DroydSeuss: A Mobile Banking Trojan Tracker - Short Paper

Alberto Coletta[1], Victor van der Veen[2], and Federico Maggi[1]

[1] Politecnico di Milano, Italy
alberto.coletta@mail.polimi.it,federico.maggi@polimi.it

[2] VU University Amsterdam, The Netherlands
vvdveen@cs.vu.nl

Abstract After analyzing several Android mobile banking trojans, we observed the presence of repetitive artifacts that describe valuable information about the distribution of this class of malicious apps. Motivated by the high threat level posed by mobile banking trojans and by the lack of publicly available analysis and intelligence tools, we automated the extraction of such artifacts and created a malware tracker named DroydSeuss. DroydSeuss first processes applications both statically and dynamically, extracting relevant strings that contain traces of communication endpoints. Second, it prioritizes the extracted strings based on the APIs that manipulate them. Finally, DroydSeuss correlates the endpoints with descriptive metadata from the samples, providing aggregated statistics, raw data, and cross-sample information that allow researchers to pinpoint relevant groups of applications.
We connected DroydSeuss to the VirusTotal daily feed, consuming Android samples that perform banking-trojan activity. We manually analyzed its output and found supporting evidence to confirm its correctness. Remarkably, the most frequent itemset unveiled a campaign currently spreading against Chinese and Korean bank customers.
Although motivated by mobile banking trojans, DroydSeuss can be used to analyze the communication behavior of any suspicious application.

1 Introduction

With the widespread use of mobile devices as a second factor of authentication, malware authors equipped their *banking trojans* (e.g., ZeuS, SpyEye, Carberp, and derivatives), to leverage dedicated companion mobile malware trojan apps. With apps known in the underground as ZitMo, SpitMo, and CitMo, attackers create a man-in-the-middle between the banking website and the victim, effectively bypassing two factor authentication.

Cyber criminals strive to streamline the generation and distribution of mobile bankers as much as possible, using so-called *crimeware kits*, in order to reach large pools of victims. However, the more a cyber gang needs to automate their operations, the more likely their "dev ops" will leave traces that allow to identify groups of related apps. To our knowledge, however, most of the analysts' work is still done manually. Moreover, there is no mobile equivalent of the ZeuS Tracker [1], which turns out to be extremely useful to security

and malware researchers. Motivated by this, we created the DroydSeuss mobile malware tracker.

Our work is inspired by observations made during manual reverse engineering efforts. First, traces of C&C endpoints (e.g., phone number, domain name, URL) are typically visible in (byte)code, at runtime, or both, depending on the sophistication of the sample. Generally, finding both static and dynamic evidence of the same endpoint is a good indicator that the sample supports some configuration mechanism, which may reveal that it was generated semi-automatically (e.g., by a crimeware kit). Second, certain static features of the malicious app (e.g., prefixes of package name) that recur frequently, *together* with the same C&C endpoints is a further indicator that the malicious sample may be part of a campaign.

DroydSeuss runs each malware sample in an instrumented environment to track statically and dynamically allocated strings that are likely to be C&C endpoints. DroydSeuss has prioritization heuristics that assign an increasing level of importance to the candidate endpoints. In addition, DroydSeuss analyzes the itemsets containing (1) the extracted endpoints and (2) descriptive metadata from the samples (e.g., prefix of the package name). The outcome is a rank of itemsets that provide succinct insights such as *package name X is almost always related to phone-based C&C endpoints in country Y.*

We manually inspected the top 10 most frequent itemsets of each endpoint type (i.e., web and phone-based) and in all cases except one, we found contextual evidence from online resources that backed our findings. Interestingly, the left-out case led us to a very active, mobile-only banking trojan campaign targeting Korean and Chinese customers, of which no public evidence was available so far.

In summary, our work makes the following contributions:

- We propose a simple but effective pattern-elicitation technique based on frequent itemset mining which unveils growing campaigns and allows researchers to pin-point relevant groups of samples.
- We design and implement the *first* mobile malware tracker that leverages frequent itemset mining, along with classic static- and dynamic-data extraction techniques, to track phone- and web-based C&C endpoints.
- We release our tracker to the public at `http://droydseuss.com` where it has been running since October 2014; during this time frame, it correctly brought to our attention groups of trojans that would otherwise have required manual analysis in order to understand their importance.

2 DroydSeuss' Approach

The rationale behind DroydSeuss is twofold. First, evidence of interesting C&C endpoints can appear statically, dynamically, or both, depending on the sophistication of the sample. Second, metadata that recur frequently together with the same C&C endpoint indicates that there may be a crimeware kit or a campaign involved.

The remainder of this section describes how we implemented the aforementioned rationale in order to elicit relevant information that helps the analyst

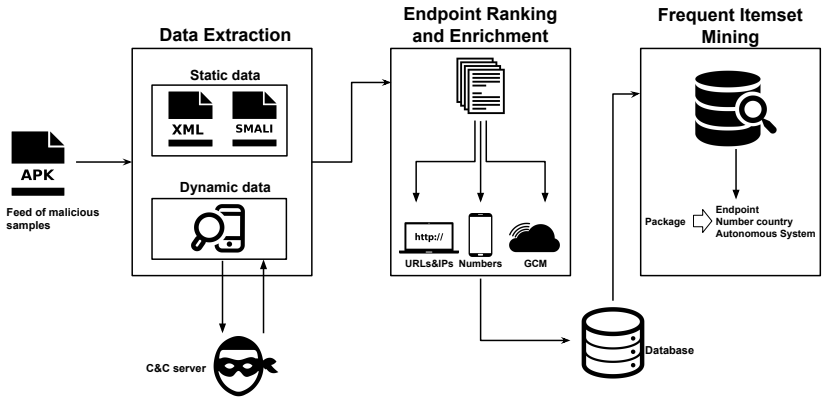

Figure 1: Data processing pipeline of DroydSeuss.

finding interesting groups of samples and, possibly, campaigns, starting from a feed of malicious apps. Figure 1 summarizes how **Phase 1** and **Phase 2** implement the first rationale, whereas **Phase 3** implements the second rationale.

2.1 Phase 1: Data Extraction

We are interested in capturing two categories of C&C endpoints: *web based* ones (e.g., IPs, URLs) and *phone based* ones. In addition, we also track recent trojans that take advantage of the Google Cloud Messaging (GCM) system [8] for push communication. While regular expressions for domain names, URLs and IPs are rather easy to write, parsing phone numbers is not straightforward. For this, we used the Python port of the `libphonenumber` library.

Static Data. As a preliminary step, we use `apktool` to unpack the APK archives, disassemble the Dalvik bytecode into an ASCII representation of the Smali assembly code and extract the manifest and resource files. Starting from static resources and Smali assembly files, this sub-phase is implemented as a set of regular expressions and post-processing scripts.

The following snippet, obtained from a real-world malicious APK, exemplifies how phone numbers and URL paths are saved in resource files.

XML resource file found in an iBanking sample.

```
<resources>
  <string name="def_tel_number">+43676800XXXX</string>
  <string name="urlPostData">/iBanking/sms/index.php</string>
  <string name="urlPostSms">/iBanking/sms/saveSMS.php</string>
  <string name="urlCommand">/iBanking/sms/sync.php</string>
  <string name="urlSmsList">/iBanking/getList.php</string>
  <string name="urlSendFile">/iBanking/sendFile.php</string>
  <string name="urlPing">/iBanking/sms/ping.php</string>
</resources>
```

Additionally, we parse Smali string constants denoted by the `const-string` instruction and check whether the manifest declares the GCM permission. An example of such string constant is shown in the following snippet.

C&C phone number declared in a malicious APK.

```
const-string v4, "+43676800XXXX"
```

Dynamic Data. In this sub-phase, we run the APK file in an instrumented sandbox. For our proof-of-concept implementation of DroydSeuss we leverage TraceDroid [20], which produces detailed, readable and easily parsable traces with deserialized arguments and return values.

We are particularly interested in stimulating the typical behaviours involved in botnet communication. To this end, TraceDroid triggers a number of special events (e.g., device reboot, phone call, network disconnect, etcetera). Furthermore, the sandbox launches all the activities and starts all the services declared in the app manifest. Finally, it runs the Android UI Exerciser Monkey that generates pseudo-random streams of user events such as clicks, touches and gestures.

The following listing shows an excerpt output resulting from the analysis of a real banker. The sub-phase continues by applying a set of regular expressions on the (string-typed) input arguments and return values of every API invocation to capture potential endpoints.

TraceDroid output file

```
public java.lang.StringBuilder
     java.lang.StringBuilder("http://dubleautoriza.net").append((java.lang.String)
     "/iBanking/sms/ping.php")
return (java.lang.StringBuilder) "http://dubleautoriza.net/iBanking/sms/ping.php"
public java.lang.String
     java.lang.StringBuilder("http://dubleautoriza.net/iBanking/sms/ping.php").toString()
return (java.lang.String) "http://dubleautoriza.net/iBanking/sms/ping.php"
new org.apache.http.client.methods.HttpPost((java.lang.String)
     "http://dubleautoriza.net/iBanking/sms/ping.php")
```

The following APIs are processed in a special way because they are directly connected with endpoint activity (see Section 2.2):

- `SmsManager`'s functions are interesting because they can be used to send text messages to a specified number.
- `URL.openConnection` is interesting because it specifies an URL to connect to. Also `apache.*` methods are tracked for the same purpose.
- `GoogleCloudMessaging.register` (and deprecated versions) are used for GCM-related operations. We are interested in extracting the sender ID, which uniquely identifies the server-side message sender.

2.2 Phase 2: Endpoint Ranking and Enrichment

In this phase, DroydSeuss ranks endpoints according to various heuristics. We define three rank levels, in order from the least to the most important:

1. **Suspicious**, if an endpoint is matched only during static analysis.
2. **Significant**, if a web endpoint is matched during dynamic analysis, yet in ancillary functions (e.g., string manipulation). This specific rank level allows to reveal cases such as the concatenation of a domain name with the paths.
3. **Important**, if the endpoint is matched during dynamic analysis in an API function which indicates that the malware has actually used the endpoint.

Additionally, we enrich extracted endpoints with the following details:

- **Geolocalization.** We use a free service [3] for IPs and the `pycountry` library for phone numbers.
- **Autonomous System.** We associate the IPs to their respective autonomous system using the Cymru service [4], which exposes a DNS-based API.

– **Phone number type.** Using `libphonenumber`, we determine whether the phone number is a fixed line, mobile, fixed line or mobile, toll free, premium rate, shared cost, VOIP, personal number, pager, UAN or voicemail.

2.3 Phase 3: Frequent Itemset Mining

In this phase, DroydSeuss implements our second rationale to elicit recurrent relations between the endpoints ranked as C&C (highest rank level) and the APK metadata. In principle, any metadata can be used but, based on the results of previous work [13] and on our results, we concentrate on the package name.

We leverage a fast, simple but effective frequent itemset mining technique. In essence, we count the occurrences of ⟨endpoint feature, package name prefix⟩ tuples, where *endpoint feature* ∈ {country, ASN, domain, IP} and *package name prefix* is the shortest prefix with at least two elements (e.g., `com.ann88.*`) (from hereinafter, for brevity, we use the terms *package name* and *package name prefix* as synonyms). Finally, we count occurrences of each itemset in our dataset of APKs, that is $\#\langle e, p\rangle \; \forall e, p$, where e and p are the actual values of the endpoint feature and package name. To obtain the *frequency* [11] ϕ of each itemset we calculate:

$$\phi\langle e, p\rangle = 2 \cdot \frac{\#\langle e, p\rangle}{\#(p) + \max_p \#(p)}$$

where $\#(p)$ is the number of APKs with that package name—we do not count multiple occurrences of a package within the same APK. We normalize $\phi \in [0, 1]$ by multiplying by a factor 2. Moreover, summing the count of the most frequent package name $\max_p \phi(p)$ gives more weight to really frequent packages and, on the other hand, avoids assigning high frequencies to itemsets generated by only few packages.

3 Experimental Results

Although providing a complete evaluation for an intelligence system that can potentially produce novel knowledge is a difficult task, we want to obtain quantitative and qualitative indicators about its performance, correctness and usefulness for the analyst.

3.1 Dataset and Setup

For the purpose of this evaluation we used 4,293 samples of banking trojans, downloaded nightly through the VirusTotal Intelligence API, that match the following malware families: ZitMo, SpitMo, CitMo, iBanking and FakeBank.

As Figure 2 shows, DroydSeuss has been running for almost 12 months since its release in late October 2014. Since January 10, we started monitoring web-based endpoints by sending an HTTP HEAD request and keeping track of whether they responded. On average, a little over 10% of the contacted endpoints responded.

3.2 Experiment 1: False Positives

As the feed of samples includes only known malicious APKs, we expect a low fraction of benign domains. This is important because the assumption behind

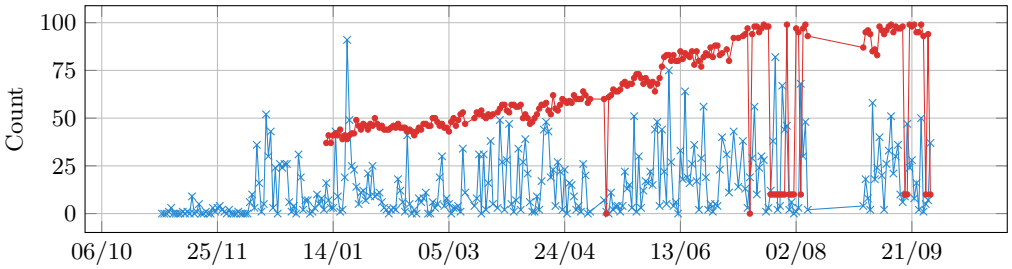

Figure 2: Daily number of downloaded/processed samples (blue) and number of responding web-based endpoints (red).

any malware tracker is that the activity performed by the sample is malicious or otherwise interesting for the malware analyst.

As of October 9, 2015, the fraction of benign domains in the feed of samples tracked by DroydSeuss is minuscule, summarized in Table 1. We obtained these numbers by using the Alexa Top 1M domain list [2] as a whitelist.

We manually analyzed the benign domains and found that the ones classified as *important* were `baidu.com` and `cl.ly`. The sample contacting `baidu.com` was using the following URL templates:

```
http://www.baidu.com/index.php?m=Api&a=SMSReceiver&imsi=CENSORED&number=
CENSORED&from=86279&content=CENSORED
```

Manual inspection of the sample confirmed that it was a data-stealing trojan that sent stolen data encoded as `baidu.com` URLs. In a similar vein, `cl.ly` URLs point to a legitimate file-hosting service, which is being used by the miscreants to host malicious APKs. This may appear strange, but using existing, popular services as the backend of a (mobile) botnet essentially creates a covert channel, making it hard for the analysts to separate between legitimate and non-legitimate traffic. In fact, although not prevalent, botnets that follow this approach have recently appeared in the wild [7]. In conclusion, although whitelisting the known benign domains is tempting, we believe that they should be reported to the analyst, possibly marked in some way.

3.3 Experiment 2: Reality Check

Quantitatively evaluating the correctness of DroydSeuss with respect to known benign endpoints is feasible, as discussed in the previous section. However, a quantitative evaluation of the recall is an ill-defined question, simply because a complete ground truth does not exist. If we had this ground truth, then there would be no reason for DroydSeuss to exist.

We opted for a qualitative evaluation that we carried out by manually Googling for endpoints which DroydSeuss assigned high rankings to and inspecting the

Table 1: Rate of benign (Alexa Top 1M) domain names found in the samples.

	Important	Significant	Suspicious
Distinct	2/410 (0,48%)	8/597 (1,34%)	57/571 (9,98%)
Overall (distinct per sample)	2/1453 (0,13%)	109/3052 (3,57%)	277/12636 (2,19%)

search results. In addition, we used threat investigation channels such as private mailing lists, CERTs, and other sources of contextual information that corroborate the correctness of our findings.

Endpoints from Most Frequent Itemsets. We took the domain names of the C&C appearing in the highest ranked itemsets (top 10) and searched for evidence to support their relevance. Over the past year, `*.vicp.co` (78.96%), `smsgrabber.url.ph`, `y30icv.com` and `124ffsaf.com` are the most relevant ones.

The latter search key lead us to a Trusteer report [10] that confirmed that it was indeed used to collect data stolen from SpyEye bots. `smsgrabber.url.ph` was part of the C&C infrastructure used by the iBanking Malware in late 2014, as confirmed by a technical report by F5 [16]. Interestingly, according to Google Safe Browsing, `*.y30icv.com` is not engaged in any malicious activity, although according to our analysis it is clearly pointing to C&C and APK-hosting server.

As we found no public reports about `*.vicp.co`, we extended our search using PassiveTotal [5] and a private mailing list used by experts to exchange fresh, threat-related information. It was found that we spotted an actively spreading trojan campaign started in December 2014, targeting Korean and Chinese customers. According to the intelligence information at our disposal and to ZeuS Tracker, none of these subdomains were used for non-mobile malicious purposes. In conclusion, the campaign that was brought to our attention, thanks to the frequent itemset mining system of DroydSeuss, is a very relevant one and seems to be exclusively targeting mobile customers. Our findings were later confirmed by the NASK/CERT Polska.

Phone Numbers from Most Frequent Itemsets. Among the most frequent itemsets we found numbers such as `+467694XXXX`, used as C&C in ZitMo campaigns [19], `+79252XXXX`, included in a spyware kit [14] or `+447781XXXX`, used by an iBanking campaign [12] and further confirmed by the NASK/CERT Polska. A curious case is `+49157061XXXX`, a German number used as a C&C endpoint by 20 samples. According to a (cached) WHOIS record, this phone number was of the admin contact the `kundencenter-accountservice.com` domain name. A further search revealed that the number was used in the past to host a PayPal phishing campaign [18].

GCM Endpoints. Among the samples that used GCM endpoints (i.e., sender ID `738965552XXXX`, a C&C server address `94.75.**.**` and the respective domain name), we found one blog post [6] and a project deliverable [9]. The blog post was written by the AndroTotal group and required to manually reverse engineer the samples and extract the concise relevant information extracted automatically by DroydSeuss. Similarly, the project deliverable described the results of an analysis of the sample carried out manually in order to recognize whether the APK was malicious or not.

To summarize, the evidence reported by DroydSeuss led us to finding either (1) automatically generated analysis reports that confirmed their correctness, or (2) manually written technical reports that certainly required human effort. Therefore, we can argue that DroydSeuss extracts data that is correct and useful

for analysts. Of course, this manual effort was required only once to confirm our findings.

3.4 Experiment 3: Runtime Performance

From analyzing 100 apps, we conclude that DroydSeuss completes the analysis of one APK in about 5 minutes on average. The heaviest part is dynamic analysis with an average of 4'40". Static data extraction and ranking are negligible.

4 Limitations and Future Work

The itemset mining process may be subject to specific evasion attempts. By randomizing the *entire* package name an attacker could make DroydSeuss generate one low-frequency itemset per distinct APK. However, this would be against the attacker's goals who wants to mimic the official apps. Indeed, recent work [13] showed that, according to the cyber criminals' modus operandi, the package name is suitable as a lightweight identifier. As a mitigation, other features could be used together with the package name to better characterize a sample.

Since Android malware's sophistication is relatively low compared to PC-based malware, DroydSeuss does not trace native code. Future work should focus on whether this is necessary and, if it is, how this can be implemented efficiently.

DroydSeuss inherits the limitations of dynamic analysis. A sample may not show its (malicious) behavior because some code paths are not reached, it recognizes that it is running in an emulator [17,21] or it employs advanced timing attacks. Our experience indicates that samples need to contact the C&C at least once, which is enough for our purposes. Notwithstanding, as a mitigation, our sandbox strives to stimulate the execution by injecting various events and we also used a patched emulator that resembles a smartphone-like hardware profile.

In addition to addressing the first two limitations, we foresee another research direction. While searching for qualitative evidence to support our findings, we understood that infection campaigns against desktop computers could be used as an early-warning indicator of upcoming mobile infection campaigns. Indeed, we foresee an approach that tracks the HTML content injected by ZeuS (desktop) and checks whether it contains signs of URLs or QR-codes pointing the user to download an APK. Moving from this observation, more advanced pre-infection indicators could be derived in order to alert mobile customers.

5 Conclusions

The community of malware researchers relies on publicly available feeds. Droyd-Seuss is a first step towards a public tracker of mobile botnets in the spirit of ZeuS Tracker. The Android Malware Trckr appeared some months [15] after DroydSeuss, showing once again the importance of such data feeds.

With this work, we showed that the simple yet effective ranking mechanisms that DroydSeuss conveys, can correctly pinpoint relevant active campaigns, concluding that our approach works.

Acknowledgments. The authors are thankful to the reviewers and to MIUR FACE Project No. RBFR13AJFT, Reply CV, and NWO CSI-DHS 628.001.021 for supporting this work.

References

1. Zeus tracker, https://zeustracker.abuse.ch/
2. Alexa top 1M (2015), http://s3.amazonaws.com/alexa-static/top-1m.csv.zip
3. HostIP (2015), www.hostip.info
4. IP to ASN mapping (2015), http://www.team-cymru.org/IP-ASN-mapping.html
5. PassiveTotal (2015), https://www.passivetotal.org
6. andrototal.org: (another) android trojan scheme using google cloud messaging (2015), http://blog.andrototal.org/post/89637972097/another-android-trojan-scheme-using-google-cloud
7. Artturi Lehtiö: C&C-as-a-service: abusing third-party web services as C&C channels (2015), https://www.virusbtn.com/conference/vb2015/abstracts/R-Lehtio.xml
8. Chebyshev, V., Unuchek, R.: Mobile malware evolution: 2013 (2014), http://securelist.com/analysis/kaspersky-security-bulletin/58335/mobile-malware-evolution-2013
9. Delosières, L., Baltatu, M.: D2.3 lightweight malware detector. Tech. rep., Enhanced Network Security for Seamless Service Provisioning in the Smart Mobile Ecosystem (2012), http://www.nemesys-project.eu/nemesys/files/document/deliverables/NEMESYS_Deliverable_D2.3.pdf
10. Heyman, A.: First SpyEye attack on android mobile platform now in the wild (2011), http://www.trusteer.com/cn/node/360
11. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining – a general survey and comparison. SIGKDD Explor. Newsl. 2(1), 58–64 (2000), http://doi.acm.org/10.1145/360402.360421
12. Kafeine: Nitmo? no!dotsjust "iBanking" used by a (the?) neverquest/vawtrak team (2013), http://malware.dontneedcoffee.com/2013/12/nitmo-no-just-ibanking-used-by-the.html
13. Lindorfer, M., Volanis, S., Sisto, A., Neugschwandtner, M., Athanasopoulos, E., Maggi, F., Platzer, C., Zanero, S., Ioannidis, S.: AndRadar: fast discovery of android applications in alternative markets. In: Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 51–71. Springer (2014)
14. Loetprasoetsit, A.: Csd 2013 sso_session_2_14112013 (2013), http://www.slideshare.net/nozumutee/csd-2013-ssosession214112013
15. Łukasz Siewierski: Tweet by maldr0id (2015), https://twitter.com/maldr0id/status/595953612032991232
16. Meller, I.: F5SOC iBanking malware analysis report. Tech. rep., https://devcentral.f5.com/d/f5soc-ibanking-malware-analysis-report
17. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: Hindering dynamic analysis of android malware. In: Proceedings of the Seventh European Workshop on System Security. pp. 5:1–5:6. EuroSec '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2592791.2592796
18. PhishReported: Submission #1531388 - http://kundencenter-accountservice.com (2015), http://www.phishtank.com/phish_detail.php?phish_id=1531388
19. Spasojevic, B.: Android.zeusmitmo (2012), http://www.symantec.com/security_response/writeup.jsp?docid=2012-080818-0448-99&tabid=2
20. Van Der Veen, V.: Dynamic Analysis of Android Malware. Ph.D. thesis (2013), http://tracedroid.few.vu.nl/thesis.pdf
21. Vidas, T., Christin, N.: Evading android runtime analysis via sandbox detection. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. pp. 447–458. ASIA CCS '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2590296.2590325