

Malware vs Big Data

Frank Denis - @jedisct1

OpenDNS - Umbrella Security Labs

<http://opendns.com> - <http://umbrella.com>

What is DNS?

DNS is an old, still not well-defined but inherently insecure protocol, that obsolete and fucked up implementations, retarded firewalls and drunk users make even worse.

Oh, and it's super critical, too.

When it breaks, the Interwebz break.

It will be fixed. Eventually. Ya know, like SMTP. Or, like, after IPv6 has been deployed everywhere. Like, after BCP 38 has also been deployed everywhere. Oh, look! A unicorn!

Why use an alternative DNS resolver?

- Because most DNS resolvers run by ISPs suck.
- Because some governments use DNS for censorship.
- Because DNS is a cheap and easy way to track users.
- OpenDNS: 208.67.222.222 & 208.67.220.220, ports 53 and 443.

Why use OpenDNS?

Fast

We have machines in 20 datacenters with 20 Gb/s bandwidth minimum everywhere, we use anycast, we peer with a lot of people, we have an operations team with mad skillz at optimizing routes for super-low latency.

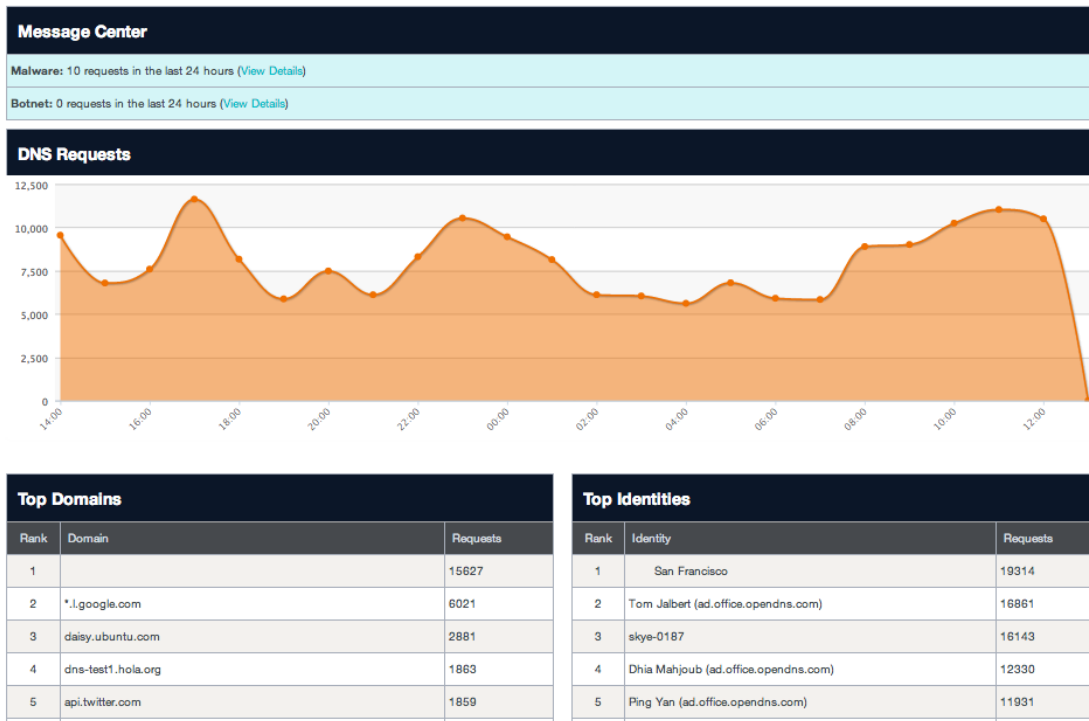
Reliable

100% uptime since 2007, at least from a user perspective (not from a Nagios perspective).

Not only DNS

We are running VPN servers, too. With the same features as what we offer for DNS.

Stats



Domain & category blocking

- Policies
- Identities
- Policy Settings**
- Domain Lists
- Category Settings**
- Security Settings
- Block Page Settings
- System Settings

Category Settings + add a new category setting

!smoke

!smoke

- High**
Blocks adult-related sites, illegal activity, social networking sites, video sharing sites, and general time-wasters.
- Moderate**
Blocks all adult-related websites and illegal activity.
- Low**
Blocks pornography.
- None**
- Custom**

<input type="checkbox"/> Academic Fraud
<input type="checkbox"/> Adult Themes
<input type="checkbox"/> Adware
<input type="checkbox"/> Alcohol
<input type="checkbox"/> Anime/Manga/Webcomic
<input type="checkbox"/> Auctions
<input type="checkbox"/> Automotive
<input type="checkbox"/> Blogs
<input type="checkbox"/> Business Services

Look 'ma. (Almost) no proxy required!

</marketing>

Security

In addition to generic categories, we can block internet threats (but you have to give us \$\$\$ for that. Please proceed):

- Phishing, fraud
- Malware (botnets, ransomware, bitcoin mining...)
- APT (actually no, you don't get it in the package).

A typical infection

Infectors

It all begins with some exciting spam. Or a Facebook app. Or some shit on Craigslist. Or an ad banner on a benign web site. Or a Google result for “download Justin Bieber MP3”.

Or a XSS vulnerability. Because these can do more than `alert(document.cookie)`. Like, injecting an `iframe`.

Exploitation

Browser exploits, Flash, PDF, Java. Even smart people can be p0wn3d by a 0day. Multiple intermediaries between an infector and the actual exploit are common.

Payload

The actual shit that is going to constantly run on your machine and that will make your life miserable.

That shit is either downloaded directly, or through an installer, involving more intermediaries

You've been p0wn3d, now what?

Ransomware: you will have to pay in order to recover your data (or not).

Scam: you will have to pay for a fake antivirus.

Botnet: welcome to the community. You're now going to send massive amounts of spam and help with DDoSing some random dudes.

Click fraud: this is your lucky day. Your browser has a new cute toolbar that hijacks all your search results and offers cures for your erectile disfunction.

Keyloggers / banking trojans / file scrapers / bitcoin miners: thanks for your donations.

Targetted attacks: if you happen to work on a nuclear weapon, shit is going to hit the nuclear fan.

And more! Endless fun!

Command and conquer^{H^H^H^H}trol

This is how botnet operators are going to mess with your computer.

- DGAs: infected clients are going to send queries to pseudorandom domain names / URLs, derived from the current date.
- Infected clients can also try to connect to a hard-coded (sometimes huge) list of URLs, some of them being totally benign, just to confuse antiviruses (and our algorithms as well). Oh, IRC is still cool as a C&C, too.
- Malware authors are getting more and more creative and are now also taking advantage of Google Docs, Google Search, Twitter, Pastebin and abusing various protocols.

OMG, this is running in the cloud!

This is a highly scalable, fault-tolerant architecture.

Each piece of the puzzle can easily be moved/removed/updated.

Exploits and payloads are constantly repacked in order to bypass antiviruses.

By the time a sample is found and analyzed, it's already too late.

Exploit kits rule the cybercrime world, and more diversity is not required.

Exploit kits are sufficiently advanced to give a lot of work to researchers already. All they need is an update from time to time, to take advantage of new Java^H^H^H^H vulnerabilities.

Our approach

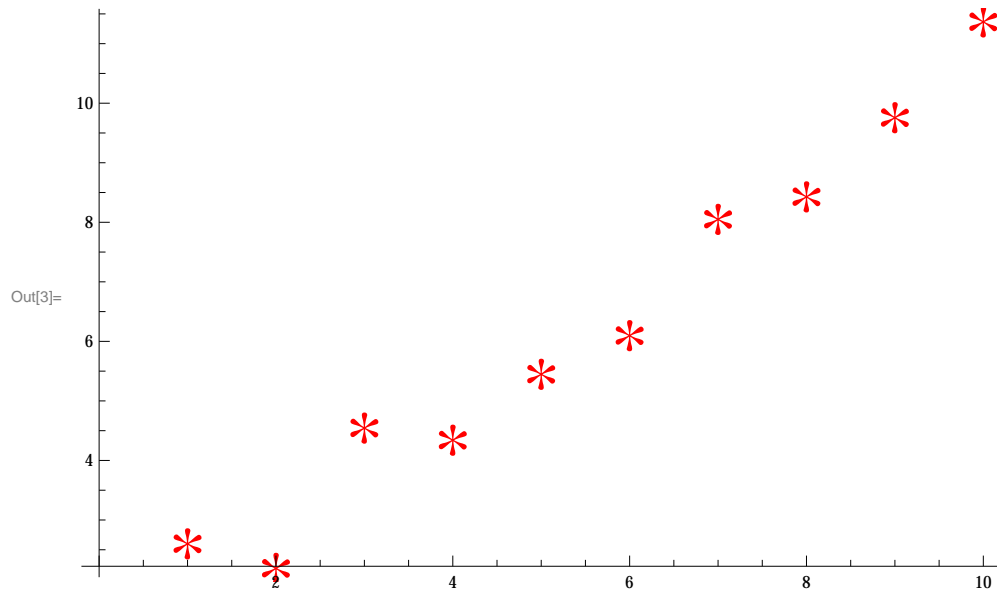
Antivirus vendors / malware researchers are doing an excellent job.

But we take a different, **complementary** approach, leveraging the massive amount of traffic that our DNS resolvers are seeing.

Not a new idea in academic papers, and “predictive analytics” has become a cool marketing buzzword. But we’re doing it with real data, for real customers, in a constantly evolving landscape.

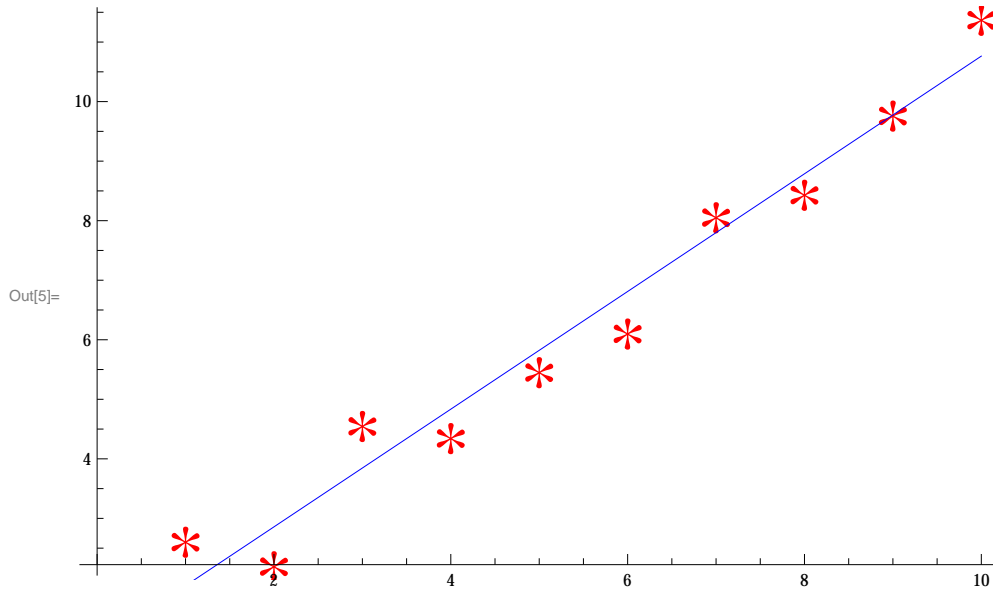
Linear regression

```
In[1]:= SeedRandom[1]
data = Table[{i, i + RandomReal[{0, 2}]}, {i, 1, 10}]
Show[ListPlot[data, PlotStyle -> Red, PlotMarkers -> {"*", 42}],
ImageSize -> Scaled[0.75]]
```



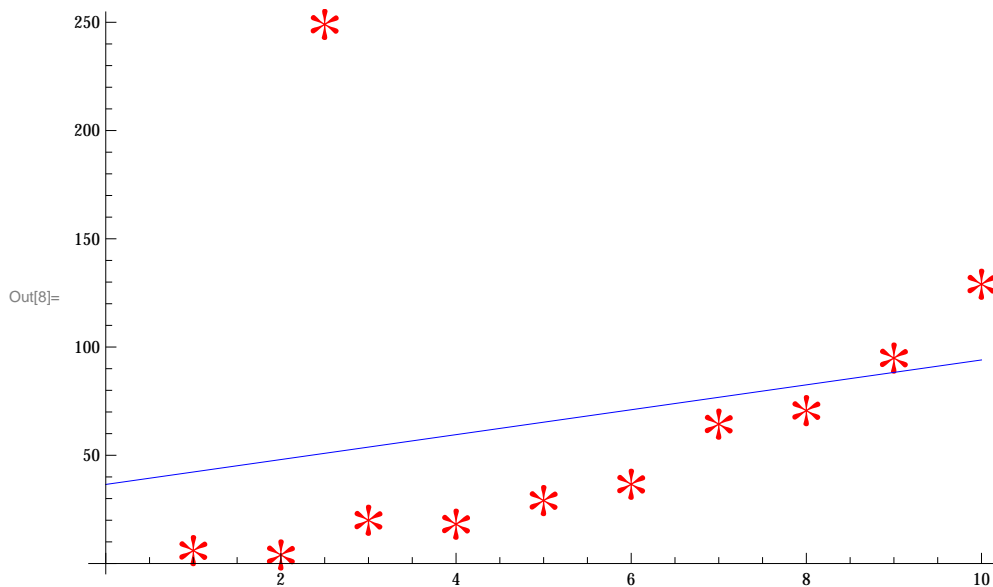
```
In[4]:= model = LinearModelFit[data, {x}, x]
Show[ListPlot[data, PlotStyle → Red, PlotMarkers → {"*", 42}],
Plot[model[x], {x, 0, 10}, PlotStyle → Blue], ImageSize → Scaled[0.75]]
```

```
Out[4]= FittedModel[ $0.880909 + 0.988312x$ ]
```



```
In[6]:= data2 = Append[#[[1]], #[[2]]^2 & /@data, {2.5, 250}]
model2 = LinearModelFit[data2, {x}, x]
Show[ListPlot[data2, PlotStyle → Red, PlotMarkers → {"*", 42}],
Plot[model2[x], {x, 0, 10}, PlotStyle → Blue], ImageSize → Scaled[0.75]]
```

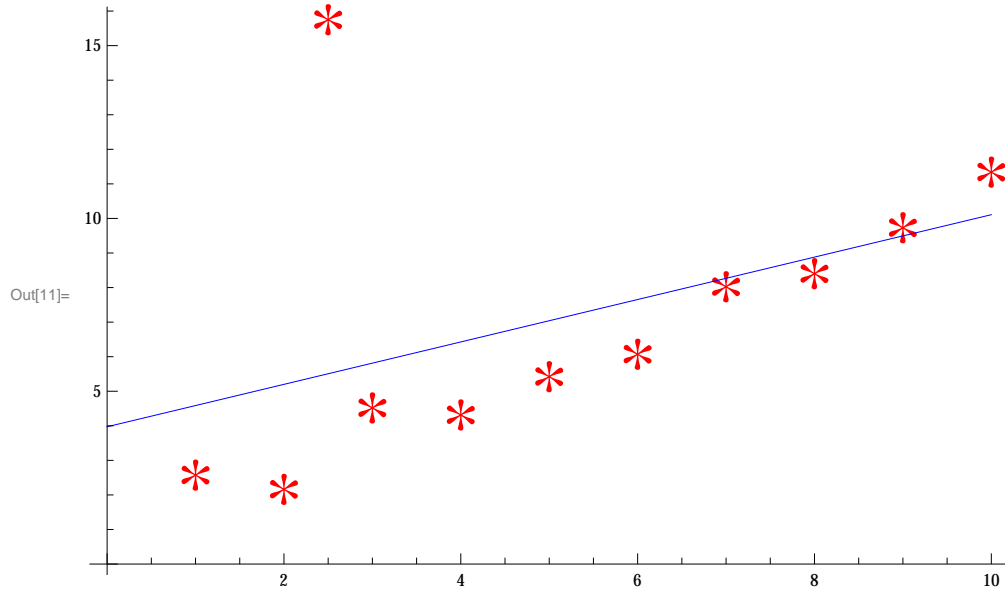
```
Out[7]= FittedModel[ $36.5318 + 5.75022x$ ]
```



Please note that throwing more data wouldn't help. We need to change the model and/or preprocess the training set.

```
In[9]:= data3 = {#[[1]], Sqrt#[[2]]} & /@ data2
(* data3 = Delete[data3, 11] *)
model3 = LinearModelFit[data3, {x}, x]
Show[ListPlot[data3, PlotStyle -> Red, PlotMarkers -> {"*", 42}],
Plot[model3[x], {x, 0, 10}, PlotStyle -> Blue], ImageSize -> Scaled[0.75]]
```

Out[10]= FittedModel [3.97241 + 0.613584 x]



In[15]=

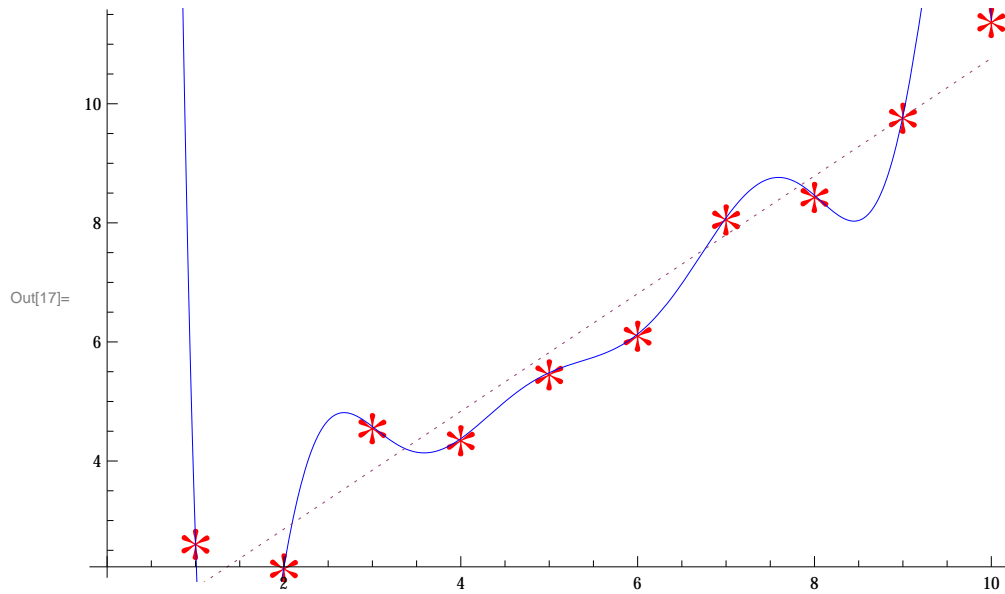
```

model4 = LinearModelFit[data, {x, x^2}, x]
model5 = LinearModelFit[data, {x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9}, x]
Show[ListPlot[data, PlotStyle -> Red, PlotMarkers -> {"*", 42}],
      Plot[{model5[x], model4[x]}, {x, 0, 10}, PlotStyle -> {Blue, Dotted}],
      ImageSize -> Scaled[0.75]]

```

Out[15]= FittedModel [$1.82804 + 0.514745x + 0.0430515x^2$]

Out[16]= FittedModel [$326.819 - 865.008x + \ll 9 \gg + 0.0204298x^8 - 0.000411603x^9$]



Overfitting can do more harm than good.

Multivariate linear regression

Unless we found an actual malware sample, all we can infer is a bunch of **scores**.

This hold true for all reputation-based systems.

This also holds true for antiviruses when using heuristics (different “severity levels”).

This also holds true for antiviruses when **not** using heuristics, because false positives and false negatives are present in any signature/whitelist/blacklist-based system.

We use multivariate linear regression to aggregate scores.

Classification

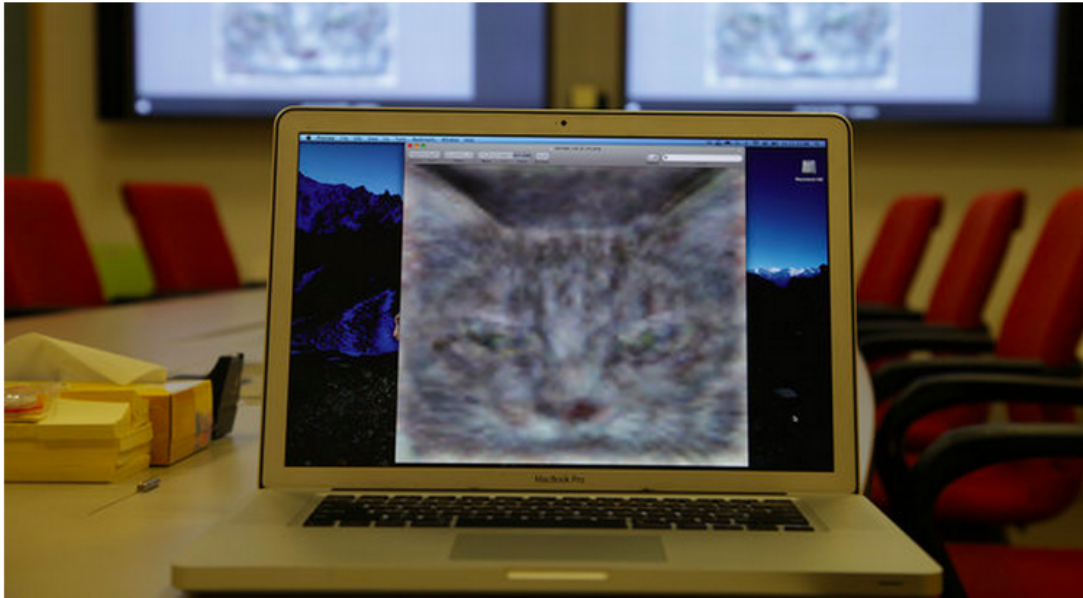
```
In[18]:= ts1 = TableForm[{{"PHP", "Cpanel", "Wordpress", "Popularity", "Reputation", "Label"},  
  {"T", "T", "T", 0.1, -50, "malicious"}, {"T", "T", "F", 75, 80, "benign"},  
  {"T", "F", "T", 2.3, -4, "malicious"}, {"F", "F", "F", 40, 2, "benign"},  
  {"...", "...", "...", "...", "...", "..."},  
  {"T", "F", "F", 20, -10, "NA"}}]
```

Out[18]/TableForm=

PHP	Cpanel	Wordpress	Popularity	Reputation	Label
T	T	T	0.1	-50	malicious
T	T	F	75	80	benign
T	F	T	2.3	-4	malicious
F	F	F	40	2	benign
...
T	F	F	20	-10	NA

What classifiers are used for

How Many Computers to Identify a Cat? 16,000



Jim Wilson/The New York Times

An image of a cat that a neural network taught itself to recognize.

By JOHN MARKOFF
Published: June 25, 2012

MOUNTAIN VIEW, Calif. — Inside [Google's](#) secretive X laboratory, known for inventing self-driving cars and augmented reality glasses, a small group of researchers began working several years ago on a simulation of the human brain.








Multimedia



Business Day Live | Google's

There Google scientists created one of the largest neural networks for machine learning by connecting 16,000 computer processors, which they turned loose on the Internet to learn on its own.

Presented with 10 million digital images found in YouTube videos, what did Google's brain do? What millions of humans do with YouTube: looked for cats.

-  FACEBOOK
-  TWITTER
-  GOOGLE+
-  E-MAIL
-  SHARE
-  PRINT
-  REPRINTS

Cat Head Detection - How to Effectively Exploit Shape and Texture Features

Weiwei Zhang¹, Jian Sun¹, and Xiaoou Tang²

¹ Microsoft Research Asia, Beijing, China
{weiweiz, jiansun}@microsoft.com

² Dept. of Information Engineering, The Chinese University of Hong Kong, Hong Kong
xtang@ie.cuhk.edu.hk

Abstract. In this paper, we focus on the problem of detecting the head of cat-like animals, adopting cat as a test case. We show that the performance depends crucially on how to effectively utilize the shape and texture features jointly. Specifically, we propose a two step approach for the cat head detection. In the first step, we train two individual detectors on two training sets. One training set is normalized to emphasize the shape features and the other is normalized to underscore the texture features. In the second step, we train a joint shape and texture fusion classifier to make the final decision. We demonstrate that a significant improvement can be obtained by our two step approach. In addition, we also propose a set of novel features based on oriented gradients, which outperforms existing leading features, e. g., Haar, HoG, and EoH. We evaluate our approach on a well labeled cat head data set with 10,000 images and PASCAL 2007 cat data.

⋮

Data we are storing, indexing and crunching

- 40 billion client queries / day. But only 4.3 billion valid distinct (client_ip, domain) pairs.
- Responses from authoritative servers: IP addresses, TTLs, mail servers and responses codes.
- Routing tables.
- Lists of URLs and domain names that have been flagged as malicious by 3rd party feeds, by individual security researchers, by ourselves (manually) and by our models. And, for phishing web sites, by our community (Phishtank).
- We keep historical data for all of these.
- Most of this data, plus the output of our models, is stored twice: in HDFS for our M/R jobs, and in HBase for ad-hoc lookups and classification. We're planning to store it in GraphLab as well.
- Some bits are stored in PostgreSQL, too. Because for JOINS and custom index types, it's dope.
- Pro-tip: bloom filters work really well for deduplicating data without having to sort it.

The Security Graph

The “Security Graph” exposes some of our data, including the output of some of our models, through a simple RESTful API.

Other security researchers can use our data to build their own models. Access is restricted, but free.

Exposed data includes:

- Lexical features computed on the fly.
- Network features: domain records + client diversity.
- Output from some models.

Client IP addresses are **never** exposed.

The Security Graph: demo!

(dear Wi-Fi, please do not crap out. Not now).

Fast flux?

Fast-fluxing is the art for a domain name to quickly switch to new IP addresses, and/or new name servers (double-fluxing).

These domains can quickly jump to another host before being detected. And firewalls can hardly block them.

The only way to take them down is to disable the domain name itself. Only registrars can do that. And they very rarely do.

We are using a classifier with a 21 features collected over a short time window to discover new fast-fluxy domains:

- TTL (mean, stddev) for A and NS records
- Number of IPs, prefixes and ASNs
- $\frac{\text{Number of IPs}}{\text{Number of ASNs}}$
- Number of countries
- Mean geographical distance for IP records and NS records
- Number of name servers
- ASN and domain age

We used a very strict training set, and as a result, only ≈ 10 domains are flagged every day, but with no known false positives so far.

Fast-flux are dying

Netcraft confirms™.

Fast-flux are mostly used for phishing, scams and by rogue pharmacies.

Cybercriminals are shifting to other techniques: disposable domains, dynamic DNS services, redirection services and compromised hosts.

Security features

Client geographic distribution

For a given domain, we use the Kolmogorov-Smirnov test to compare the geographic distribution of client queries with the predicted one for the TLD of this domain.

The output plays a very important role for detecting domain names that are ***not*** malicious.

Caveat: only applicable to domain names using a CCTLD.

ASN, prefix and IP reputation

Ex: ASN score

D_a : the set of domain names resolving to at least 1 IP announced by ASN a .

M : the set of domain names flagged as malicious.

c : mean number of malicious domains in $\{a \mid D_a \cap M \neq \emptyset\}$.

$$S(a) = \frac{|D_a \cap M|}{c + |D_a|}$$

Wait...



Hilary Mason
@hmason



Following

Speaking: 1 Kitten per Equation
bit.ly/16BJCrM

⋮



More cuteness at CuteStuff.co

Popularity

The Alexa rank only ranks web sites, not domain names. In particular, CDNs and ad networks are not included.

Mobile traffic? Nope.

And who is still using the Alexa toolbar? No, seriously?

The number of DNS queries is not a good popularity indicator, if only because it highly depends on the TTL.

Thus, we compute a “popularity score” based on the number of distinct IP addresses having visited a domain name. This is a bayesian average, similar to reputation scores.

In addition, we also run the PageRank algorithm on an undirected graph built from (client_ip, domain_name) pairs.

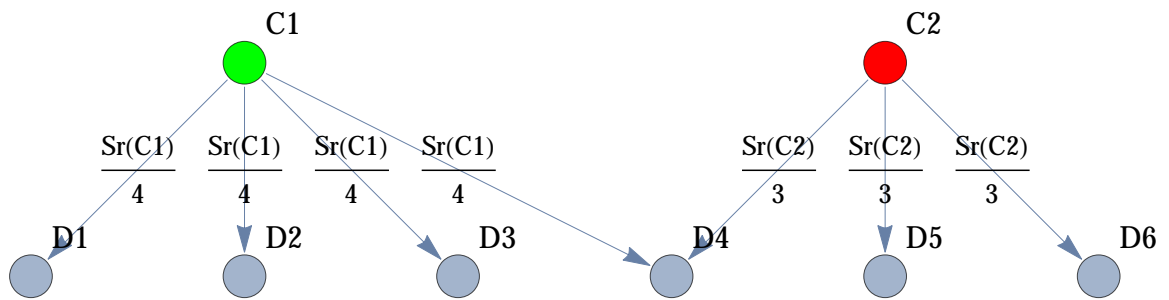
The SecureRank

We consider the (client_ip, domain_name) pairs as an undirected graph, and assign an initial positive rank Sr_0 to domain names from the Alexa top 200,000 list, and a negative score $-Sr_0$ to domain names from our blacklists.

Initialization

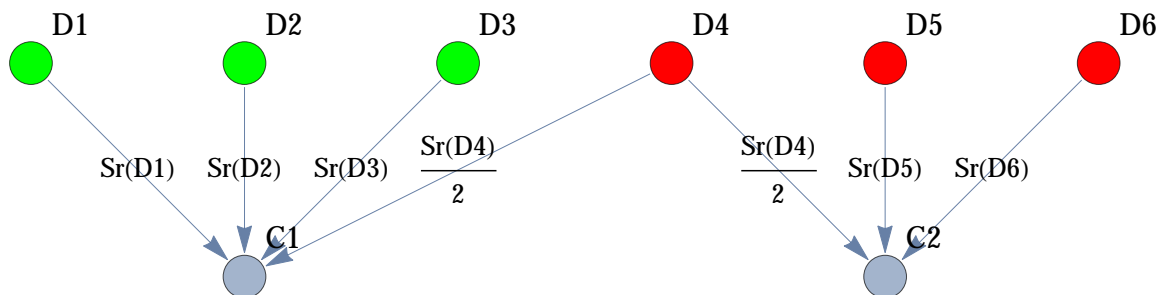
$$\begin{aligned} Sr(C1) &= Sr_0 \\ Sr(C2) &= -Sr_0 \end{aligned}$$

First iteration



$$\begin{aligned} Sr(D1) &:= \frac{Sr(C1)}{4} \\ Sr(D4) &:= \frac{Sr(C1)}{4} + \frac{Sr(C2)}{3} \\ Sr(D6) &:= \frac{Sr(C2)}{3} \end{aligned}$$

Next iteration



$$\text{Sr}(C1) := \text{Sr}(D1) + \text{Sr}(D2) + \text{Sr}(D3) + \frac{\text{Sr}(D4)}{2}$$

$$\text{Sr}(C2) := \frac{\text{Sr}(D4)}{2} + \text{Sr}(D5) + \text{Sr}(D6)$$

Caveats

- What prevents the SecureRank from converging towards the actual PageRank in practice:
 - We only perform a limited set of iterations, not actually iterating until convergence
 - The graph is not dense
 - Only immediate neighbors are visited at each iteration
- Low-degree vertices can lead to false positives
- The final SecureRank is highly dependant on the initial SecureRank
- A dangerous variable to use for classification

Still a very useful algorithm, see next slide...

Candidates selection

Every day, we build several lists of domain names having very low scores:

- SecureRank
- IP, Prefix and ASN reputation scores
- Lexical scores
- FrequencyRank
- Traffic spikes
- Traffic spikes for nonexistent domains
- C-Rank?

These lists have too many false positives to be blindly added to our list of blocked domains. However, they are used as inputs for other models.

Co-occurrences: demo

(Saint Wi-Fi, priez pour que le reseau ne foire pas pendant cette demo)

DNS is a mess

```
github.com [A]
a248.e.akamai.net [A]
1.courier-push-apple.com.akadns.net [A]
e3191.c.akamaiedge.net.0.1.cn.akamaiedge.net [A]
api.travis-ci.org [A]
codeclimate.com [A]
api.twitter.com [A]
s.twitter.com [AAAA]
gmail-imap.l.google.com [A]
i2.wp.com [A]
raw.github.com [A]
github.com [AAAA]
a248.e.akamai.net [AAAA]
api.travis-ci.org [AAAA]
i2.wp.com [AAAA]
raw.github.com [AAAA]
secure.gravatar.com [A]
travis-ci.org [A]
secure.gravatar.com [AAAA]
travis-ci.org [AAAA]
l.ghostery.com [A]
codeclimate.com [AAAA]
```

DNS logs are a mess

Our DNS resolver event loop:

- Accept / read a client query
- Retrieve ACLs, Active Directory permissions, and stuff
- Check if the response is cached
- Bummer, it's not. Ask authoritative servers, cope with errors, timeouts and stuff
- Store the response in a dedicated log file for the beloved research team
- Cache the response
- Check for blocked domains, categories, security flags and stuff
- Recurse if required: go to (current step - 5)
- Update useless stats, counters and debugging crap
- Send the response to the client
- Add a line to the log file, containing the current timestamp, and something that approximately describes what just happened.

See the problem?

Timestamps we are logging don't match timestamps of queries sent by a client, not even when the queries were received. To top it off, UDP packets can be received in a different order, too.

But temporal proximity is all we have in order to link domains to each other, anyways.

More often than once, more data doesn't improve a crappy model, it just makes it slow. But in this case, throwing a lot of data definitely helps transforming shit to gold.

Co-occurrences

Intuition: domains names frequently looked up around the same time are more likely to be related than domain names that aren't.

Intuition: the data really needs to be cleaned up if we want to get decent results.

Intuition: the previous intuitions don't look too bad. Let's do it.

Co-occurrences (2)

All we have to identify a device is a client IP, and the 1 client IP = 1 device assertion just doesn't work:

- Dynamic IP addresses / roaming users
- NAT

What we are doing to mitigate this:

- Instead of processing log files from a entire day, we process a 1 hour time slice over 5 days (the peak hour for each datacenter).
- When multiple (client_ip, domain_name) pairs are seen, we only keep the one with the oldest timestamp, so that our model can't be screwed up (intentionally or not) by a single client.
- Queries from clients IPs having sent more queries than 99.9% of other clients are discarded. This ignores heavy NAT users in order to improve our model.

Pro-Tip: LinkedIn's DataFu package is cool for computing approximate quantiles on a data stream.

Co-occurrences (3)

t_1, \dots, t_n : timestamps of queries sent by a given client.

We need to define the distance between 2 domains i and j looked up by this client.

How about: $d(i,j) = |t_i - t_j|$?

What does the distribution of $d(i,j)$ look like, on real examples of related domains?

$$g(i, j) = \frac{1}{\sqrt{1 + \alpha |t_i - t_j|}}$$

What if all the clients having sent queries to domains i and j had a say?

$$g(i, j) = \sum_{\{c | c \in C \wedge \{i, j\} \subset D_c\}} \frac{1}{\sqrt{1 + \alpha |t_i(c) - t_j(c)|}}$$

C : client IPs

D : domain names

D_c : domains names looked up by client c

$t_i(c)$: $\min(\text{timestamp})$ of a query for domain i by client c .

$g(i,j)$: co-occurrence score of domain j for domain i .

Co-occurrences (4)

This simple model performs very well.

Unfortunately, it's useless for discovering interesting domains.

Let's refine the function to address this:

$$s(i, j) = \frac{g(i, j)}{\sum_{k \in D} g(k, j)}$$

Normalization doesn't hurt:

$$s'(i, j) = \frac{s(i, j)}{\sum_{k \in D} s(i, k)}$$

Co-occurrences (done!)



C-Rank

Intuition: domain names frequently co-occurring with domain names already flagged as malicious are most likely to be malicious.

Let's use the co-occurrences scores for that.

M : set of domain names already flagged as malicious.

$$Cr(i) = - \frac{\sum_{j \in M} S'(j, i)}{\sum_{j \in D} S'(j, i)}$$



Domain classification

We extract 50 features (prior to vectorization) from our log records, from the output of our models, and some basic features computed on the fly:

- TLD, TLD length
- TTL {min, max, mean, median, stddev}
- Server countries and country count
- ASNs and ASN count
- Prefixes and prefix count
- Geographic locations, geographic locations count and mean distance
- IPs stability and diversity
- Lexical scores and other features based on the domain name
- KS test on geo client diversity
- Response codes
- CNAME record
- Popularity and PageRank
- C-Rank
- Tags, for domain names that have been manually reviewed
- Number of days since the first lookup for this domain has been seen
- ...

We use a Random Forest classifier (1000 trees, 12 random variables / tree, using a private fork of the RF-Ace project) to predict a label for unknown domains.

Predictions are done in real-time, in order to always use the most up-to-date dataset.

The amount of false positive is very low ($\approx 1\%$), but the classifier has to be retrained frequently.

On the importance of the training set

Our current training set contains $\approx 200,000$ records.

The initial training set we used had $\approx 500,000$ records but performed really poorly. We threw everything we had, including domains from the Conficker botnet, phishing and domain names that were flagged as malicious a long time ago.

Since we didn't had a lot of internal categories to sort our blacklists ("malware" was a catch-all list we dumped everything into), we used the Google Safe Browsing API in order to filter the list.

The size of the training set went down to $\approx 30,000$ records, but the performance of the classifier increased drastically, without any other changes.

Combining the output of different models

Every day, we build several lists of domain names having very low scores (yay, that was already said a couple slides back).

We remove domain names that we already flagged as malicious from these lists, and use these to build new lists, some of which are added to our blacklists.

- Prefix scores \cap Spikes
- Prefix scores \cap C-Rank
- IP score \cap SecureRank
- IP score \cap FrequencyRank
- IP score \cap NXDomain spikes
- ASN score \cap Google Safe Browsing \cap classifier label
- 3rd party lists \cap classifier label
- Experimental models \cap VirusTotal

A paradigm shift

Newly registered domains. Pseudorandom names. Short TTLs. A myriad of IPs spread over unrelated ASNs, most of them being already well known for hosting malicious content.

These are strong indicators, among others, that a domain is very likely to be malicious, and we have been using algorithms leveraging these to automatically spot these and protect our customers for a long time.

However, the security industry is currently observing a significant paradigm shift. Spammers, scammers and malware authors are now massively abusing compromised machines in order to operate their business.

Apache/Lighty/Nginx backdoors can stay under the radar for a long time.

The Kelihos trojan

Right after the Boston marathon bombing tragedy, a spam campaign drove recipients to a web page containing actual videos of the explosion.

With, as a bonus, a malicious iframe exploiting a recent vulnerability in Java in order to download and install the Kelihos trojan.

Here are some of the infectors serving the malicious Java bytecode:

```
kentuckyautoexchange.com  
infoland.xtrastudio.com  
aandjlandscapecreations.com  
detectorspecials.com  
incasi.xtrastudio.com  
sylaw.net  
franklincotn.us  
earart.com  
bigbendrивertours.com  
aeroimageworks.com  
winerackcellar.com
```

What all of these web sites have in common is that they were not malicious.

These were totally benign web sites, established for a long time, with a decent security track record and no distinctive network features.

They just got compromised, usually because of weak or stolen passwords.

Results

```
# Frank - Pandora botnet - 032813
axhost.info

# Frank - Athena botnet, see http://sk.tl/vQyY4J - 032813
truboot.org

# Frank - Blackhole, via Webroot - 032813
# Holly shit, no, we found and blocked that one 4 days ago
# and I didn't see any mentions of it on the interwebz before today.
# Yay! Champagne! <insert dancing banana here>

# Frank - RedKit - 032813
joffroy.de
my-biz.co.za

# Frank - BH - 032213
hohohomaza.ru
```

Results (2)

When a new threat has been found by other researchers, we frequently discover related domains before these are flagged by antiviruses and reputation systems. “before” varies between 1 minute and 1 week. Median is < 1 day.

Our models actually do a pretty good job at spotting false positives from 3rd party models.

But let's be fair, we didn't discover anything major (yet).

Or maybe we actually did block malicious content that nobody knows it was malicious. Not even us.

We are constantly seeing suspicious behaviors, suspicious domain names, suspicious client activity, suspicious network features. Now, is it an actual threat? Unless we find actual samples, we don't know.

This why we don't only rely on our own work. We just add our little piece to the puzzle.

An endless battle?

- We are mining DNS queries in order to find a needle in the haystack, namely domain names that are more likely to be malicious than others.
- Malware authors are smart, and are constantly finding new tricks to keep security researchers busy.
- There's no such thing as a one-size-fits-all model. We build different models to predict the maliciousness of a domain, combine these with 3rd party models, do a lot of manual investigation, and proactively block really suspicious stuff.
- This is slide #42.

