UNIVERSITY OF CAMBRIDGE

Measuring Android vulnerability, and UDP DDoS attacks

Daniel R. Thomas

Cambridge Cybercrime Centre, Department of Computer Science and Technology, University of Cambridge, UK

Luxembourg 2018-10

GPG: 5017 A1EC 0B29 08E3 CF64 7CCD 5514 35D5 D749 33D9
Firstname.Surname@cl.cam.ac.uk

Abstract
Measuring security allows us to understand whether security is improving, evaluate interventions, and provide evidence to regulators. Measuring security includes the measurement of the security of isolated systems or devices, and of entire ecosystems — including cybercrime within those systems.
This talk will focus on two pieces of security measurement work: Firstly measuring the relative vulnerability of Android devices from different providers [1]. Between 2011 and 2015 70% of Android devices were exposed to known critical vulnerabilities due to a lack of updates. There was wide variation between manufacturers who we found were the main bottleneck. We developed the FUM metric to measure the difference in security between different providers, the average score was 2.9 out of 10. Secondly I will describe our measurements of UDP reflection attacks over 1000 days using reflection honeypots and describe a technique for estimating the total number of UDP reflection attacks given only partial data [2].

Measuring security and cybercrime is important

- Is security getting better or worse?
- Did this intervention work?
- Is there a difference in security between these products?

Are we on a positive trajectory or do we need to start doing something differently

Testing whether interventions work is necessary for science but we need to be able to measure the improvement.

If we can compare products then we can pick more secure ones and that creates an economic incentive for manufacturers of those products to provide better ones. If regulators can tell the difference then they can regulate.

Two examples of security measurement research

- Measuring security of Android
- Measuring DDoS attacks (cybercrime)

Drawing out the principles, insights, and mistakes as we go along.

Security metrics for the Android ecosystem[1]

`https://androidvulnerabilities.org/`

Daniel R. Thomas

Andrew Rice

Alastair R. Beresford

Daniel Wagner

[1]Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. 2015. Security metrics for the Android ecosystem. In *ACM CCS workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, Denver, Colorado, USA, (Oct. 2015), 87–98. ISBN: 978-1-4503-3819-6.

This research is from 2015 and I am have mostly not updated figures or numbers, mostly because I don't have updated figures or numbers (more on that later).

Smartphones contain many apps written by a spectrum of developers

How "secure" is a smartphone?

Smartphones have lots of sensitive content on them and the quantity of sensitive data is still growing.
We don't trust developers
We have introduced a sandbox
Is the sandbox working?

Root/kernel exploits are harmful

- Root exploits break permission model
- Cannot recover to a safe state
- In 2012 37% Android malware used root exploits
- We're interested in critical vulnerabilities, exploitable by code running on the device

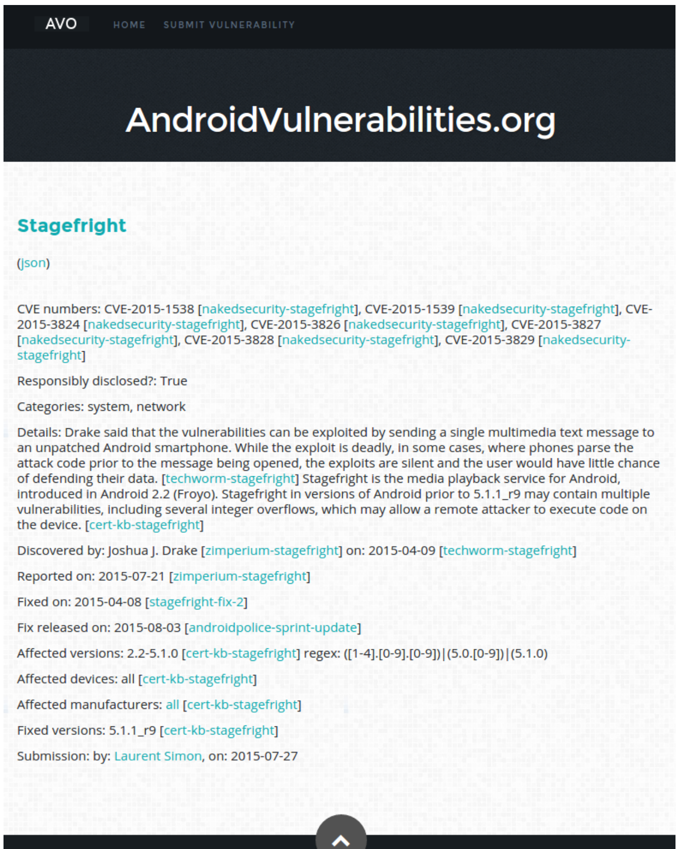
Is malware trying to break out of the sandbox?
We know that malware does not necessarily need to break out of the sandbox to cause problems, but that is not our focus here. Vulnerability is also rather more subtle than this critical/not critical distinction used here for simplicity. Composite vulnerability modelling is future work.

Hypothesis: devices vulnerable because they are not updated

- Anecdotal evidence was that updates rarely happen
- Android phones, sold on 1-2 year contracts

My anecdotes are now a bit out of date as I have not replaced my phone since writing this in 2015 and I also have not had any updates since 2015. While there is anecdotal evidence, there is a lack of concrete data about what is really happening.
Many devices actually used for longer than 2 years.
In contrast Windows XP could be purchased for a one off payment and got updates from 2001 until 2014.

No central database of Android vulnerabilities: so we built one



Stagefright

(json)

CVE numbers: CVE-2015-1538 [nakedsecurity-stagefright], CVE-2015-1539 [nakedsecurity-stagefright], CVE-2015-3824 [nakedsecurity-stagefright], CVE-2015-3826 [nakedsecurity-stagefright], CVE-2015-3827 [nakedsecurity-stagefright], CVE-2015-3828 [nakedsecurity-stagefright], CVE-2015-3829 [nakedsecurity-stagefright]

Responsibly disclosed?: True

Categories: system, network

Details: Drake said that the vulnerabilities can be exploited by sending a single multimedia text message to an unpatched Android smartphone. While the exploit is deadly, in some cases, where phones parse the attack code prior to the message being opened, the exploits are silent and the user would have little chance of defending their data. [techworm-stagefright] Stagefright is the media playback service for Android, introduced in Android 2.2 (Froyo). Stagefright in versions of Android prior to 5.1.1_r9 may contain multiple vulnerabilities, including several integer overflows, which may allow a remote attacker to execute code on the device. [cert-kb-stagefright]

Discovered by: Joshua J. Drake [zimperium-stagefright] on: 2015-04-09 [techworm-stagefright]

Reported on: 2015-07-21 [zimperium-stagefright]

Fixed on: 2015-04-08 [stagefright-fix-2]

Fix released on: 2015-08-03 [androidpolice-sprint-update]

Affected versions: 2.2-5.1.0 [cert-kb-stagefright] regex: ([1-4].[0-9].[0-9])|(5.0.[0-9])|(5.1.0)

Affected devices: all [cert-kb-stagefright]

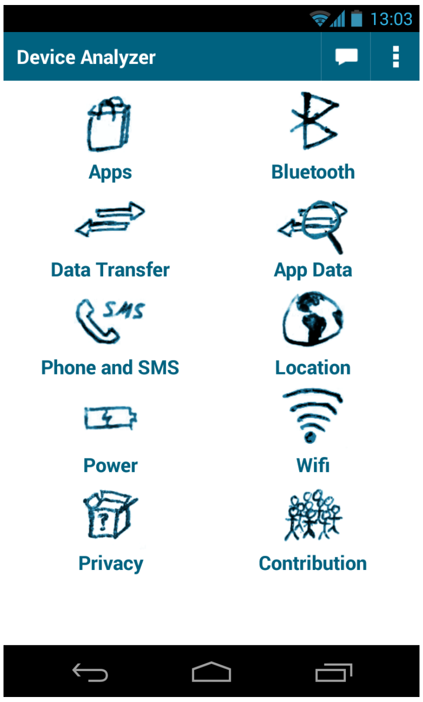
Affected manufacturers: all [cert-kb-stagefright]

Fixed versions: 5.1.1_r9 [cert-kb-stagefright]

Submission: by: Laurent Simon, on: 2015-07-27

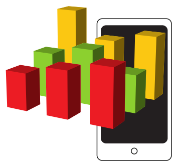
Collected a whole bunch of vulnerabilities, including a number of critical vulnerabilities that lack CVE numbers.
Standard trawling of forums, blog posts etc. as well as the CVE databases.
Not been updated since 2015 and so now very outdated. However, all ready to go if someone wants to start it up again. This seems to always happen with research projects, I was critical of others who did the same thing but then did it myself. There is little incentive to keep updating something like this if you don't have another paper coming out of it. Lots of tedious manual work to maintain.
I would perhaps also not use the same terminology "responsible disclosure" has gone out of fashion in favour of "coordinated disclosure" as callint it "responsible" is considered a pejorative towards people chosing different disclosure strategies.

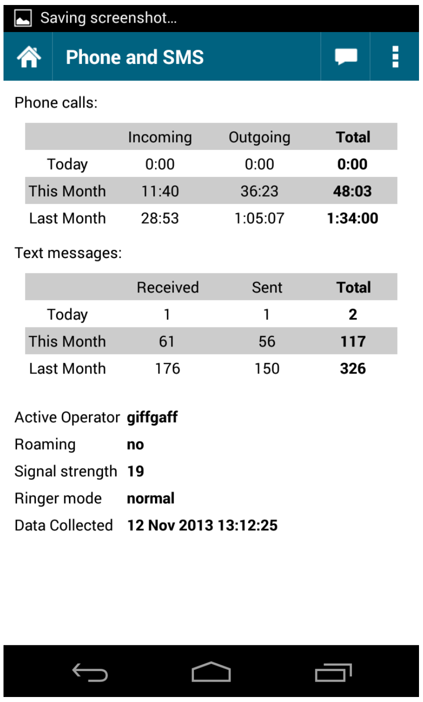
Device Analyzer gathers statistics on mobile phone usage



- Deployed May '11
- 30 000 contributors
- 4 000 phone years
- 180 billion records
- 10TB of data
- 1089 7-day active contributors

(2015 numbers)





Phone calls:

	Incoming	Outgoing	Total
Today	0:00	0:00	**0:00**
This Month	11:40	36:23	**48:03**
Last Month	28:53	1:05:07	**1:34:00**

Text messages:

	Received	Sent	Total
Today	1	1	**2**
This Month	61	56	**117**
Last Month	176	150	**326**

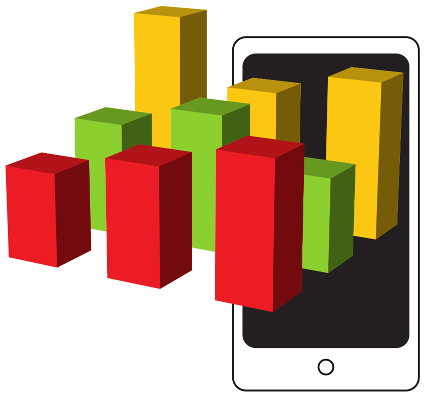
Active Operator	**giffgaff**
Roaming	**no**
Signal strength	**19**
Ringer mode	**normal**
Data Collected	**12 Nov 2013 13:12:25**

Device Analyzer has been running since 2011.
You can use the data for your own research and you can install the app to contribute to research.
Actually being actively developed at the moment (not true back in 2015).

Device Analyzer gathers wide variety of data

Including: system statistics

- OS version and build number
- Manufacturer and device model
- Network operators

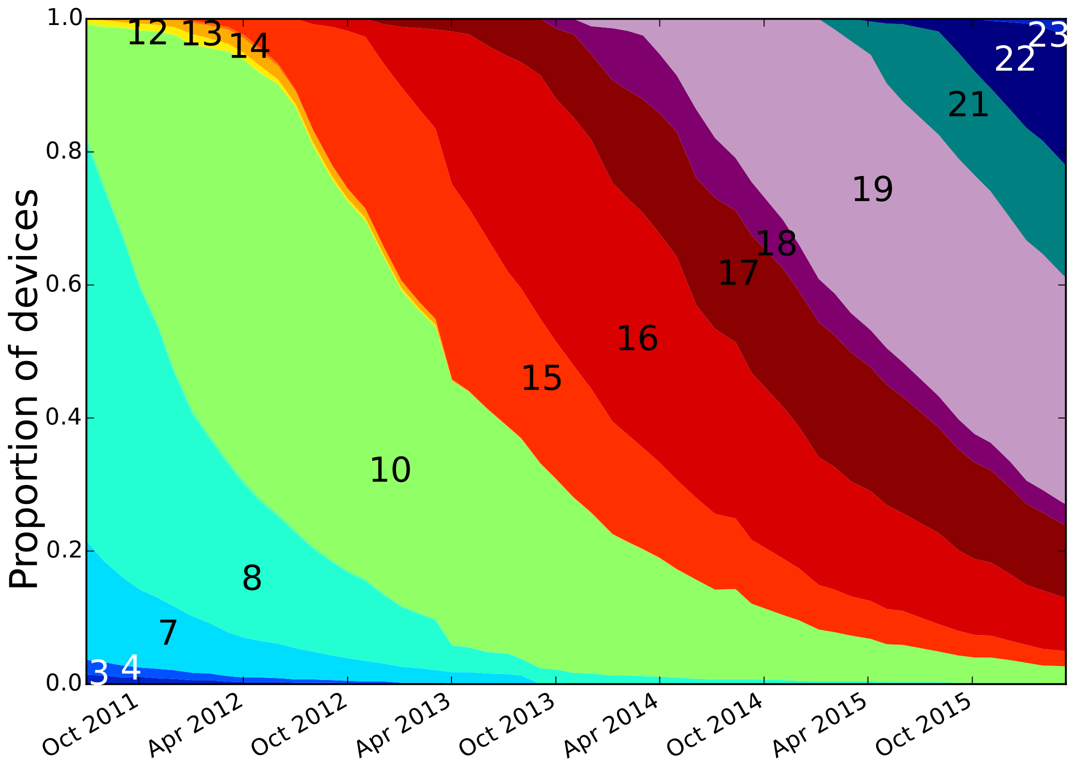


We use the OS version and build number information along with the manufacturer and device model information.
This can be combined with data on vulnerabilities to work out which devices were exposed to which vulnerabilities over time and apportion that to manufacturers, network operators and device models.

Is the *ecosystem* getting updated?

One thing we can look at is whether the ecosystem as a whole is being updated. If it is not being updated then it can't be secure.

Google data: device API levels



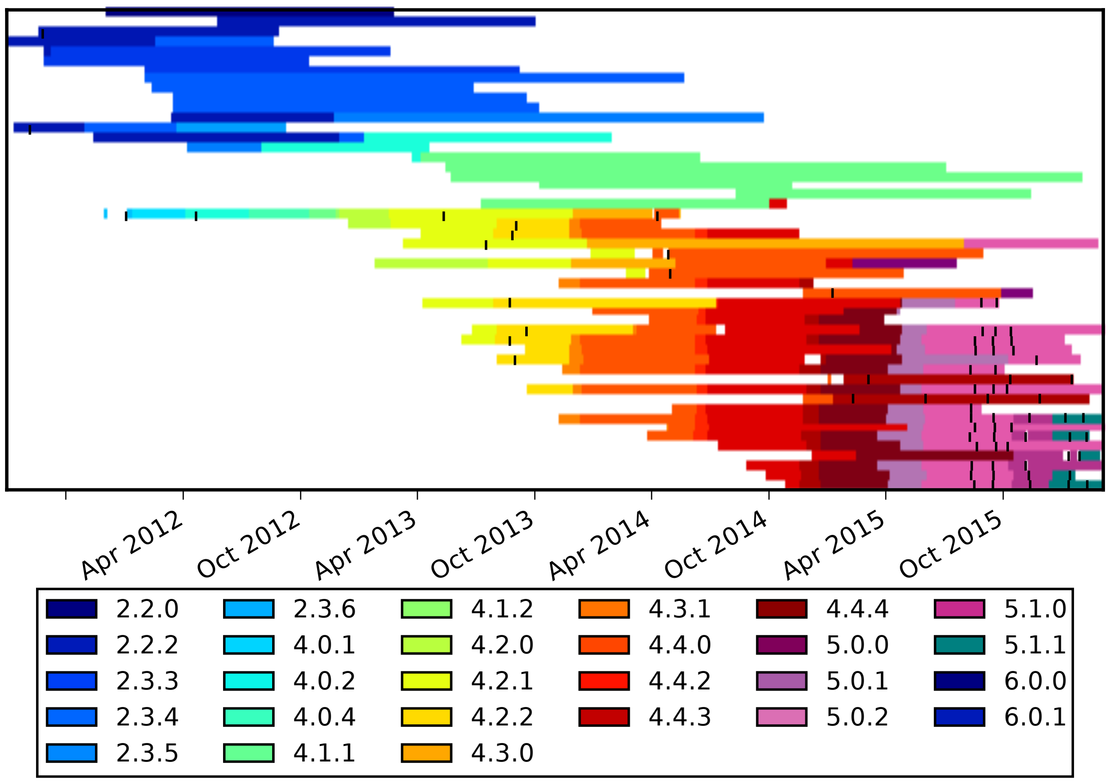
I collected (and still collect) Google Play's monthly data on API versions installed on devices contacting Google Play.
This shows that it takes a long time for updates to be deployed. This graph shows both updates due to devices getting updates, updates due to devices getting replaced, and updates due to new phones being sold to new people who didn't have phones before (reducing the proportion of old phone users).
Aside: longitudinal studies are important but hard so try to think if there is some data that you could start collecting now so that in 5 years time you can publish something really interesting.

Are *devices* getting updated?

However the change in the ecosystem could be due to old devices getting binned and new ones being bought.

To work out if devices are being updated we need longitudinal data on individual devices. This is provided by Device Analyzer.

LG devices by OS version



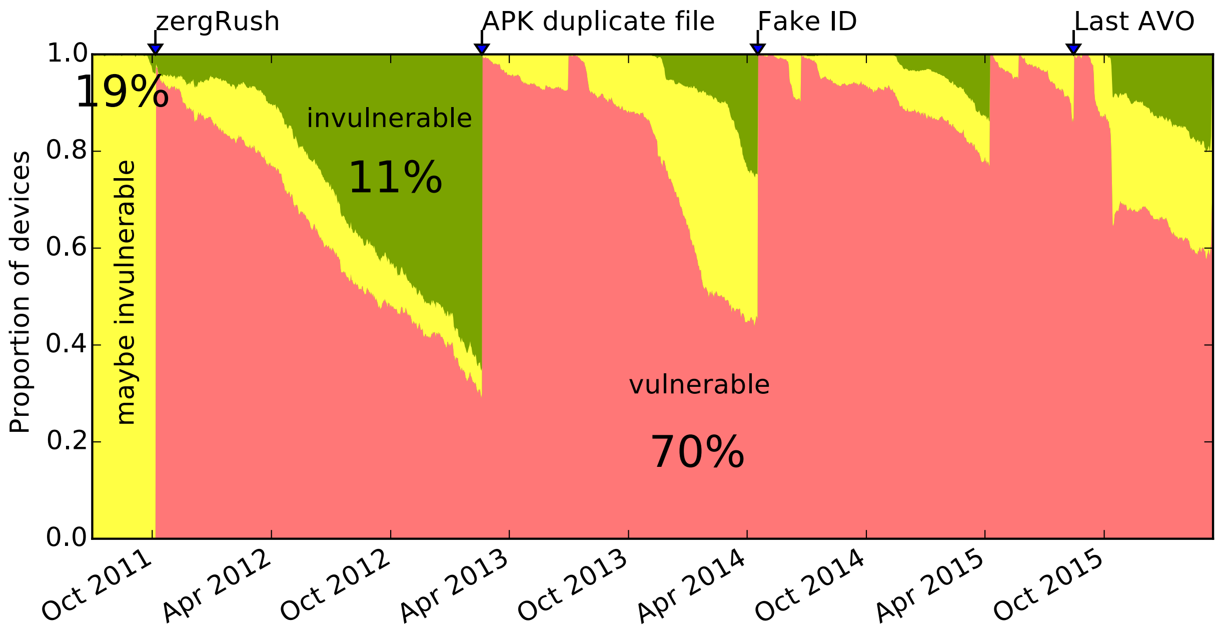
Top 50 LG devices (by length of contribution), many have received updates. But you can also see that many of the older devices didn't receive updates, there appears to have been a change in LG's behaviour.
Slightly strange looking hard to read but colourful plot, so many days of my life spent trying to make these work well in matplotlib.
The black marks indicate build number only updates where the version number did not change.

Connecting the two data sets: assume OS version $\rightarrow$ vulnerability

- We have an OS version from Device Analyzer
- We have vulnerability data with OS versions
- Match on OS and Build Number and assign:
  - Vulnerable
  - Maybe invulnerable
  - Invulnerable (not known vulnerable)

A device is insecure if it is exposed to known vulnerabilities because it's OS was built before the vulnerability was fixed and so must contain the vulnerability.
It is maybe secure if its build number was only observed after the vulnerability was fixed but the OS version number is known to be insecure.
It is secure if it is running a known good version of Android for that date.

Vulnerability varies over time



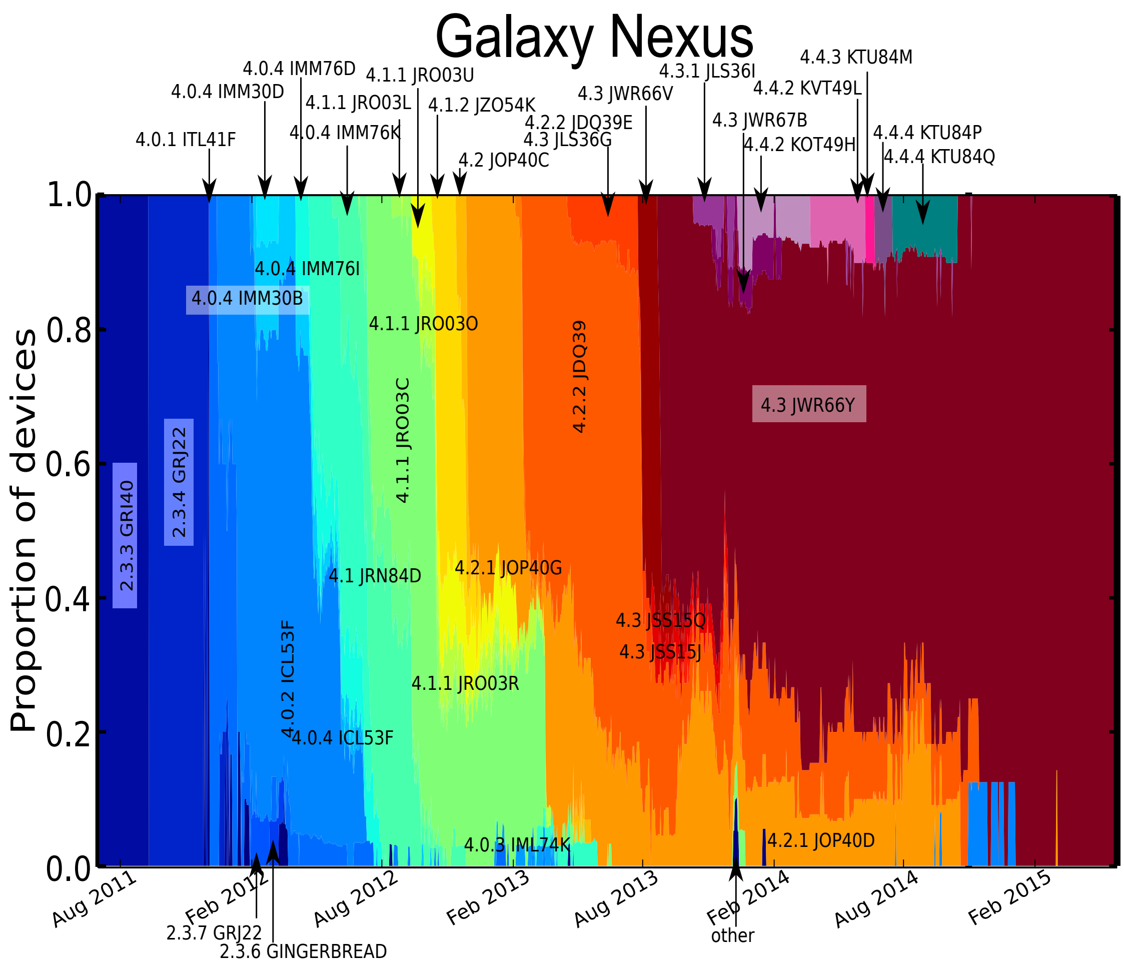
To start off with everything is maybe secure as we don't have data before a vulnerability was discovered to know if the build number was made after it, however once zergRush was discovered we knew how bad things were. 14 vulnerabilities contribute to this graph. Red vertical lines are caused by the discovery of vulnerabilities. After "Last AVO" the graph shows improvement, but this might just be an artefact of the lack of additional AVO data.

The FUM metric measures the security of Android devices

$$\mathrm{FUM} = 4f + 3u + 3\frac{2}{1 + e^{m}}$$

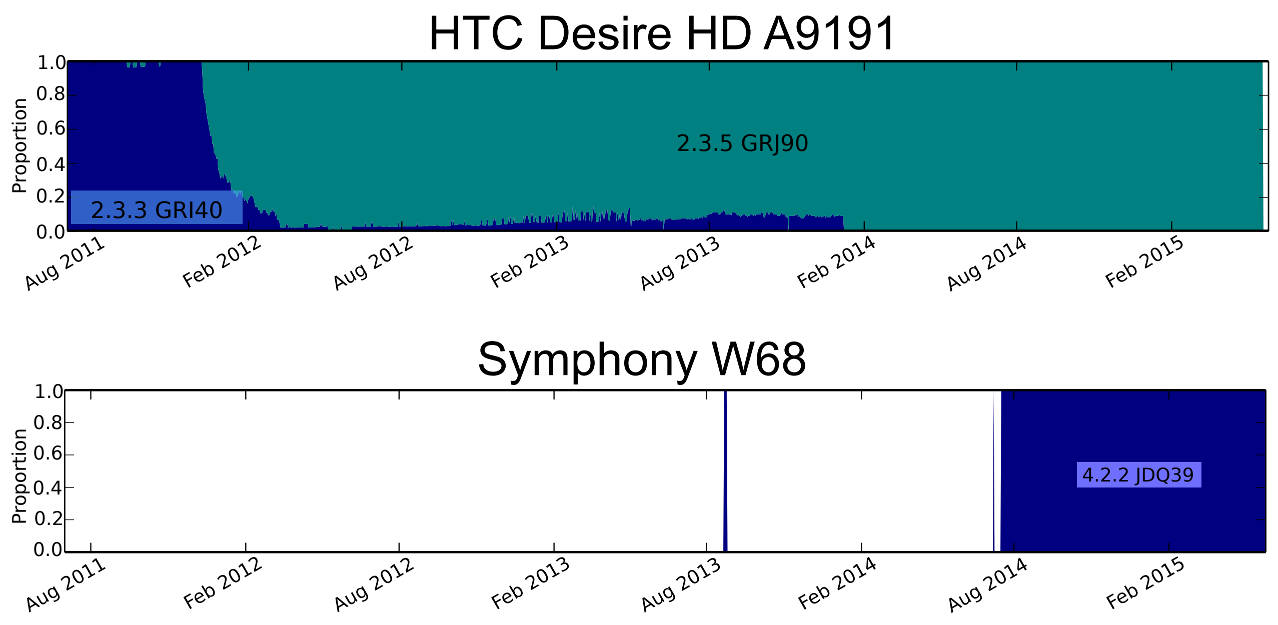
*f*ree from (known) vulnerabilities
*u*pdated to the latest version
*m*ean unfixed vulnerabilities

To provide a score out of 10 to compare manufacturers we combine three metrics that are variations on ones that have been used in the past.
Free = Proportion of devices free from vulnerabilities
Update = Proportion of devices running the latest version of Android used by that device manufacturer.
Mean = Mean number of vulnerabilities affecting not fixed on any device by the device manufacturer. This has to be scaled to between 0 and 1, hence the more complicated expression.
4+3+3=10
The sensitivity of this metric to changes is discussed in the paper.
We think it is hard to game this score without actually improving security.
Caveat: Proportion maybe invulnerable is just ignored in these calculations for historical reasons.



Is the score reasonable?
This is the highest scoring device model, it gets lots of updates. Sometimes only a few devices ever see a particular update and distribution of updates is not immediate, but it is pretty quick, especially in comparison with the ecosystem view.

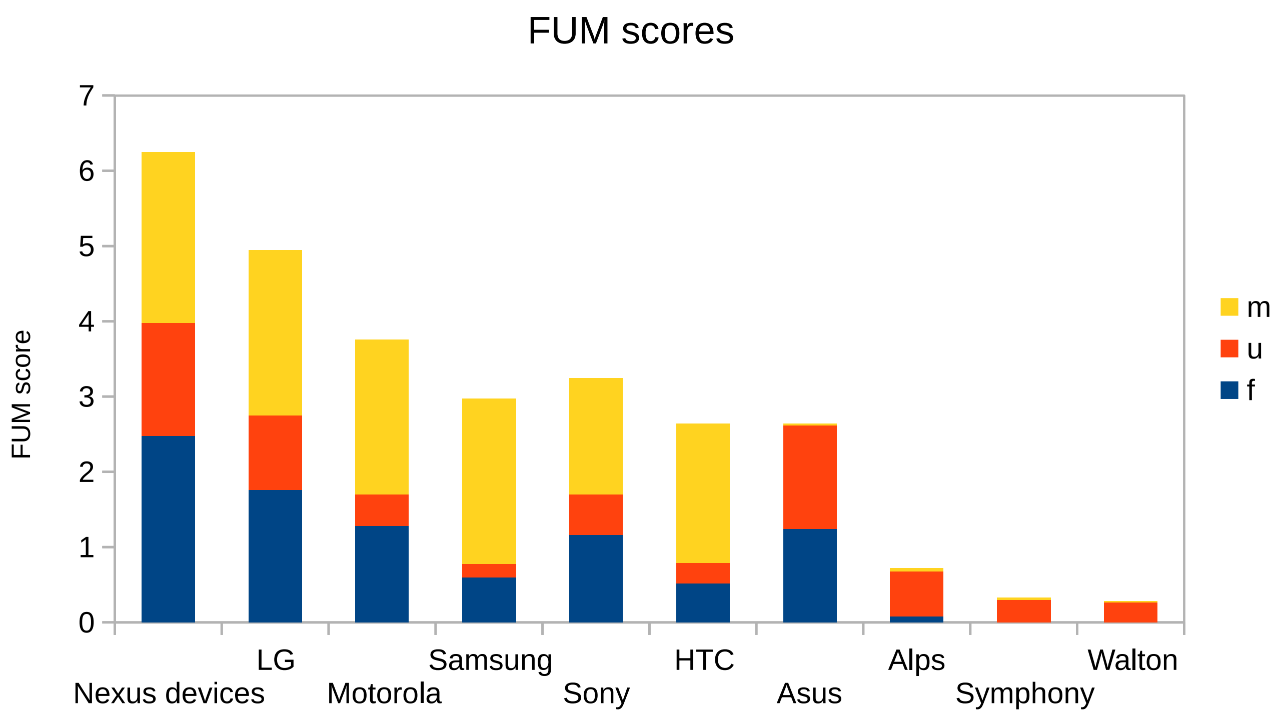
Lack of security updates



UNIVERSITY OF CAMBRIDGE

These are two of the lowest scoring device models, one of them got one update.
The first one starts on the same build number as our highest scoring device model. Other device models with similar names got rather better scores. How do you show uncertainty on plots like this? Some parts of these plots may be based on contributions from a statistically insignificant number of devices.

Comparing manufacturers



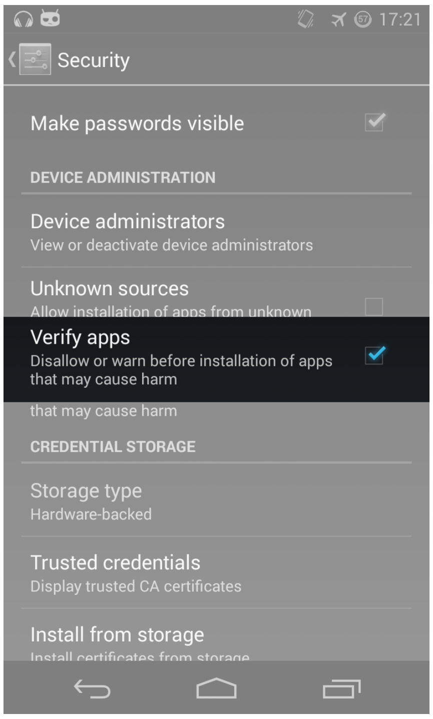
Nexus devices are not really a manufacturer, and actually LG (second on this list), was the main manufacturer of Nexus devices during the period of study. This means that the fact that LG does rather worse than Nexus devices implies that its non-Nexus devices are not looked after nearly so well.
There are companies on this list you probably haven't heard of because they were big in Bangladesh where we had a focussed study. There are also manufacturers that you may well have heard of, which are not on this list, because they were not big at the time.

Why is fixing vulnerabilities hard: software ecosystem is complex

- Division of labour
  - Open source software
  - Core OS production
  - Driver writer
  - Device manufacturer
  - Retailer
  - Customer
- Apple and Google have different models
  - Hypothesis: Apple's model is more secure

Security updates have to pass through a lot of different hands before they reach the device and any step could impose a delay.
Apple has a vertically integrated solution, perhaps simplifying things though they still have some external dependencies.

Google to the rescue


Security

Make passwords visible ✓

DEVICE ADMINISTRATION

Device administrators
View or deactivate device administrators

Unknown sources
Allow installation of apps from unknown

Verify apps ✓
Disallow or warn before installation of apps that may cause harm

that may cause harm

CREDENTIAL STORAGE

Storage type
Hardware-backed

Trusted credentials
Display trusted CA certificates

Install from storage
Install certificates from storage

- Play Store
- Verify apps
- Android Security Patch Level
- Later: Android Enterprise Recommended

We saw that Android devices were mostly vulnerable to know critical security vulnerabilities, but we didn't see widespread exploitation. Why? Well you first have to get the malicious app onto the device. Composition and scalability of vulnerabilities comes into play again here.
Security updates within 90 days for at least 3 years

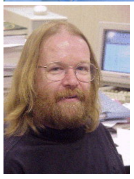
What happened next?

- Plenty press coverage
- Contacts with Google, manufacturers, UK Home Office
- FTC cites work.
- Google uses graphs to pressure manufacturers to improve update provision
- We move on: no further collection of vulnerability data, no updated scores.

Presenting metrics that produce comparative scores for the security provided by different entities such as manufacturers.
We collected data on what devices were doing and data that meant we could ascribe security properties to that data and then we could produce a score.

1000 days of UDP amplification DDoS attacks[2]

Daniel R. Thomas

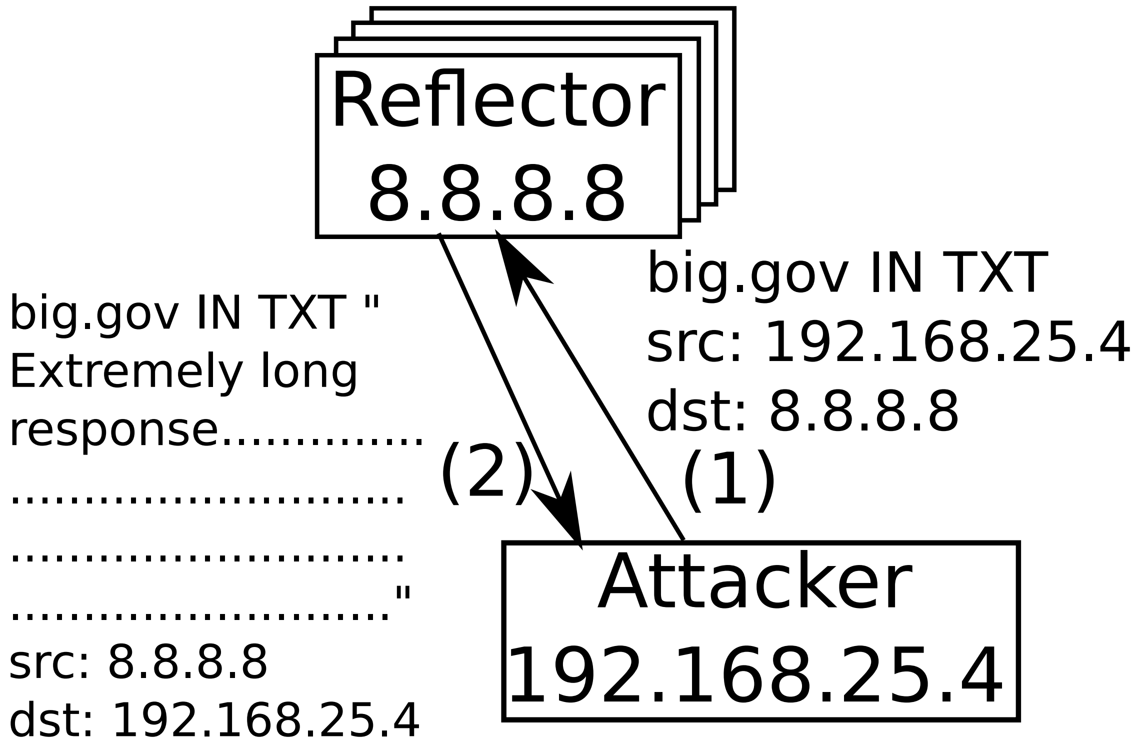

Richard Clayton

Alastair R. Beresford

[2]Daniel R. Thomas, Richard Clayton, and Alastair R. Beresford. 2017. 1000 days of UDP amplification DDoS attacks. In *APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, (Apr. 2017).

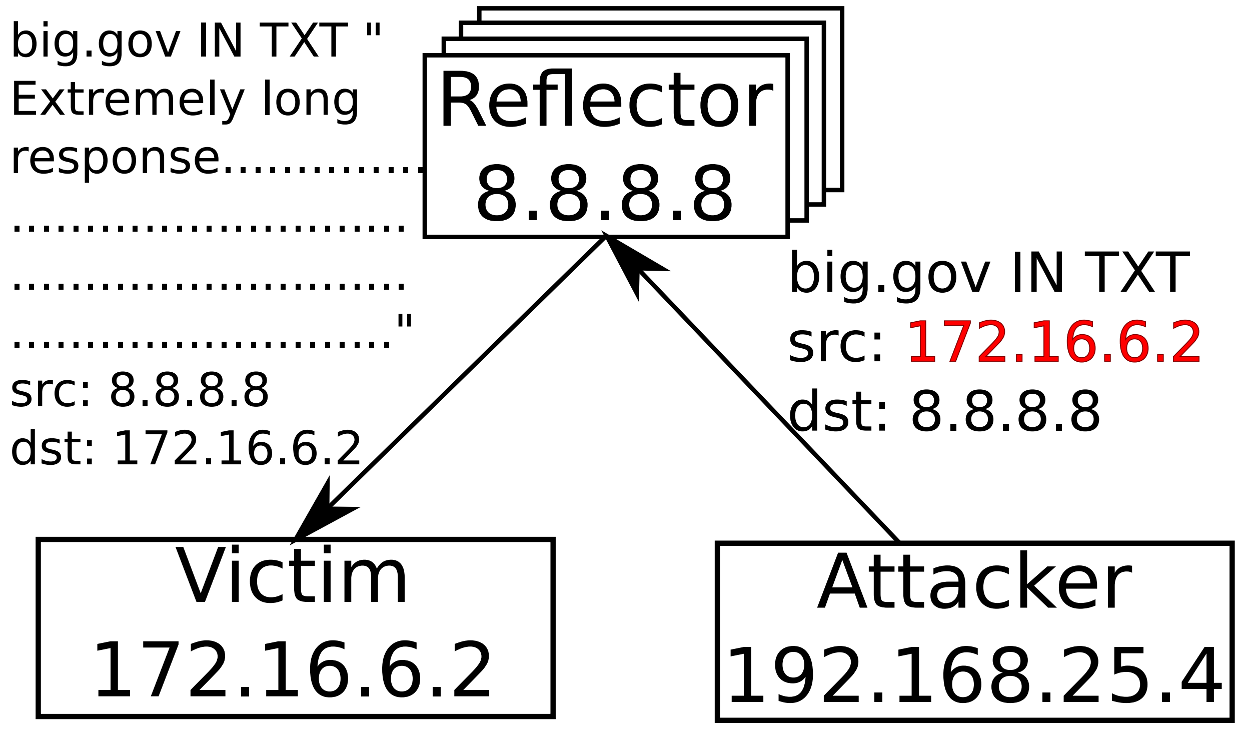
We have been using honeypots to collect data on UDP amplification Distributed Denial of Service attacks since March 2014. I will describe some of what we have learnt from this data and how we verified our results using leaked data.

UDP scanning


Reflector
8.8.8.8
big.gov IN TXT
src: 192.168.25.4
dst: 8.8.8.8
(1)
big.gov IN TXT " Extremely long response..."
src: 8.8.8.8
dst: 192.168.25.4
(2)
Attacker
192.168.25.4

To conduct UDP amplification DDoS attacks the attacker first needs to find reflectors it can use to reflect off.
To do this it uses UDP in a standard way, sending out UDP packets and collecting the responses.
In this example it sends out a DNS packet, and when it finds a real reflector it gets a response back.
In this way by scanning the IPv4 space attackers can build up a list of all the reflectors they can use for attacks. This can be done in 45 minutes on a fast connection. Some ISPs rate limit scanners and so you get better coverage with slower scans.
I am going to focus on attacks, but the paper has further discussion of scanners.

UDP reflection DDoS attacks



big.gov IN TXT " Extremely long response............ ...."
src: 8.8.8.8
dst: 172.16.6.2
Reflector
8.8.8.8
big.gov IN TXT
src: 172.16.6.2
dst: 8.8.8.8
Victim
172.16.6.2
Attacker
192.168.25.4

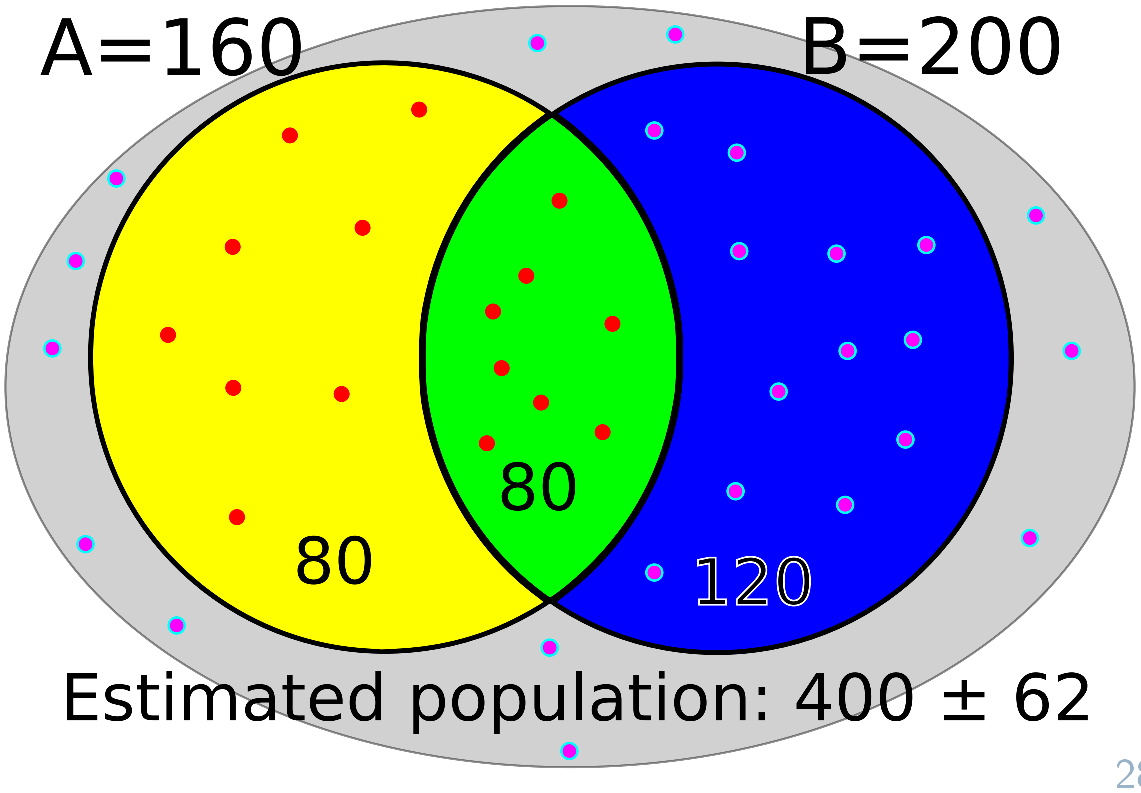
UDP reflection DDoS attacks exploit the fact that UDP (unlike TCP) does not verify the source IP address with a 3 way handshake. Hence, if an attacker can spoof the source IP address on the packets they send then the response will go to their victim.
In this example the attacker sends a DNS query to a resolver but spoofs the source IP address as the victim IP address. The much larger response goes to the victim.
The attacker can repeat this many times and over thousands of resolvers. This results in a large volume of traffic to the victim. The victim does not know the address of the attacker. Most of the attacks using this method are from booters: DDoS as a service.

We run lots of UDP honeypots

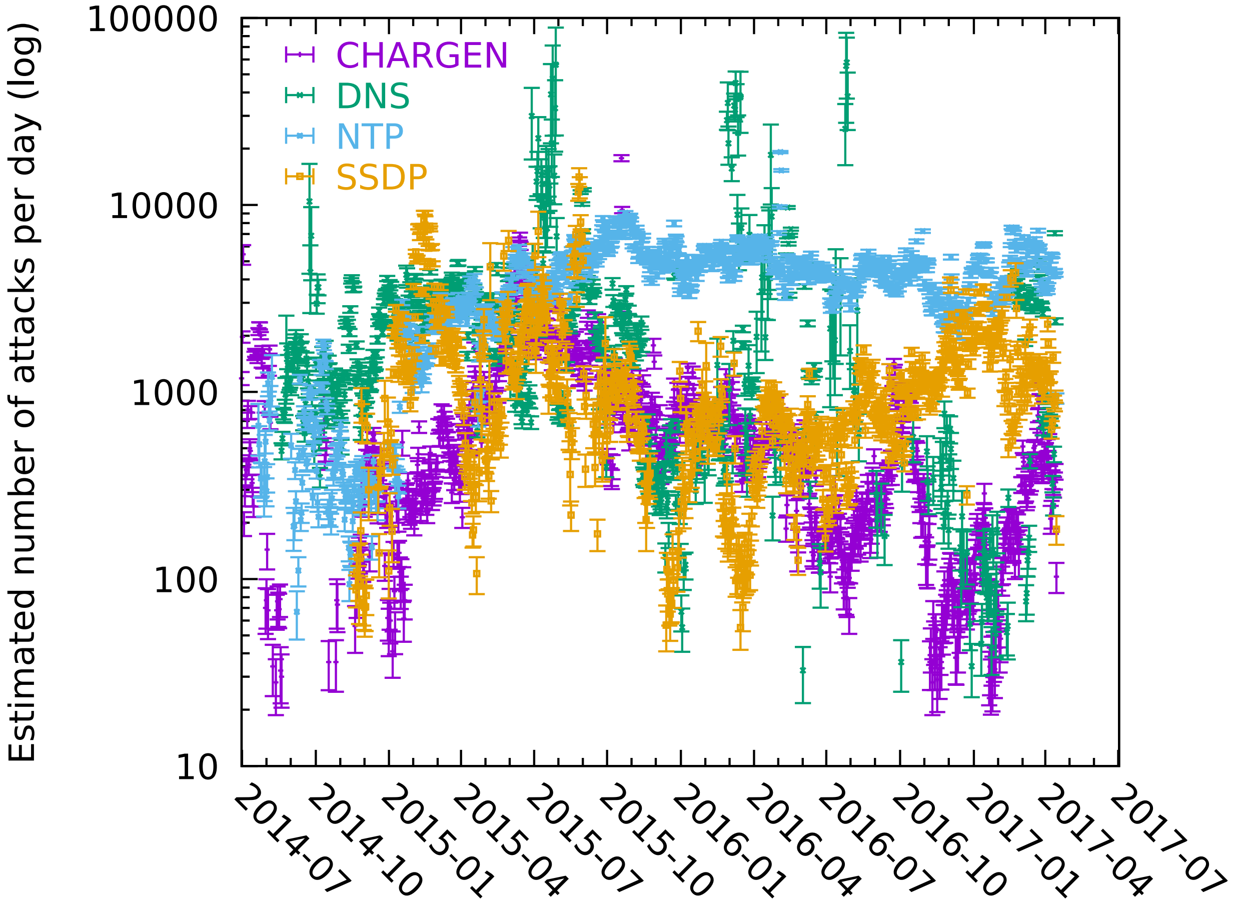
- Median 65 nodes since 2014
- Hopscotch emulates abused protocols
  QOTD, CHARGEN, DNS, NTP, SSDP, SQLMon, Portmap, mDNS, LDAP
- Sniffer records all resulting UDP traffic
- (try to) Only reply to black hat scanners

Since March 2014 we have been running UDP honeypots.
A small program called hopscotch emulates UDP protocols that are abused in UDP reflection attacks.
Another small program called sniffer records UDP traffic.
Hopscotch aims to only reply to black hat scanners and so when it has seen more than a handful of packets from the same destination it stops responding. The honeypots also collaborate to report victims and so not send them traffic.

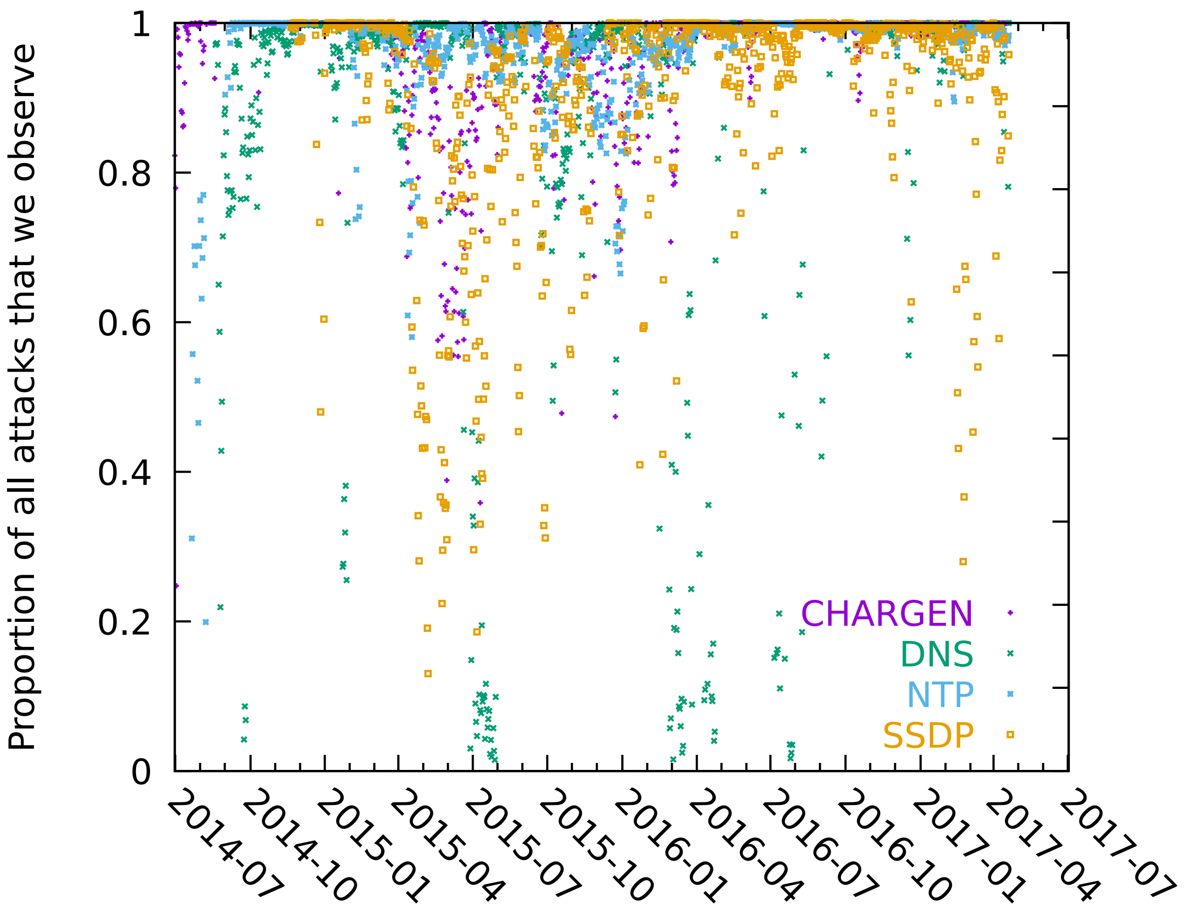
Total attacks estimated using capture-recapture



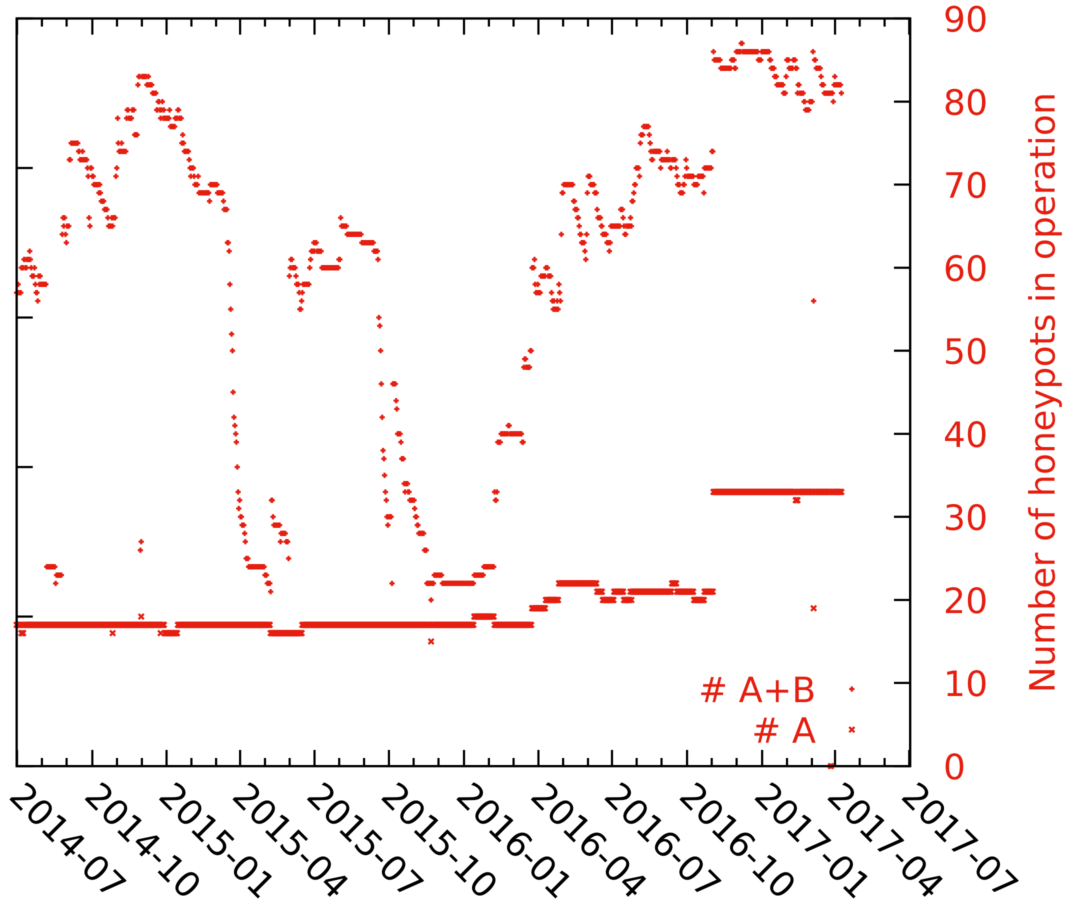
With these sensors we can see some attacks, but we want to know how many attacks there were, including the attacks we did not observe.
We can do this using the capture-recapture technique originally developed for ecology.
On day A we go fishing in a lake and catch 160 fish, mark them and return them to the lake, on day B we go fishing and catch 200 fish, of which 80 were marked as being previously caught. From this we can estimate that there are 400 fish in the lake.
We can then use this to estimate the total number of UDP attacks. We can split our sensors into two groups, A and B and look at the number of attacks that each detected and the size of the overlap.



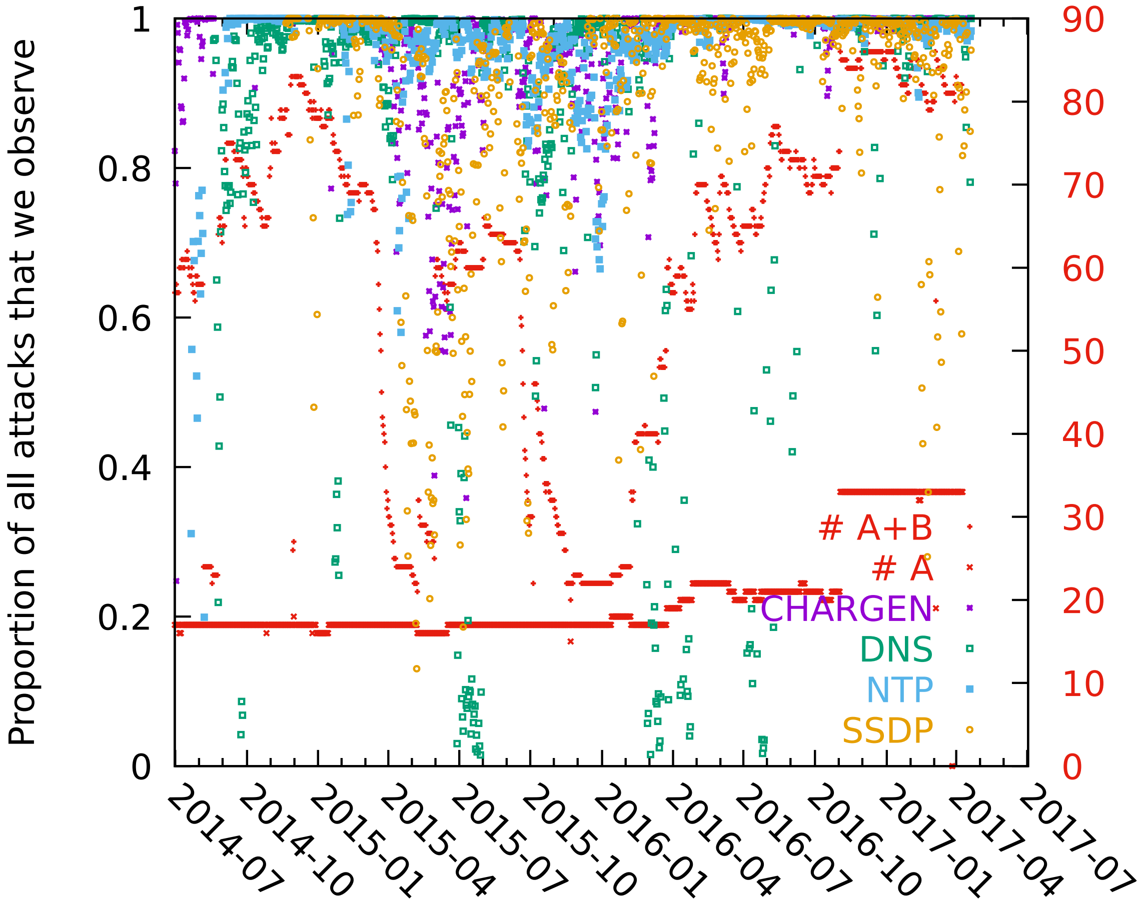
This graph shows the estimated total number of attacks per day for the four most used protocols.
It shows substantial changes in the number of attacks being made with each protocol over time.
SSDP is becoming more fashionable again after a period when it was much less widely used.
NTP has remained consistently popular and DNS has varied a lot, our data for DNS is not quite as good due to the large number of real DNS reflectors.
There was a paper that examined data from before the start of our measurement period and concluded that NTP was declining in popularity. Our longitudinal study shows that protocols go in and out of fashion. Just because it stops being used so much doesn't mean it won't come back.



This graph shows the proportion of the estimated total number of attacks that we observe each day. In general we have very good coverage, seeing almost all attacks. However, on some days we do rather worse, particularly for DNS and SSDP.



This graph shows both the total number of honeypots we had in operation and the number in the A set used for capture-recapture. It varies over time as a result of our main contributor ceasing to share data with us and our rebuilding our own network of sensors.



As you might expect there is correlation between the number of honeypots in operation and the proportion of attacks that we observe.

This was ethical

- We reduce harm by absorbing attack traffic
- We don't reply to white hat scanners (no timewasting)
- We used leaked data for validation, this was necessary and did not increase harm.
- Further discussion of the ethics of using leaked data for research tomorrow.

We followed our institutions ethics procedure. Running these honeypots reduces harm as when an attacker uses our honeypots to attack their victim their victim will receive rather less traffic than they would have if the attacker had used one of the many real reflectors.
To avoid wasting white hat's time we never reply to their scanners so they don't report us as being reflectors.

This is a solvable problem

- BCP38/SAVE
- Follow the money
- Enforce the law
- Warn customers it is illegal

CAIDA's spoofer prober project measures compliance with BCP38.
Paypal has made a big impact on booter revenue.
Lots of arrests have been made.
Booter users don't all realise fully that what they are doing is illegal.

Thank you! Questions?

Daniel R. Thomas
Daniel.Thomas@cl.cam.ac.uk
@DanielRThomas24
https://www.cl.cam.ac.uk/~drt24/
5017 A1EC 0B29 08E3 CF64 7CCD 5514 35D5 D749
33D9

Daniel Thomas is supported by the EPSRC [grant number EP/M020320/1].

References I

[1] Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. 2015. Security metrics for the Android ecosystem. In *ACM CCS workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, Denver, Colorado, USA, (Oct. 2015), 87–98. ISBN: 978-1-4503-3819-6.

[2] Daniel R. Thomas, Richard Clayton, and Alastair R. Beresford. 2017. 1000 days of UDP amplification DDoS attacks. In *APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, (Apr. 2017).