RBL Evaluation Methodology

ICANN Office of the Chief Technology Officer

Siôn Lloyd
Carlos Hernández-Gañán
Samaneh Tajalizadehkhoob

OCTO-037v2
11 December 2023

TABLE OF CONTENTS

This document supports ICANN’s strategic goal to improve the assessment of, and responsiveness to, new technologies that impact the security, stability, and resiliency of the Internet’s unique identifier systems through greater engagement with relevant parties. It is part of ICANN’s strategic objective to evolve the unique identifier systems in coordination and collaboration with relevant parties to continue to serve the needs of the global Internet user base.

This document is a minor revision to OCTO-037v1 to correct a mistake in the author attribution on the cover page.

This document is part of ICANN’s Office of the Chief Technical Officer (OCTO) document series. Please see the OCTO publication page for a list of documents in the series. If you have questions or suggestions on any of these documents, please send them to octo@icann.org.

Executive Summary

Reputation Block Lists (RBLs) serve as a common defense mechanism against harmful and unwanted Internet content. These lists contain the Internet Protocol (IP) addresses, domain names, or full Uniform Resource Locators (URLs) of known spam sources, phishing pages, malicious sites, or other unwanted content. By using RBLs to block incoming or outgoing access, Internet service providers, email providers, and other organizations can offer some level of protection for their users from online threats. They are also used for academic research and as training sets for machine learning models.

To help evaluate and understand the characteristics of RBLs, this document covers a set of metrics that can be used to help evaluate them both in isolation and in combination with other sources. These metrics, used within the Office of the Chief Technology Officer (OCTO) of the Internet Corporation for Assigned Names and Numbers organization (ICANN org), include RBL focus, mechanics, metadata, volume, overlap, timeliness, and churn. We categorize the metrics into four groups: a general description; metrics that can be directly measured; metrics that can be indirectly measured; and metrics that can only be discovered secondhand. We also show the structure we use to store different RBLs in a consistent manner, which simplifies the process of comparing RBLs and using them as a single data source.

We in OCTO use the processes when we look at new data sources to add to our set of RBLs, it helps us to make a judgment on whether the value of the data is commensurate with the cost. We also plan on using these measures to periodically reevaluate our data sources, particularly when we need to renew their licenses, to make sure that they are still performing as we would expect.

Our experience tells us that there is no “one size fits all” when it comes to using RBLs. We argue that understanding the strengths and weaknesses of any particular RBL, or combination of multiple RBLs, is key to getting a good fit for a particular use-case. To maximize the benefit of RBLs, we suggest combining two or more to get more comprehensive protection than can be provided by any single RBL.

1 Introduction

Domain name and IP address reputation lists are used to identify and block potentially harmful or unwanted traffic on the Internet. The earliest known reputation list was created by Paul Vixie in the 1990s, and was called the "Real-time Blackhole List" or RBL. The meaning of an “RBL” has evolved, and is described below. This list contained the IP addresses of known spam sources and was used by mail servers to block incoming email from those sources.

Over time, similar lists were created for other types of online activities, such as domain or URL reputation lists for identifying phishing or malware-related websites, and IP address reputation lists for identifying sources of malware or other online threats. Today, these lists are widely used by Internet service providers, email providers, and other organizations to help protect their users from online threats. They continue to evolve and improve as new threats emerge and new technologies are developed to combat them.

These sources are commonly known as “Reputation Block Lists” or RBLs. However, they are often referred to by slightly different names like “threat intelligence,” “security feeds,” “abuse feeds,” among other similar names. They can contain different identifier types: domain names, IP addresses or full URLs, and in many cases a mixture of two or more identifier types. They can also specialize in particular threat types, like spam, phishing, malware, etc.; or they may contain a mixture of multiple threat types. They can differ in collection methodology, licensing, distribution method, intended use, and almost every other conceivable way.

There are many examples of RBLs being used in many different scenarios, some more obvious than others. For example, services like Google Safe Browsing (https://developers.google.com/safe-browsing) can be thought of as an RBL protecting a browser user from known phishing sites.

The academic community also makes use of RBLs to understand the current and historical reputation of domain names in various types of analysis. We in OCTO see studies to measure security threat concentrations within the Internet intermediaries such as top-level domain (TLD) registry, registrars, or hosting providers, and to assess mitigation strategies of Internet intermediaries.

In many cases, the use of this data is not aligned with how the producers intend it to be used; and so, its suitability may not be clear. In other cases, the conclusions drawn from the analysis based on this data do not necessarily reflect the specifications and limitations of the data. Moreover, for all use cases it is hard to know if the RBL being used is the best fit, if there is a better option, or if a combination of two or more RBLs would add enough benefit to justify any extra cost. It is worth noting here that the cost can be in terms of time and complexity, as well as financial; even free, open-source feeds have some cost associated with them.

Misalignment with the intended use can have a significant impact on a project. For example, an RBL which contains low confidence or non-vetted entries could result in an appreciable number of incorrect entries, known as false positives. Such a data feed might be perfectly acceptable if used to protect a small network where the mitigation of incorrect entries has a low associated cost. However, the same RBL may not be suitable for an application where a false positive results in a time and resource consuming investigation.

There is also a more basic issue for any organization wishing to read multiple RBL sources, but use them as a single, unified data feed. The different formats and contents need to be harmonized into one structure that can fairly represent all the constituent parts without losing important features from the original data.

This work was presented at the Anti-Phishing Working Group’s “technical summit and researchers sync-up 2023,” which was held in Dublin. A version of this paper will be published as part of the proceedings from that event.

2 Aims

Given the context introduced above, in this document we outline our method to evaluate and characterize any given RBL, not just in isolation, but also in how multiple RBLs complement one another. We look at the general description of an RBL; things we can measure directly; things that we can make approximations of, and things that we can only discover secondhand. We also discuss the implications and limitations of these measurements.

We will not discuss here the steps required to read RBL data as this will vary between RBLs. We do, however, show in Appendix B the data structure that we use to harmonize data into a single, consistent format. All of the RBL data we read is written into this structure, although it has had to evolve as new RBLs with new fields have been added, and it will likely continue to evolve as new RBLs or requirements are added.

This work has been informed by earlier examples (see Appendix A), but we have kept or modified parts of their suggested method to suit our requirements. As such, our approach is grounded in the projects that we have been involved with, and other parties with other experiences may well have different metrics that they regard as important. This work is also based on the sources that we are already familiar with; it is likely that other RBLs have features which will require modifications to this method.

There are some things that we are explicitly not trying to measure. We are not looking to put a score on an RBL, or say that one is demonstrably better than another. We want to increase our understanding of RBL data used regularly by us and our community, so that we can either use them with confidence, or understand why they are not suitable for a particular project. We also do not consider cost or licensing terms here, although they could be significant factors in any decision on whether to use an RBL. Lastly, we are not aiming to evaluate the absolute effectiveness of the RBLs, which is something that some of the work referenced in Appendix A has done.

In brief then, in this document we aim to answer the following questions:

- How can we characterize an RBL, and what measurements can we make?
- How well do multiple RBLs combine, and what does a particular RBL add?

As mentioned above, we separate our metrics into four categories: the general description, metrics we can measure directly, those we can measure indirectly (e.g., via sampling), and those we can only discover secondhand (e.g., from frequently asked questions (FAQs)).

2.1 The General Description

These are characteristics that we need to know before we can start to ingest data into our system. It may be a feature that initially brought the RBL to our interest, perhaps to fill an identified gap in our other RBLs. We also include details that we need to know during the integration of an RBL into our system, like how it is distributed and what data it contains.

- RBL focus – What entry types does it contain (spam, phish, etc.)?
- RBL mechanics – what delivery mechanism is used, does it provide the whole list or a stream of new entries, etc.?
- Metadata – does the RBL provide more information, like malware family, phished brand, etc.?
- Entry provenance – do entries come from observations, or are they “probabilistic”?

2.2 What We Can Measure Directly

These are the measurements we can make based on the data in the RBL.

- Volume – how many entries are present?
- Overlap – how many entries are in common with other RBLs?
- Timeliness – how quickly do entries appear?
- Churn – how dynamic are the entries?

2.3 What We Can Measure Indirectly

These are metrics that we measure indirectly from the data: where we may have to sample the data to get an approximation of the answer, or where we have surrogate measurements in lieu of what we actually want to investigate.

- Liveliness – how many entries are "active"?
- Purity – can we see potential false positives?
- Accuracy – does the categorization and metadata look correct?

2.4 What the Data Cannot Tell Us

These are characteristics which we cannot derive from the data itself, but rely on second-hand information or longer experience. For example we look at the documentation for the RBL, consult FAQs, or talk to the RBL providers to get this information.

- Catchment – are there geographic biases, collection method gaps (e.g., no mobile data), etc.?
- Entry retesting – are statuses periodically reconfirmed?
- Reliability – is the data always available or are there issues transferring?

3 Method Detail

Looking in more detail at the metrics outlined above, we will show what we measure and, where appropriate, how we use visualizations.

3.1 The General Description

3.1.1 RBL Focus

Perhaps, the first thing to consider is the threat types that the RBL contains. Does it focus on a single threat type or contain multiple types? How does this relate to any other RBLs in our set, does it fill a known gap? How does it relate to the expected use?

3.1.2 RBL Mechanics

A prosaic and significant issue is how we read the RBL and merge it into our larger dataset. We need to understand what delivery mechanism is used, is there an API, do we get the data in CSV, JSON, etc. Note, there are two types of RBLs that need to be understood, related to what data is retrieved when we read the RBL. Some RBLs provide a snapshot of the whole current list on every read; while others provide a stream of new entries (a list of "point in time" observations). When dealing with the latter case, the decision on how long an entry remains active is one for us to decide, rather than relying on the provider removing it. The decision also has the effect of requiring us to maintain state for that source, so that we know the period covered by our last successful read in order to not miss entries. Of course, we may miss data on RBLs that provide the complete list, e.g., an entry is added after we read it but removed before we refresh it.

3.1.3 Metadata

It can be useful to have context around a particular entry, and some RBLs provide more information than others, like a timestamp the entry was added, the malware family seen, the brand being phished, and so on. Another useful data point is whether the entry is believed to be a malicious registration or a compromised but otherwise legitimate registration.
All of this forms the metadata of an observation.

3.1.4 Entry Provenance

Some RBLs have entries based on observations, for example from analyzing email being delivered to a spam trap. Others use statistical methods like machine learning to mark entities as "suspicious" or "likely to be misused." Some RBLs may even mix these two sorts of entry in ways that can make it hard to determine the reasons for their inclusion.

Entries included, without direct observation, can be problematic for some use cases, especially where a certain level of evidence is required. If an RBL contains this sort of entry, and does not clearly indicate those entries, then it may be considered to be unsuitable for some projects.

3.2 What We Can Measure Directly

3.2.1 Volume

Possibly the easiest measurement to make is how many entries are present, although some care needs to be taken to make sure that the same thing is being measured in each case. For example, some RBLs contain just domains while others may contain URLs, but of course multiple URLs may well map to a single domain. We look at unique entries over a period of time, preferably a month or more, to give as good a representation as possible. This is particularly significant for those RBLs that provide a stream of new entries and therefore do not have a concept of a "current list." If we look at unique domains with no other filters, we see something like Figure 1. We can also produce similar figures but show unique hosts, URLs, entries broken down by different threat types, etc.

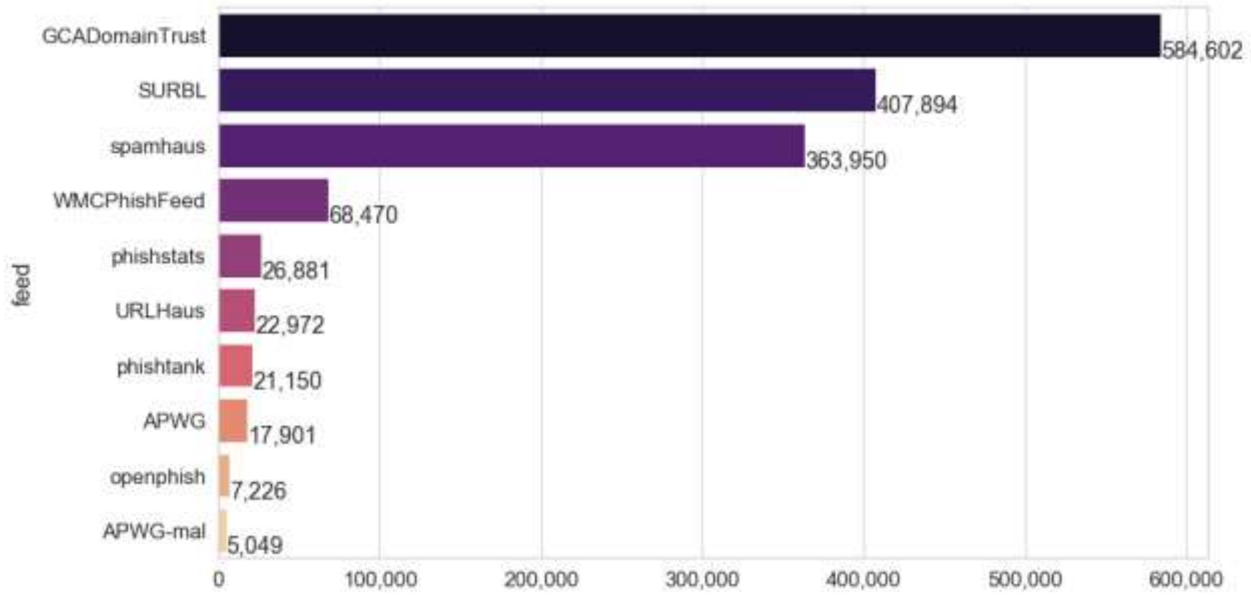
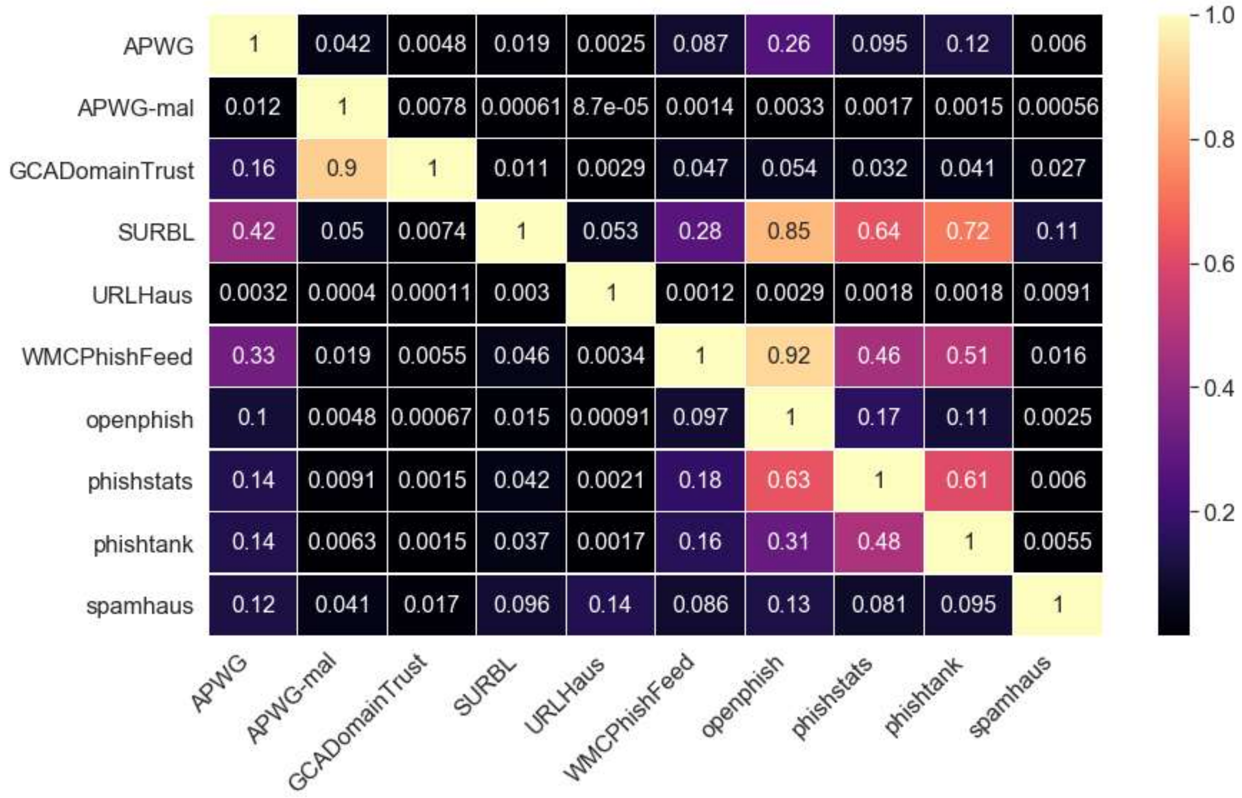


Figure 1. Plot of counts of unique domains per RBL for all entries in February 2023.

Higher volumes are, in general, desirable. This is not, however, the whole story. For example DGArchive (https://dgarchive.caad.fkie.fraunhofer.de), which is not shown above, contains over two million entries per month. That particular data is based on enumeration of domain generation algorithms, and so the majority of those entries may never be registered. It is therefore arguable that we are not comparing like with like to other RBLs; we look to address issues like these later on. It is also true that some threats are more serious or active than others. Therefore some entries offer more "value."

3.2.2 Overlap

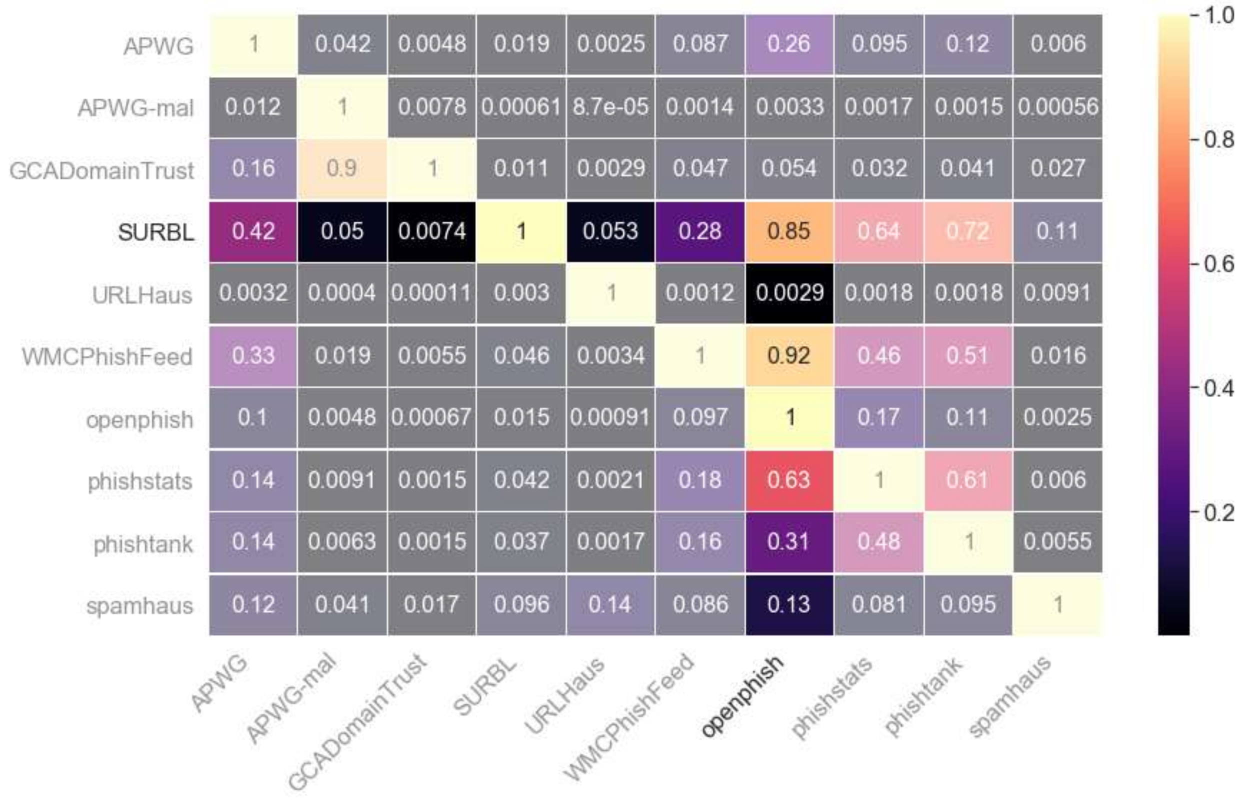
If we are looking to add a new RBL into our existing set, it is interesting to see how many entries are in common with our current data. One simple measure is the overlap of unique domains, shown in Figure 2a. Again, we need to aggregate over a period of time and be careful to compare like with like. It may also be instructive to see different threat types separately, especially if we are looking to fill a specific gap.



	APWG	APWG-mal	GCADomainTrust	SURBL	URLHaus	WMCPhishFeed	openphish	phishstats	phishtank	spamhaus
APWG	1	0.042	0.0048	0.019	0.0025	0.087	0.26	0.095	0.12	0.006
APWG-mal	0.012	1	0.0078	0.00061	8.7e-05	0.0014	0.0033	0.0017	0.0015	0.00056
GCADomainTrust	0.16	0.9	1	0.011	0.0029	0.047	0.054	0.032	0.041	0.027
SURBL	0.42	0.05	0.0074	1	0.053	0.28	0.85	0.64	0.72	0.11
URLHaus	0.0032	0.0004	0.00011	0.003	1	0.0012	0.0029	0.0018	0.0018	0.0091
WMCPhishFeed	0.33	0.019	0.0055	0.046	0.0034	1	0.92	0.46	0.51	0.016
openphish	0.1	0.0048	0.00067	0.015	0.00091	0.097	1	0.17	0.11	0.0025
phishstats	0.14	0.0091	0.0015	0.042	0.0021	0.18	0.63	1	0.61	0.006
phishtank	0.14	0.0063	0.0015	0.037	0.0017	0.16	0.31	0.48	1	0.0055
spamhaus	0.12	0.041	0.017	0.096	0.14	0.086	0.13	0.081	0.095	1

Figure 2a. Heatmap showing proportions of domains seen in common between different RBLs.

This shows us how much of one RBL is contained within another (and vice versa) -- it can be a bit complicated to understand at first glance. Reading across the rows we see how much the RBL listed on the y-axis contains of the RBL listed on the x-axis. For example, if we take the row for SURBL we can see that SURBL contains 0.85 (85%) of openphish, see Figure 2b.



	APWG	APWG-mal	GCADomainTrust	SURBL	URLHaus	WMCPhishFeed	openphish	phishstats	phishtank	spamhaus
APWG	1	0.042	0.0048	0.019	0.0025	0.087	0.26	0.095	0.12	0.006
APWG-mal	0.012	1	0.0078	0.00061	8.7e-05	0.0014	0.0033	0.0017	0.0015	0.00056
GCADomainTrust	0.16	0.9	1	0.011	0.0029	0.047	0.054	0.032	0.041	0.027
SURBL	0.42	0.05	0.0074	1	0.053	0.28	0.85	0.64	0.72	0.11
URLHaus	0.0032	0.0004	0.00011	0.003	1	0.0012	0.0029	0.0018	0.0018	0.0091
WMCPhishFeed	0.33	0.019	0.0055	0.046	0.0034	1	0.92	0.46	0.51	0.016
openphish	0.1	0.0048	0.00067	0.015	0.00091	0.097	1	0.17	0.11	0.0025
phishstats	0.14	0.0091	0.0015	0.042	0.0021	0.18	0.63	1	0.61	0.006
phishtank	0.14	0.0063	0.0015	0.037	0.0017	0.16	0.31	0.48	1	0.0055
spamhaus	0.12	0.041	0.017	0.096	0.14	0.086	0.13	0.081	0.095	1

Figure 2b. Reading across from SURBL we see how much of the other RBLs it contains; the highlight shows it in relation to openphish.

However, openphish contains just 0.015 (1.5%) of SURBL, see Figure 2c. While the absolute number of domains in common is the same, the difference is the underlying size of the RBL (we saw above that SURBL is substantially larger than openphish).

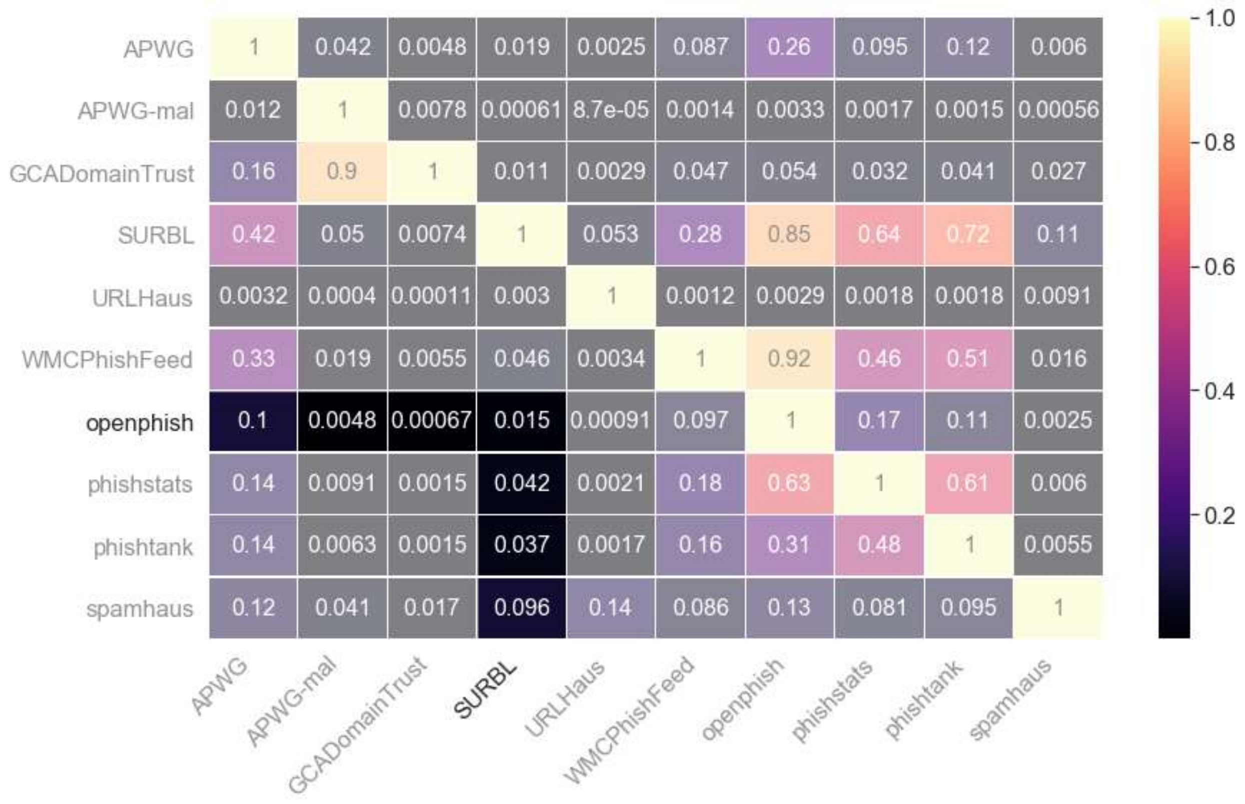


Figure 2c. The reverse lookup to Figure 2b, how much of SURBL is contained in openphish.

The overlap view shows us some interesting features. While the majority of overlaps are small, less than about 5%, there are some which are much higher. This is where open sources are being read and incorporated into other RBLs, presumably after being validated to the required standard for that RBL. These cases could be significant if entries on multiple RBLs are being taken as independent observations, when they may in fact stem from a single original source.

3.2.3 Timeliness

The view above is shows some cross-pollination between RBLs. The next question is “where two or more RBLs have the same entry, which gets the data earlier and by how much?” To this end, we look at the time delta between an entry appearing on a “base feed” that we are considering and the other RBL(s). This is represented visually in Figure 3.

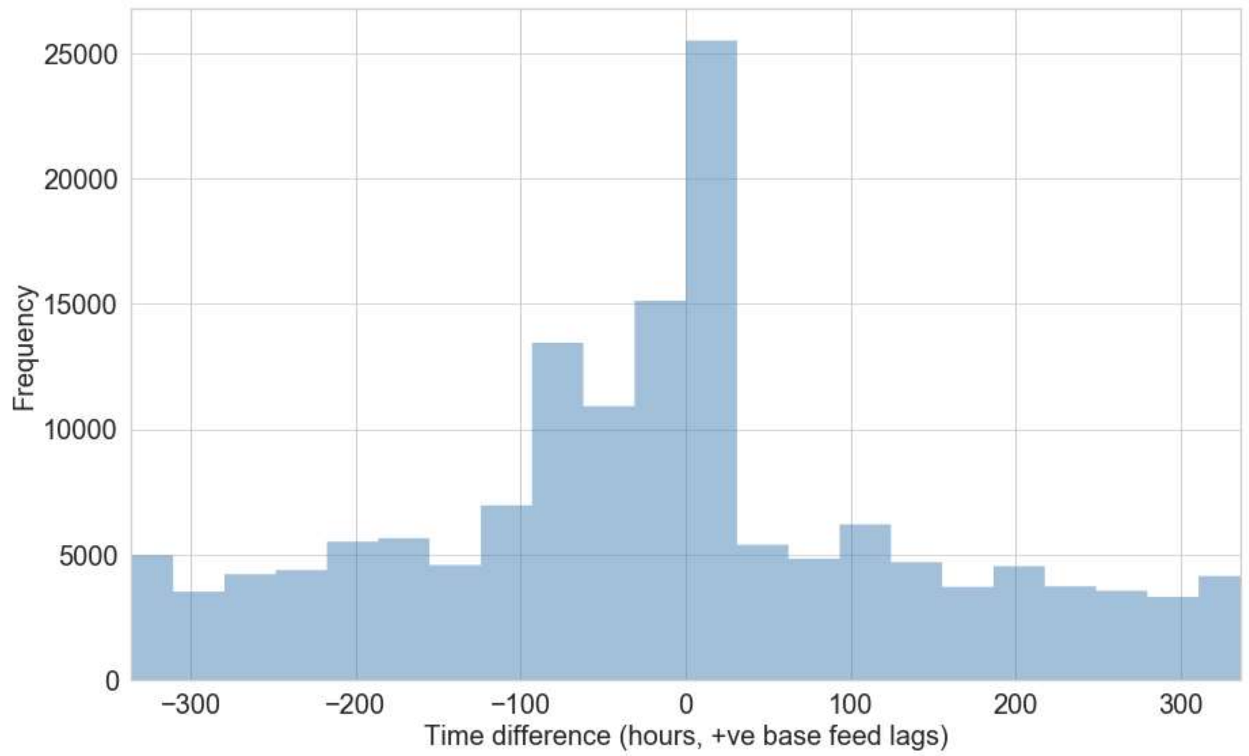


Figure 3. Time difference between entries appearing on one RBL and any other in our set.

In Figure 3, entries with a negative time are where the base feed leads other RBLs, whereas a positive time indicates it lagging behind. Ideally, we want to see more weight to the left of the graph indicating that the RBL being considered is consistently getting entries earlier than others. There is some ambiguity which is introduced from entries that come and go from an RBL multiple times. We also have effects caused by the edges of the time window we use. For example, when a domain is seen earlier than our time window in a streaming RBL, it does not appear in our data.

3.2.4 Churn

For the RBLs that provide their whole current set of entries on each read, it is also useful to know how dynamic the list is. If an RBL's volume stays the same as on the previous read, is it because the list is static or is it because as many entries are being removed as are being added? To this end, we consider a single RBL over a period of time and plot its volume, along with the number of new and removed entries, as shown in Figure 4.

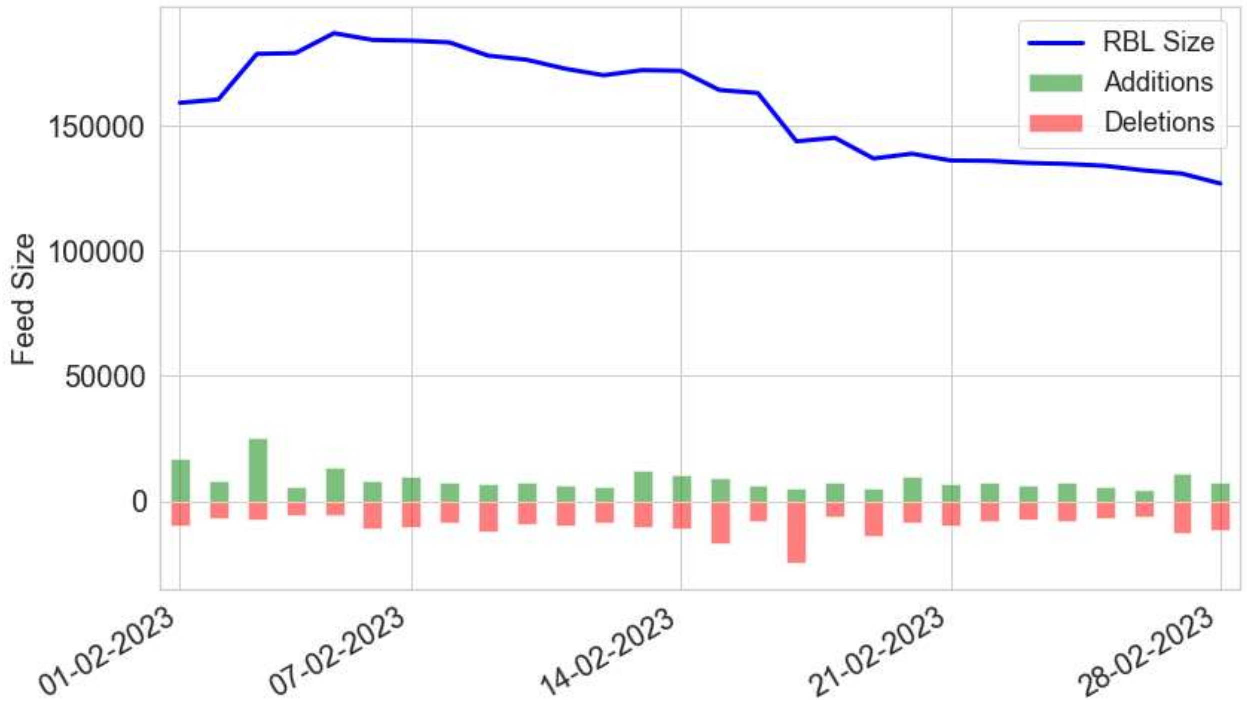


Figure 4. Daily volumes plus additions and deletions for a single RBL over a month.

It is worth remembering that removing stale entries, which are no longer active threats, can be as important as adding new entries, but that is often an afterthought. To this end, we also look at the histogram of the ages of entries as seen in Figure 5 (note the log y-scale). This particular plot shows a healthy mix where the majority of entries have a short lifespan of days or weeks, with only a small number being on the RBL for a year or more.

This analysis gives us more insight into how active the RBL is, how many new threats are being added, and how many old entries are being removed. A higher churn reflects a more active RBL and is seen as a positive feature.

For those feeds which just provide “point in time” observations this analysis is not relevant, although we can still look at the volumes of new threats being added.

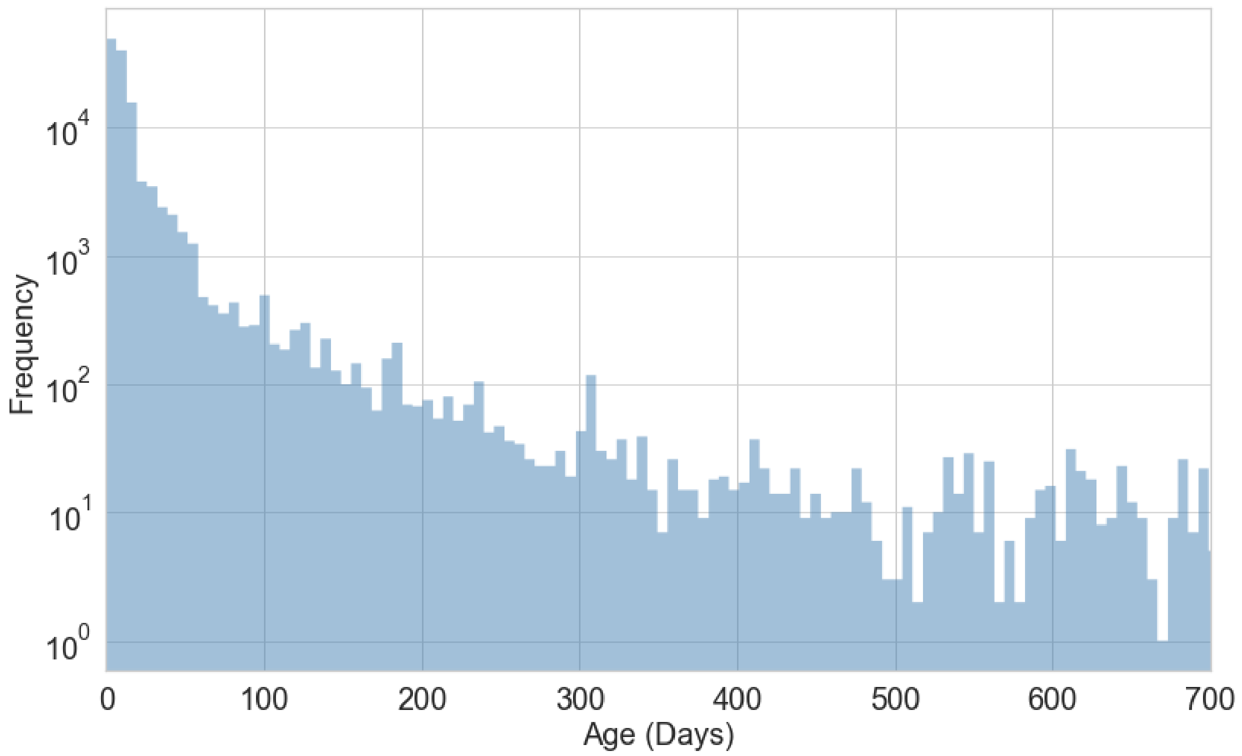


Figure 5. Ages of entries for a single RBL, note the log scale on the y-axis.

3.3 What We Can Measure Indirectly

3.3.1 Liveliness

Above, we measured the volume of entries on an RBL, but without looking more closely we do not know how many of those entries are "active." There may be entries which no longer resolve or have been mitigated in other ways. For example, some registrars take control of a domain and "park" it in order to remove abusive content.

The time taken to capture this information for every entry on a sizable RBL is prohibitive; and once we had finished we would need to start again to catch any new entries or changes in existing ones. One way to tackle this issue, and get a snapshot of the data, is to pick a random sample of sufficient size to be representative of the whole population, and measure that.

If we see a large proportion of the entries not resolving then we need to think why this might be. While one reason may be that the RBL has stale information, there may be other explanations. For example, maybe the RBL includes the output from one or more domain generation algorithms (DGAs), many of which are never actually registered.

In Table 1 we see results from two RBLs where we look at how many resolve, and of those how many appear to be parked. The method for making this determination will be covered in an upcoming publication.

Status	Feed 1	Feed 2
NXDOMAIN	253	664
NS no IP	74	27
Resolved	673	309
Resolved (Parked)	37	23
Total	1,000	1,000

Table 1. Measurements of liveliness for two RBLs, taking a sample of 1,000 domains from each.

3.3.2 Purity

One of the more serious issues for RBLs is when they contain false positive reports, that is they contain entries that are not, and never have been, malicious. These entries are nearly impossible to discover *en masse*, they will only really become apparent through close examination. Even then it is possible that the malicious content is hidden and only visible with particular connection characteristics, like a certain user-agent string or referral path. However, we try to discover potential issues ahead of time.

One thing we would like to look at is the overlap between the RBL and a source of positive, or good, reputation. We are not aware of such a list, however, we use a surrogate source – a list of popular domains, like the TRANCO top 1M (although not all of it). While these domains may still be malicious, they are less likely to be. Also, for uses like blocking network traffic, any entry in the top 10,000 would potentially be very disruptive if replaced by an NXDOMAIN response.

We obviously want this metric to be as low as possible, and where we suspect false positives we would like to understand if there are explanations or mitigations we can use. To take DGAs as an example again, short DGA domains may coincidentally overlap with real words and legitimate registrations. To make this less of an issue, it may be that only DGA domains with seven or more characters are retained; the cost of any false positives will determine the amount of acceptable risk and hence tune this cutoff.

3.3.3 Accuracy

Where an RBL provides extra metadata, like categorizations, do we believe that they are correct? Where we see entries in common between different RBLs, do they agree? This can be difficult to pin down as we do see the same entity reported for different threat types within the same feed; so, again, we need to sample and check to get an idea of the scale of any issues.

We would like to be able to trust all the data that an RBL provides, not just the presence of entries, and the misclassification of entries can have serious consequences in some cases. For example, if an RBL has a low accuracy in terms of the metadata we may not be able to use it to generate statistics.

3.4 What the Data Cannot Tell Us

3.4.1 Catchment

RBLs have different collection mechanisms, even though some are aggregates of multiple primary sources. This will end up giving the RBL strengths and biases, which could be geographic or delivery related; for example, they may have no mobile data, no visibility of threats targeted at specific countries, etc.

Explanations of these collection mechanisms can sometimes be found in FAQs, white papers, conversations with the providers, or other secondhand methods. In many cases, however, the amount of information is, for sound operational reasons, limited.

We may need this information to identify RBLs that fill gaps in our current set; for other uses it may be that data biased toward a particular locale is actually desirable.

3.4.2 Entry Retesting

We have seen that entries are removed from RBLs, but we cannot, from our measurements, definitively say why. Are statuses of entries being periodically reconfirmed, or are they just timed out? Some RBLs provide this information, but most do not, and deciding how long we trust entries can be influenced by how this is being handled by the RBL.

Ideally all entries are frequently retested, but we appreciate that operationally this may be impossible.

3.4.3 Reliability

Something that will only become apparent with continued monitoring and use is whether the data is always available or whether there are sometimes issues transferring? This can influence our confidence in using an RBL in a production environment, since, if we have our own service-level agreements (SLAs), then the RBL should have something at least similar but preferably better.

For open-source RBLs with no contract (and therefore no SLA) then only our experience with the RBL can give us this confidence.

4 Conclusions

Combining all these pieces gives us a good idea of the characteristics of an RBL and how it complements our current data set. This style of combining is a process we go through when considering adding new sources to our set, and can inform us whether a source is worth continuing with or not.

However, to understand and match RBLs that are suitable for specific projects, we need to also understand the project requirements. RBL characteristics, and how multiple RBLs interact with each other, is only one side of the story.

Therefore, it is not generally possible to simply state that some RBLs are better than others. However, it can be the case that some are more suited to specific projects than others.

That said, from what we have seen of the RBLs we have access to, adding multiple sources increases the number of unique entities included and hence the comprehensiveness of the data.

It is also possible that, for whatever reason, the time frame investigated is particularly good or bad for the RBL in question. We try to counter that by having a broad window of a calendar month. However, if an RBL happens to have visibility of a large campaign in that time, it might reflect an unusually positive impression.

While we outlined our evaluation processes in this document, we want to emphasize the fact that these measures are not meant to be complete or prescriptive. Our processes have evolved based on our use cases. It is quite likely that future projects, or new RBLs, will suggest new measures and modifications to existing ones. To be clear, the output of the process is a report that does not score the RBL, but gives us the information needed to decide whether to invest the resources required to incorporate the RBL into our system.

Appendix A: References

Papers related to the evaluation or characterization of RBLs:

S. Sinha, M. Bailey, F. Jahanian, Shades of grey: On the effectiveness of reputation-based “blacklists”, in: 2008 3rd International Conference on Malicious and Unwanted Software (MALWARE), 2008, pp. 57–64.
doi:10.1109/MALWARE.2008.4690858

A. Pitsillidis, C. Kanich, G. M. Voelker, K. Levchenko, S. Savage, Taster’s choice: A comparative analysis of spam feeds,
In: Proceedings of the 2012 Internet Measurement Conference, IMC ’12, Association for Computing Machinery, New York, NY, USA, 2012, p. 427–440.
doi: 10.1145/2398776.2398821

J. Zhang, A. Chivukula, M. Bailey, M. Karir, M. Liu, Characterization of blacklists and tainted network traffic,
In: M. Roughan, R. Chang (Eds.), Passive and Active Measurement, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 218–228.

M. Kührer, C. Rossow, T. Holz, Paint it black: Evaluating the effectiveness of malware blacklists, in: Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17, Springer, 2014, pp. 1–21.

T. Vissers, P. Janssen, W. Joosen, L. Desmet, Assessing the effectiveness of domain black-listing against malicious dns registrations,
In: 2019 IEEE Security and Privacy Workshops (SPW), 2019, pp. 199–204.
doi: 10.1109/SPW.2019.00045

V. G. Li, M. Dunn, P. Pearce, D. McCoy, G. M. Voelker, S. Savage, Reading the tea leaves: A comparative analysis of threat intelligence,
In: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, 2019, pp. 851–867.

URL: https://www.usenix.org/conference/usenixsecurity19/presentation/li

S. Ramanathan, J. Mirkovic, M. Yu, Blag: Improving the accuracy of black-lists,
NDSS (2020).
URL: https://par.nsf.gov/biblio/10205652
Doi: 10.14722/ndss.2020.24232

P. Vallina, V. Le Pochat, A. Feal, M. Paraschiv, J. Gamba, T. Burke, O. Hohlfeld, J. Tapiador, N. Vallina-Rodriguez,
Mis-shapes, mistakes, misfits: An analysis of domain classification services,
in: Proceedings of the ACM Internet Measurement Conference, IMC ’20, Association for Computing Machinery, New York, NY, USA, 2020, p. 598–618.
doi: 10.1145/3419394.3423660

Appendix B: Database Schema

We write all of our RBL data to a single database table per month; most sources are read daily, some more frequently.

Some processing is required for most entries to be written to these tables, for example URLs are turned into domains, also the TLD and suffix are extracted. This means we can get a more consistent view across all of our RBLs, coping with those which provide different fields or use slightly different terminology.

Our schema currently looks like this:

Column Name	Type	Notes
report_date	date	Some RBLs tell us, for others report date is when we read that RBL.
domain	text	Stripped domain name
feed	text	Which source it came from
reason	text	Spam, phishing, etc.
full_identifier	text	Some RBLs give URLs or include subdomains.
score	int	Some RBLs give a confidence score
suffix	text	Suffix according to the public suffix list
tld	text	tld
tld_type	text	ccTLD or gTLD

Column Name	Type	Notes
registrar	text	If known, often NULL
reg_id	int	Registrar ID, if known
seen_since	timestamp	When did this entry first appear on the RBL
url_shortener	boolean	Is it on our list of known URL shorteners (e.g., bit.ly)
sub_feed	text	Some RBLs aggregate other sources, if this is the case those sources will be here
notes	text	Any other info the RBL gave that might be useful. Will depend on the RBL
dga	boolean	Is the entry from a domain generation algorithm, if known
ip	boolean	Is the entry an IP address

Appendix C: Code for Plots

The plots displayed in this document were all created from real data using Python. In this appendix we show the relevant parts of the code OCTO created. We have removed some details of the database connection, but show the SQL used to retrieve the data, any post-processing, and the plotting code.

C.1 Setup and Configuration

We have the following imports.

```
# Imports
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import datetime, timedelta
# You will also need to import your DB connector
```

Also some configurations, note we have abstracted the database connection.

```
# CONFIG OPTIONS ##
# Set the date - either today or some fixed date, you choose
#theDate = datetime.today()
theDate = datetime.strptime('2023-02', '%Y-%m')

# Sometimes we need the data in a different format
chartDate = theDate.strftime('%Y-%m') # String for displaying on graphs
dbMon = theDate.strftime('%Y%m') # String for using in the DB table

BASE_FEED = 'spamhaus' # Feed to concentrate on, for individual plots or those
comparisons which are 1:many

# Connect to the database; use your connector here to get a cursor object
dataBase = DBconnector([CONFIGURATION_DETAILS])
cur = dataBase.conn.cursor()

# The tables we'll be selecting from
tablename = 'feeds.daily_all_'+dbMon # Table of RBL data
```

We want barcharts with values on them.

```
# A helper function to add values to barchart
def values_on_bars(axs, h_gap=0.5, v_gap=0.0):
    for p in axs.patches:
        _x = p.get_x() + p.get_width() + float(h_gap)
        _y = p.get_y() + p.get_height() - float(v_gap)
        value = int(p.get_width())
        axs.text(_x, _y, '{:,}'.format(value), ha="left")
```

C.2 Code for Figure 1

The code generates a simple bar chart, but with values displayed on the bars.

```
# Overlaps sql
overlaps_sql = 'select distinct domain, feed from '+tablename
column_names = ['domain','feed']

# Example extra clause
overlaps_sql += ' where reason = "phishing"'

# Get the data
cur.execute(overlaps_sql)
res = cur.fetchall()

df = pd.DataFrame(res, columns=column_names)

plt.rcParams.update({'font.size': 20})

gby = df.groupby(["feed"])
s = gby.size().sort_values(ascending=False)
sns.set(style="whitegrid", font_scale=1.5, rc={"figure.figsize": (14, 8)})
ax = sns.barplot(orient='h', x=s, y=s.index, palette="magma")
ax.xaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))
ax.set(title="Barplot showing the count of distinct domain entries per feed - %s\n" %
(chartDate))

# To anonymise we just turn off the axis labels
y_axis = ax.axes.get_yaxis()
#y_axis.set_visible(False)
values_on_bars(ax, 80, 0.2)
plt.show()
```

C.3 Code for Figure 2

This code was used to create the RBL overlap heatmap.

```
# Basic data is the same as the previous plots
# Get list of RBLs
rbls = list(df.feed.unique())

# Create a matrix of which domains are in which RBLs
rep_matrix = pd.crosstab(df.domain, df.feed)

# This will be our heatmap with X contains Y and X is contained in Y
tmp = pd.DataFrame({'contains': rbls, 'is contained': rbls})
heat = pd.crosstab(tmp['contains'], tmp['is contained'])

# Loop over each combination of RBLs, avoid duplicates for efficiency
for first in rbls[:-1]:
    for second in rbls[rbls.index(first)+1:]:

        # how many domains are common? I.e. both lists have a value of 1
        tmp2 = rep_matrix.eq(1, axis='index')[[first, second]].all(1)
        common = tmp2[tmp2.values == True].count()

        # The number of domains in each
        first_sum = rep_matrix[first].sum()
        second_sum = rep_matrix[second].sum()

        # proportions of overlap vs each list size
        heat[first].loc[second] = common/first_sum
        heat[second].loc[first] = common/second_sum

# ANNOTATE - Set to True if actual value should be written in each heatmap cell
ANNOTATE=True

sns.set(style="whitegrid", font_scale=1.5, rc={"figure.figsize": (14, 8)})
ax = sns.heatmap(heat, linewidths=.2, annot_kws={'size': 16}, annot=ANNOTATE,
cmap="magma")
ax.set(title="Heatmap showing overlap between RBLs - %s\n" % (chartDate))
ax.set(xlabel="", ylabel="")
plt.xticks(rotation=45, horizontalalignment='right')
plt.show()
```

C.4 Code for Figure 3

The following code was used for pulling data for lead-lag times for overlapping entries.

```
# Timings SQL - time we first see entries compared to our base feed
time_sql = ("with tmp as (select domain, feed, seen_since from "+tablename+" "
       "where feed <> '"+BASE_FEED+"' group by domain, feed, seen_since) "
       "select distinct r.domain, r.seen_since, tmp.feed, tmp.seen_since "
       "from "+tablename+" r, tmp "
       "where r.feed = '"+BASE_FEED+"' and r.domain = tmp.domain")
time_cols = ['domain','base_since','feed','feed_since']

# Get the data
cur.execute(time_sql)
res = cur.fetchall()
```

```
time_df = pd.DataFrame(res, columns=time_cols)

# We need to convert
time_df['base_since'] = time_df['base_since'].astype("datetime64")
time_df['feed_since'] = time_df['feed_since'].astype("datetime64")

# Take the diff in hours
time_df['diff'] = (time_df['base_since'] -
time_df['feed_since']).astype('timedelta64[h]')

data = np.array(time_df['diff'])

FONTSIZE = 20

plt.figure(figsize=(13,8))
plt.hist(data, bins=30000, alpha=0.5,
         histtype='stepfilled', color='steelblue',
         edgecolor='none')
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Time difference (hours, +ve base feed lags)', fontsize=FONTSIZE)
plt.ylabel('Frequency', fontsize=FONTSIZE)
plt.xlim([-336,336])
plt.xticks(size=FONTSIZE)
plt.yticks(size=FONTSIZE)
plt.title('Histogram of Base feed lead/lag', fontsize=FONTSIZE)
plt.tight_layout()
```

C.5 Code for Figure 4

The following code pulls data on daily additions and deletions from a single RBL.

```
# We need two different datasets here; first up the feed size per day
# We also need to include previous months final day
lastDay = (theDate - timedelta(theDate.day))
lastDBMon = lastDay.strftime('%Y%m')
lastTablename = 'feeds.daily_all_'+lastDBMon
lastDBDay = lastDay.strftime('%Y-%m-%d')
size_sql = ("select report_date, count(1) from "+lastTablename+" "
       "where feed = '"+BASE_FEED+"' and report_date = '"+lastDBDay+"' "
       "group by report_date "
       "union "
       "select report_date, count(1) from "+tablename+" "
       "where feed = '"+BASE_FEED+"' group by report_date")
size_cols = ['Date','Count']

# How many new entries per day
add_sql = ("select report_date, count(1) from "+tablename+" "
       "where feed = '"+BASE_FEED+"' and report_date = seen_since "
       "group by report_date")
add_cols = ['Date','Additions']

# Get those datasets

# Daily Counts
counts = {}
cur.execute(size_sql)
rows = cur.fetchall()
for row in rows:
       counts[row[0].strftime('%Y-%m-%d')] = row[1]
```

```
# Get daily additions, calculate deletions from daily count diffs and additions
l=[]
cur.execute(add_sql)
rows = cur.fetchall()
for row in rows:
       today = row[0].strftime('%Y-%m-%d')
       yesterday = (row[0] - timedelta(1)).strftime('%Y-%m-%d')
       deletions = counts[today] - counts[yesterday] - row[1]
       l.append([today,counts[today],row[1],deletions])

# Put that into a frame
df_add = pd.DataFrame(l, columns=['Date','Count','Additions','Deletions'])
df_add['Date'] = df_add['Date'].astype("datetime64")
df_add = df_add.sort_values(by=['Date'])

# Plot the data
FONTSIZE = 20

plt.figure(figsize=(13,8))
plt.rcParams.update({'font.size': FONTSIZE})

plt.bar(df_add['Date'], df_add['Additions'], width=0.5, alpha=0.5, facecolor="green",
label='Additions')
plt.bar(df_add['Date'], df_add['Deletions'], width=0.5, alpha=0.5, facecolor="red",
label='Deletions')

plt.plot(df_add['Date'], df_add['Count'], color="blue", label="RBL Size", linewidth=3)

ax = plt.gca()
ax.xaxis.set_major_locator(mdates.DayLocator([1,7,14,21,28]))
ax.xaxis.set_minor_locator(mdates.DayLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y'))
plt.gcf().autofmt_xdate() # Rotation

plt.ylabel('Feed Size', fontsize=FONTSIZE)
plt.xticks(size=FONTSIZE)
plt.yticks(size=FONTSIZE)
plt.title('Daily RBL Churn for '+BASE_FEED, fontsize=FONTSIZE)
plt.legend(fontsize=18)
plt.show()
```

C.6 Code for Figure 5

This code filters for ages (in days) of entries for a single RBL. Note that the SQL below is not fully portable, because of the “timestampdiff” function. It works in Snowflake and probably MySQL (but this was not tested). In Postgres, for example, the “date_part” function could be used.

```
# Ages of entries -
ages_sql = ("with tmp as (select max(report_date) as max from "+tablename+" where feed
= '"+BASE_FEED+"') "
       "select timestampdiff('day', seen_since, report_date) as age "
       "      from "+tablename+",tmp where feed = '"+BASE_FEED+"' and report_date =
tmp.max")
ages_cols = ['Age']

# Get the data
cur.execute(ages_sql)
res = cur.fetchall()
```

```
df_ages = pd.DataFrame(res, columns=ages_cols)

data = np.array(df_ages['Age'])

# Plot the data
FONTSIZE = 20

plt.figure(figsize=(13,8))
plt.rcParams.update({'font.size': 14})
plt.hist(data, bins=3000, alpha=0.5,
         histtype='stepfilled', color='steelblue',
         edgecolor='none')
plt.grid(axis='y', alpha=0.75)
plt.xticks(size=FONTSIZE)
plt.yticks(size=FONTSIZE)
plt.xlabel('Age (Days)', fontsize=FONTSIZE)
plt.ylabel('Frequency', fontsize=FONTSIZE)
plt.yscale('log') # In our data the y-axis has a large range; so we use a log scale.
plt.title('Histogram of Feed Entry Age for '+BASE_FEED, fontsize=FONTSIZE)
```