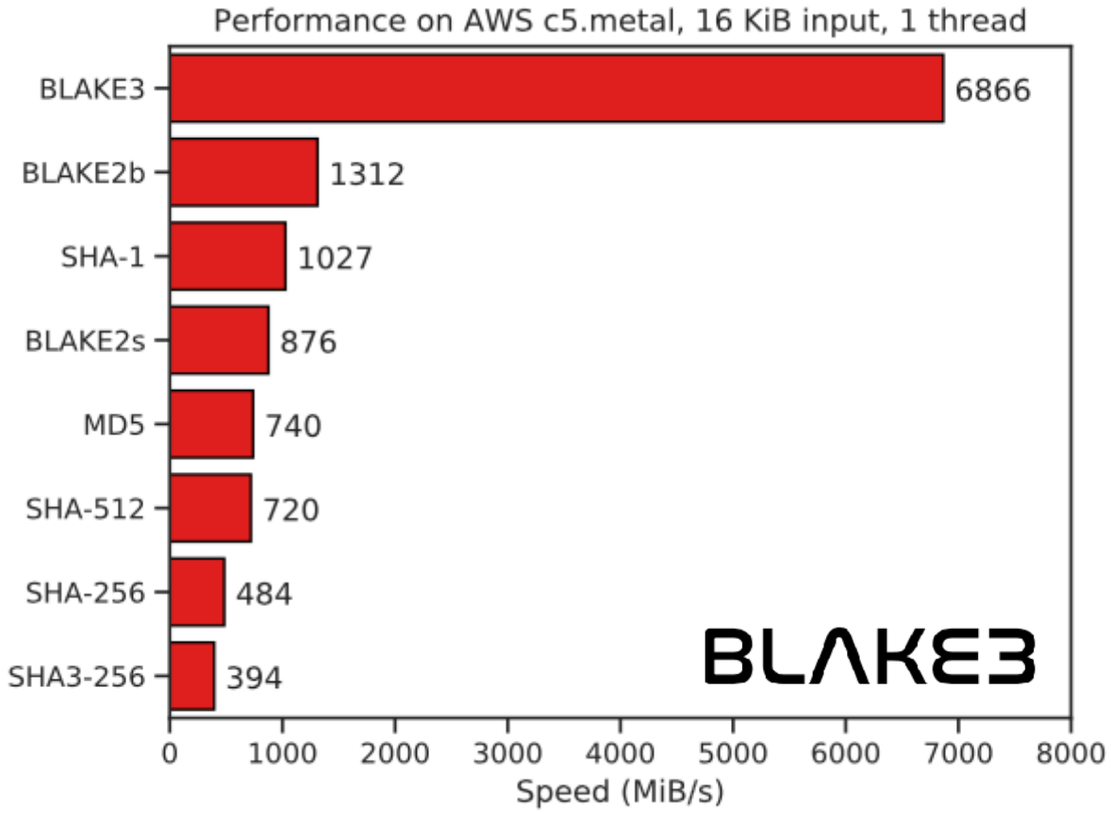
[1] https://github.com/massar/hashedrpz

BLAKE3

- Don't roll your own crypto: thus I trust people like Zooko, Jean-Philippe Aumasson, Samuel Neves & Jack O'Conner for their excellent cryptographic work.

- BLAKE3 is secure, fast, applicable for short strings (like DNS labels).

- BLAKE3 can be keyed with a passphrase thus rainbow-style attacks are mostly futile.



[1] https://github.com/BLAKE3-team/BLAKE3

Key Selection & Distribution

- The BLAKE3 hashing key is composed of two parts:
    - In-band: included in-band public in cleartext in the zone as a TXT record (_rpzhashkey.<domain>); thus allowing the producer of the hash to rotate the key often and distribute it to consumers.
    - OOB: as per-configuration of the RPZ zone, distributed OOB as part of RPZ zone setup, suggested: include source name.
- One should of course choose a long (64 char+) complex one, preferable generated with proper randomness.
- Due to in-band key rotation and the unknown, it becomes extremely tough to be able to use rainbow-style attacks against the list.

Examples

- Input:

```
www.example.net
one.example.com
two.example.com
```

- Output:

```
9mgrvf8.qa4gjtuvuia82ubhh705n29hm0.0hjg4h0
fca618e.r939194s2f5m5rdougo4rvc0gg.u32p0s0
w21jice.r939194s2f5m5rdougo4rvc0gg.u32p0s0
```

- The example.com portion is thus hashed the same, but a different TLD causes 'example' not to match.

- Even though 'www' is common, it won't ever hash the same.

Putting it together

- A restricted VM (Full Disk Encrypted with password in a safe), that does not listen on the network except for SSH, does not allow non-restricted SSH login after boot or even to do upgrades (one can generate a new image to do updates). At boot it asks for the FDE key to unlock the root filesystem and then the PGP key to decrypt the configuration file with SFTP/RAR passwords. All intermediate files are stored in memory (tmpfs).

- This system has a crontab that fetches the RARs from SFTP and hashes each entry and produces a RPZ zone file.

- Our DNS backend system then fetches the hashed zone over SFTP with a very restricted account that can only fetch the zone file and then loads that up in our DNS system, which distributes to the recursors.

- Future: ask FEDPOL to generate the hashed zone file instead of us.

(Somebody could do VM memory introspection etc, but then you are actively looking for the something illegal...
-- dedicated hardware could partially address these kind of attacks, though Cold Boot Attacks exist ;)

Implementation: Noqoshu

- For quicker implementation and as we had other requirements already: extended an existing DNS recursive server in golang called Noqoshu (Somali for "to return").

- Not published as Open Source yet, but soon... not intended to compete with existing recursors that are very feature rich, stable and well tested.

- The real hard work is done by Miek Gieben's Golang DNS library[1].

- Uses OSRG GoBGP[2] for integrated anycast support.

- At Cache-Miss (~10% of queries), before looking up the real result, a RPZ lookup is performed (Whitelists, Blacklists[Casino,ESBK,COMLOT] or the HashedRPZ list for FEDPOL, at which point we hash the left hand side with the keys for that zone with [3] and verify if the entry is in zone.

- Need to still run flamethrower[4] over it for performance tests (time, so little of it, even if you don't travel) and do a lot of docs etc.

[1] https://github.com/miekg/dns
[2] https://github.com/osrg/gobgp/
[3] https://github.com/massar/hashedrpz
[4] https://github.com/DNS-OARC/flamethrower

Future Work

- Integrate feedback from operators and developers.
- IETF Draft to standardise this method so that interoperable code can be made.
- More implementations (read: provide patches for unbound / pdns-recursor / knot resolver / bind9)
- More optimisations & testing.
- Attempting to get FEDPOL to do the producing of the RPZ zone, so that they can effectively publicly distribute the zone file, and consumers never ever have to see the cleartext version of the list (oversight becomes a question of course)

With HashedRPZ we keep engineers safe!

(and who does not love mashed potatoes???)

Jeroen Massar
<jeroen@massar.ch>

Bonus Slides

Extra answers about the Dutch Naming System

Why per-label Hash?

From a discussion with Peter van Dijk (habbie/PowerDNS), there are effectively three options:

- 1) **hash(www).hash(example).hash(com)** leaks more than you want (can see example in example.com == example.net, due to hash), but has efficient lookups

- 2) **hash(www.example.com)** leaks as little as possible, but has expensive lookups (can't have a tree structure in your lookups, all has to be in a single table)

- 3) **hash(www.example.com).hash(example.com).hash(com)** leaks very little (one can see a domain has multiple entries), but has efficient lookups.

Hashing is cheap in comparison, and for each sub-domain one hash to hash anyway. One can thus progressively hash, lookup, hash, lookup.

Option 3) is what HashedRPZ uses.

Including the origindomain?

I had a thought about including the origin domain, thus instead of:

- hash(www.example.com).hash(example.com).hash(com)

- hash(www.example.com.origindomain).hash(example.com.origindomain).hash(com.origindomain)

While this will produce 'more salt', the key already provides a decent amount of salt.

It would "lock" the ownernames to that zone, thus disallowing records to be easily transferred from say rpz.example.org to rpz.example.com as the origin would be included in the calculation.

Doesn't blocking suck?

- Yes, I personally rather also not do it because it breaks many more things, but it is the law even though many people tried to explain the disadvantages, the voting people did not get that information, nor will they understand how DNS works, let alone what it is.

- 90% of customers tend to keep/use the ISP DHCP-provided DNS recursive: thus for most customers it will cause an effect. Thus DNS-based blocking is great for malware/phishing blocking as it breaks those sites and typically there is nothing else on it. SafeBrowsing takes swift care in many cases too though, and actually provides a proper readable warning that avoids helpdesk calls. DNS blocking though is 'intransparent' for the user: a TXT record or similar for the blocking reason would be helpful to have, if that signal could reach the UI of the customer.

- Customers that want to access the content anyway can circumvent ISP-provided DNS based blocks easily configure DNS servers under other jurisdictions (e.g. 8.8.8.8 is provided not by a Swiss Internet Access Provider thus does not have to block things) or a VPN of many kinds or even Tor.

- Due to HTTPS and TLS in general, 'redirecting' (RPZ CNAME) to a blocking server results in SSL certicate errors.

- Sometimes protocols get broken, eg. SMTP mail delivery (thus can't send a contact mail to the site either), as the blocking server only supports HTTP(S) and not SMTP (the Internet is more than just the web...).

- Many of these problems (especially the casino law) thus result in extra helpdesk calls... => helpdesk has a tool for checking if a domain is on and which blocking list; though they need to know the domain, which the customer has to provide.

Running Code?

- Code: https://github.com/massar/hashedrpz

- Docs: https://pkg.go.dev/github.com/massar/hashedrpz

```
package main

import (
	"fmt"
	"github.com/massar/hashedrpz"
)

function main() {
	h := hashedrpz.New("teststring")

	o, err := h.Hash("host.example.net", 200)
	if err != nil {
		fmt.Printf("Hashing gave error %s", err)
		return
	}

	fmt.Printf("Hashed to:\n%s\n", o)
	return
}
```

Performance?

2012 iMac 27" i7-2600 @ 3.4Ghz

```
$ go test -bench .
goos: darwin
goarch: amd64
pkg: github.com/massar/hashedrpz
BenchmarkHashTests-8              281691              3832 ns/op
BenchmarkHashMany-8                40305             29439 ns/op
BenchmarkHashSimple-8             384877              3063 ns/op
BenchmarkHash10M-8              10000000              5190 ns/op
                                                         3 avg.#labels
                                                        23 avg.length
                                                      2368 toolong
                                                        38 wrongwildcard
PASS
ok      github.com/massar/hashedrpz     55.744s
```

2018 MBP 13" i7-8559U @ 2.7Ghz

```
$ go test -bench .
goos: darwin
goarch: amd64
pkg: github.com/massar/hashedrpz
BenchmarkHashTests-8        1672880               688 ns/op
BenchmarkHashMany-8          196862              5762 ns/op
BenchmarkHashSimple-8       2139783               577 ns/op
BenchmarkHash10M-8         10000000              1163 ns/op
                                                    3 avg.#labels
                                                 23.0 avg.length
                                                 2368 toolong
                                                   38 wrongwildcard
PASS
ok     github.com/massar/hashedrpz       16.575s
```

The example implementation has golang based test & benchmark code.

As can be seen, average domains based on DNS-OARC 10M list from 2012 uses 5190ns/ownername on a very trusty machine from that year, but the 6 year newer machine is already 60% faster...

No more mashed potatoes... (final slide)