DOI:
10.1109/TDSC.2017.2739145

Document Version
Peer reviewed version

EC2: Ensemble Clustering and Classification for Predicting Android Malware Families (Supplementary Materials)

Tanmoy Chakraborty, Fabio Pierazzi and V.S. Subrahmanian

1 Malware Family Distribution

We observe that the size of the malware families present in our dataset follows a skewed distribution. In Figure 1, we plot the non-cumulative size distribution of malware families. We notice that out of 156 families, there are 47 families of size 1, 20 families of size 2, and 112 families of size less than 10.

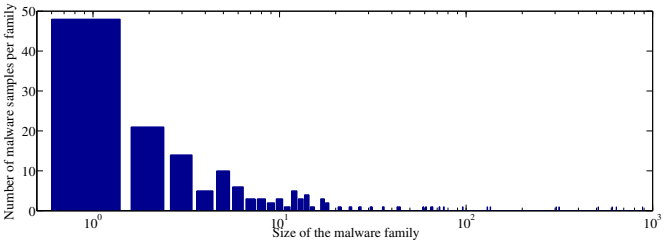


Fig. 1: Number of malware family of certain size.

2 Generation of Malware Dynamic Logs

This section provides more comprehensive details on how the dynamic malware logs have been generated.

The DREBIN malware samples have been dumped into a Ubuntu 16.04 server which was completely disconnected from the Internet in order to avoid malware spread risks. On this server, we installed and configured the official Android emulator[1], which represents a fully-functional Android system that runs application with a graphical user interface. Command-line Linux debug tools such as adb allows to log system events (e.g., READ and WRITE operations of an application) and provide an interface to the Android emulated device.

- *T. Chakraborty is with Dept. of Computer Science and Engineering, Indraprastha Institute of Information Technology, Delhi (IIIT-D), India. F. Pierazzi is with the Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, 41125 Modena, Italy. V.S. Subrahmanian is with Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA.*
  E-mail: tanmoy@iiitd.ac.in, fabio.pierazzi@unimore.it, vs@umiacs.umd.edu

Manuscript received November, 2016; revised –.

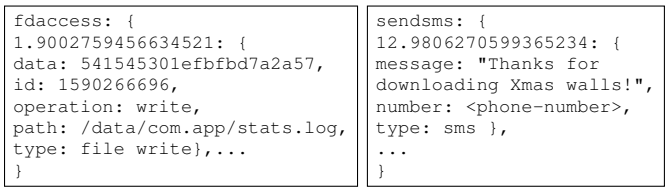
1. https://developer.android.com/studio/run/emulator.html

Since the malware may not trigger its malicious behavior in the absence of network connectivity, we used a network simulator tool purposely intended for malware analysis: `inetsim`[2], that simulates many network services, including HTTP / HTTPS, SMTP / SMTPS, POP3 / POP3S, DNS, FTP, FTPS, TFTP, IRC, NTP. Moreover, `inetsim` also provides realistic files in response to malware requests (e.g., if the malware tries to download an .exe file from an external website).

To run the malware samples, we used `DroidBox 4.1.1`[3], an open-source sandbox especially tailored for dynamic malware analysis of Android application. DroidBox comes with an Android filesystem and RAM that contain some realistic but fake data (e.g., contacts, SMS, phone logs) and that are used to restore the Android emulator to a consistent state after each malware execution. DroidBox builds upon an emulator running an Android 4.1.2 image, which is the late 2012 edition of Android, and is perfectly aligned with the DREBIN dataset that spans from 2010 to 2012. DroidBox can install and execute `apk` applications in the Android emulator. For each malware sample in the DREBIN dataset, we execute the malware for *120 seconds* and log all its activities through DroidBox, which include: file system activities (read/write), network activity (send-net/recvnet), usage of cryptographic primitives, dynamic loading of classes, start of new services (background processes), generation of system events. After the execution of each malware, the emulator is restored to the initial state, so that the execution of the next malware does not pollute the environment. In this paper we consider the $4,845$ samples that were executed without crashes during the 120 seconds time span. The execution of the overall dataset spanned over about three weeks.

Figures 2 report an example related to logs generated by execution in DroidBox. Figure 2(a) reports a log of a *write* operation on the filesystem, where after about $1.9s$ from starting the execution of the malware on DroidBox, some data is written in `/data/com.app/stats.log`. The field named "data" in Figure 2(a) contains the data written by the application in hexadecimal format. In Figure 2(b) we report another example of a log related to SMS sending activity.

2. http://www.inetsim.org
3. https://github.com/pjlantz/droidbox



(a) (b)

Fig. 2: Example of (a) write-log and (b) SMS activity log generated by DroidBox.

Many malware try to deceptively send SMS to premium-rate numbers owned by the malware developer in order to get money from the user. Sometimes, SMS are also used to spread the malware by sending a text to users from the contact list to deceive them into clicking a malicious URL.

3 Classification performance

In this section, we report the detailed raw values of classification performance discussed in Section 6.

3.1 All families

Table 1 reports the detailed results of supervised classifiers comparison in which we consider *all malware families*, including singleton families consisting of only one malware sample. Results are computed with 5-fold cross-validation and averaged over 50 iterations. Table 1 reports results for Decision Tree (DT), K-NearestNeighbor (K-NN), Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machines (SVM) and Random Forest (RF). Results in Table 2 refer also to three set of features: only static, only dynamic, and both static and dynamic. Reported results correspond to Micro F-Score (MiF), Macro F-Score (MaF), Micro AUC (MiAUC) and Macro AUC (MaAUC). Formal definitions of these metrics are reported in Section 6.

The purpose of this experiment is to show how supervised classification performance decreases significantly when also considering also small families. As expected, Table 1 shows that the worst performance are associated with Macro statistics, because they are computed as an average of the characteristics of each family. Hence, the low performance on small families and singletons decrease the Macro performance.

3.2 Families with at least 10 samples

Table 2 reports the 5-fold results averaged over 50 iterations with all the classifiers considered in the previous sections. In particular, it reports results for Decision Tree (DT), K-NearestNeighbor (K-NN), Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machines (SVM) and Random Forest (RF). Results in Table 2 refer also to three set of features: only static, only dynamic, and both static and dynamic. Reported results correspond to Micro F-Score (MiF), Macro F-Score (MaF), Micro AUC (MiAUC) and Macro AUC (MaAUC). Formal definitions of these metrics are reported in Section 6.

As expected, the classifiers performance on families with at least 10 samples are higher to performance obtained when considering *all* families. This is especially evident in the Macro statistics. Moreover, Table 2 shows that although some of the classifiers achieve better performance with only dynamic features, the best overall classification performance is achieved by RF with both static and dynamic features.

TABLE 1: Raw performance on the classifiers on different feature sets, while considering all families (also singleton families with only one sample).

	Classifier	MiF	MaF	MiAUC	MaAUC
Static	DT	0.91	0.63	0.96	0.85
	K-NN	0.44	0.17	0.72	0.59
	LR	0.21	0.01	0.60	0.50
	NB	0.28	0.12	0.63	0.58
	SVM	0.38	0.15	0.69	0.56
	RF	0.94	0.71	0.97	0.87
Dynamic	DT	0.89	0.54	0.95	0.80
	K-NN	0.76	0.32	0.88	0.66
	LR	0.85	0.50	0.92	0.75
	NB	0.86	0.54	0.93	0.81
	SVM	0.53	0.12	0.76	0.55
	RF	0.89	0.54	0.94	0.79
Stat.+Dyn.	DT	0.92	0.58	0.96	0.83
	K-NN	0.45	0.17	0.72	0.59
	LR	0.36	0.06	0.67	0.53
	NB	0.29	0.15	0.64	0.60
	SVM	0.41	0.16	0.70	0.57
	RF	0.93	0.64	0.97	0.83

TABLE 2: Raw performance on the classifiers on different feature sets, while considering only families with at least 10 samples.

	Classifier	MiF	MaF	MiAUC	MaAUC
Static	DT	0.94	0.85	0.97	0.92
	K-NN	0.46	0.26	0.72	0.62
	LR	0.23	0.03	0.60	0.51
	NB	0.29	0.13	0.64	0.58
	SVM	0.39	0.23	0.69	0.59
	RF	0.94	0.88	0.97	0.92
Dynamic	DT	0.92	0.75	0.96	0.88
	K-NN	0.80	0.55	0.90	0.75
	LR	0.89	0.74	0.95	0.84
	NB	0.89	0.79	0.95	0.91
	SVM	0.56	0.25	0.78	0.60
	RF	0.92	0.78	0.96	0.88
Stat.+Dyn.	DT	0.94	0.85	0.97	0.92
	K-NN	0.47	0.28	0.73	0.63
	LR	0.39	0.14	0.69	0.56
	NB	0.34	0.20	0.66	0.62
	SVM	0.42	0.30	0.70	0.60
	RF	0.95	0.89	0.97	0.93

3.3 Binary classification

In this section we report a table with the raw binary classification performance. Table 4 reports results for the Random Forest classifier, for different feature combinations. In most cases, performance are even higher than the multi-class classification, although some samples (e.g., `Zitmo`) are harder to classify correctly due to stronger obfuscation techniques.

TABLE 3: Raw performance on the clustering algorithms on different feature sets (static, dynamic, static+dynamic).

	Clustering	MiF*	MaF*	NMI	ARI	PU
Static	DBSCAN	0.17	0.80	0.61	0.10	0.46
	Hierarchical	0.22	0.61	0.71	0.20	0.40
	Affinity	0.11	0.66	0.63	0.10	0.27
	K-Means	0.15	0.68	0.70	0.19	0.48
	MeanShift	0.16	0.72	0.29	0.03	0.10
Dynamic	DBSCAN	0.15	0.86	0.46	0.03	0.36
	Hierarchical	0.12	0.43	0.66	0.10	0.27
	Affinity	0.11	0.62	0.58	0.07	0.29
	K-Means	0.15	0.67	0.59	0.09	0.42
	MeanShift	0.16	0.80	0.25	0.04	0.09
Stat.+Dyn.	DBSCAN	0.17	0.90	0.44	0.02	0.44
	Hierarchical	0.10	0.35	0.67	0.10	0.24
	Affinity	0.11	0.57	0.66	0.13	0.36
	K-Means	0.15	0.60	0.68	0.14	0.46
	MeanShift	0.16	0.82	0.22	0.03	0.07

4 Clustering performance

In this section we report the detailed clustering performance results of Section 8. In particular, Table 3 reports the results of the following clustering algorithms: DBSCAN, Hierarchical Clustering (we report the best results achieved with *CompleteLinkage*), Affinity Clustering, K-Means, MeanShift. We report the performance in terms of the following clustering metrics: Micro F-Score (MiF*), Macro F-Score (MaF*), Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), and Purity (PU). Description of these metrics are reported in Section 8.

5 Malware family characterization

Apart from the overall discriminative features obtained from the multi-class classification, it is interesting to identify the *specific* features that characterize and distinguish *each* malware family from the rest. To this end, we design the following experimental setup – for each of the 44 large malware families, we learn the best *binary* classifier (Random Forest) to distinguish each family from the rest, such that there exist 44 set of classification rules that separate the behavior of one malware family from all others. In this way, for each family we can sort the features by decreasing *weight* learned by the classifier, thus ranking the features that are more relevant for classification.

Figure 3 shows some relevant examples of top 5 features for families of different sizes: some large families and some small families.[4] Each row in Figure 3 corresponds to different family; and each histogram reports feature values on the X-axis, and the fraction of malware samples (in %) having that feature value on the Y-axis. In particular, we report results about the following families:

- `FakeInstaller` pretends to be an app that installs (or uninstalls) other apps from the system, whereas instead its primary purpose is to send premium-rate SMS without the user consent.
- `Opfake` is another popular malware family that tries to send premium-rate SMS.
- `GinMaster` is a Trojan that tries to perform a *privilege escalation* on the device, then steals information through background processes, and sends the stolen data to a remote website.
- `Jifake` is another malware family trying to send premium-rate SMS, and often tries to perform aggressive advertisement and to install unwanted toolbars.
- `MobileTx` is a Trojan that both steals information and also tries to send premium-rate SMS.
- `Geinimi` is a Trojan that opens a backdoor to send information from the device to a remote server.

Despite the fact that some of these families share a similar purpose (e.g., sending premium-rate SMS), each of them is characterized differently through the top 5 features shown in Figure 3.

FakeInstaller. Four of the top 5 features in `FakeInstaller` are static. The top 5 features distributions are reported in the first row of Figure 3. The first feature is related to a permission accessing the network state (i.e., whether at a certain moment a connection is available or not on the device). This feature is important for discriminating `FakeInstaller` as it is one of the few malware families that does *not* require it (i.e., where permission `ACCESS_NETWORK_STATE` is set to 0, as can be observed in Figure 3). The `filesize` of `FakeInstaller` samples is usually lower than the other families. Moreover, the `recvsaction DATA_SMS_RECEIVED` is used to monitor whenever an SMS is received, and is probably used by `FakeInstaller` to remove evidences of its texts sent to premium-rate number and to abort or delete incoming SMS. Moreover, the overall number of permissions required by `FakeInstaller` is lower than the average for the other malware categories (less than 15 permissions, of only about 5 of which are marked as *dangerous*). This is probably because this type of family is very targeted for SMS fraud.

Opfake. Although `Opfake` shares a similar goal as `FakeInstaller` (i.e., SMS fraud), in the second row of Figure 3 we see that its top 5 distinguishing features are different. In particular, the most relevant feature to distinguish Opfake is related to the `author` attribute (each author found in the dataset is mapped to an integer value on the X-axis of Figure 3). This is because the classifier discriminates that the majority of `Opfake` variants have been signed with the same signature. Out of the 607 samples in Opfake, 496 samples have author=`697`, 60 samples have author=`740`, and 40 samples have author=`40`, and others have different authors. `Opfake` also modifies the `settings.xml` file, which is not touched by the majority of other malware families. It also loads an apk in memory, which may be a repacked application that contains the actual malicious code. Finally, a peculiarity of `Opfake` is that its Activies (i.e., windows shown to the user) are very few – less than 10, whereas most malware have at least 20. Its `filesize` is also usually smaller than other families

GinMaster. The `GinMaster` family is better discriminated through dynamic features, since all its top 5 features shown in Figure 3 are dynamic. The first feature is `read cpuinfo`, and is probably used to determine the hardware characteristics of the mobile device. This infor-

4. The average F-score for the binary classification is even higher than multi-class classification, and is reported in supplementary materials.

TABLE 4: Raw performance per-family for binary classification problem for the families with at least 10 samples in the DREBIN dataset (Random Forest classifier). Results are ordered by decreasing family size.

Family	Static				Dynamic				Stat. + Dyn.			
	MiF	MaF	MiAUC	MaAUC	MiF	MaF	MiAUC	MaAUC	MiF	MaF	MiAUC	MaAUC
FakeInstaller	0.98	0.98	0.98	0.98	0.95	0.95	0.96	0.96	0.97	0.97	0.98	0.98
DroidKungFu	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.98
Opfake	0.99	0.99	0.99	0.99	0.97	0.97	0.98	0.98	0.99	0.99	0.99	0.99
Plankton	0.98	0.98	0.98	0.98	0.90	0.90	0.93	0.93	0.97	0.97	0.97	0.97
BaseBridge	0.95	0.95	0.95	0.95	0.95	0.95	0.96	0.96	0.94	0.94	0.94	0.94
GinMaster	0.96	0.96	0.96	0.96	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99
Iconosys	1.00	1.00	1.00	1.00	0.69	0.69	0.79	0.79	0.95	0.95	0.96	0.96
FakeDoc	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99
Kmin	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00
Adrd	0.92	0.92	0.93	0.93	0.94	0.94	0.95	0.95	0.95	0.95	0.96	0.96
DroidDream	0.83	0.83	0.85	0.85	0.92	0.92	0.93	0.93	0.94	0.94	0.95	0.95
Glodream	0.93	0.93	0.94	0.94	0.89	0.89	0.90	0.90	0.92	0.92	0.93	0.93
ExploitLinuxLotoor	0.69	0.69	0.78	0.78	0.74	0.74	0.80	0.80	0.69	0.69	0.78	0.78
FakeRun	0.95	0.95	0.98	0.98	0.93	0.93	0.94	0.94	0.99	0.99	0.99	0.99
SendPay	0.98	0.98	0.98	0.98	0.91	0.91	0.92	0.92	0.97	0.97	0.97	0.97
Gappusin	0.73	0.73	0.80	0.80	0.70	0.70	0.82	0.82	0.80	0.80	0.84	0.84
Imlog	0.96	0.96	0.97	0.97	0.76	0.76	0.88	0.88	0.96	0.96	0.97	0.97
Yzhc	0.99	0.99	0.99	0.99	0.94	0.94	0.96	0.96	0.98	0.98	0.99	0.99
SMSreg	0.75	0.75	0.83	0.83	0.47	0.47	0.68	0.68	0.54	0.54	0.73	0.73
Jifake	0.94	0.94	0.96	0.96	0.50	0.50	0.69	0.69	0.94	0.94	0.96	0.96
Boxer	0.60	0.60	0.79	0.79	0.06	0.06	0.52	0.52	0.45	0.45	0.72	0.72
MobileTx	1.00	1.00	1.00	1.00	0.88	0.88	0.91	0.91	0.94	0.94	0.95	0.95
Xsider	0.88	0.88	0.92	0.92	0.85	0.85	0.92	0.92	0.89	0.89	0.92	0.92
Fakelogo	0.88	0.88	0.93	0.93	0.86	0.86	0.89	0.89	0.86	0.86	0.88	0.88
Dougalek	0.93	0.93	0.94	0.94	0.00	0.00	0.50	0.50	0.83	0.83	0.88	0.88
Fatakr	1.00	1.00	1.00	1.00	0.73	0.73	0.82	0.82	0.86	0.86	0.90	0.90
FakePlayer	1.00	1.00	1.00	1.00	0.81	0.81	0.93	0.93	1.00	1.00	1.00	1.00
FoCobers	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Typstu	1.00	1.00	1.00	1.00	0.88	0.88	0.90	0.90	0.79	0.79	0.85	0.85
SerBG	0.92	0.92	0.93	0.93	0.96	0.96	0.97	0.97	0.96	0.96	0.97	0.97
Steek	0.83	0.83	0.88	0.88	0.96	0.96	0.97	0.97	0.96	0.96	0.97	0.97
Zitmo	0.59	0.59	0.75	0.75	0.36	0.36	0.63	0.63	0.60	0.60	0.77	0.77
Penetho	0.96	0.96	0.97	0.97	0.62	0.62	0.77	0.77	0.79	0.79	0.85	0.85
Nandrobox	0.89	0.89	0.92	0.92	0.73	0.73	0.85	0.85	0.85	0.85	0.88	0.88
Geinimi	0.69	0.69	0.78	0.78	0.81	0.81	0.85	0.85	0.63	0.63	0.78	0.78
FakeTimer	0.79	0.79	0.83	0.83	0.65	0.65	0.78	0.78	0.80	0.80	0.87	0.87
Copycat	0.93	0.93	0.95	0.95	0.20	0.20	0.60	0.60	0.29	0.29	0.62	0.62
Placms	0.53	0.53	0.70	0.70	0.96	0.96	0.97	0.97	1.00	1.00	1.00	1.00
TrojanSMS.Hippo	0.70	0.70	0.78	0.78	0.83	0.83	0.87	0.87	0.69	0.69	0.82	0.82
Fakengry	0.53	0.53	0.70	0.70	0.33	0.33	0.65	0.65	0.43	0.43	0.68	0.68
Cosha	0.73	0.73	0.80	0.80	0.60	0.60	0.75	0.75	0.80	0.80	0.85	0.85
SMSZombie	1.00	1.00	1.00	1.00	0.87	0.87	0.90	0.90	0.87	0.87	0.90	0.90
DroidSheep	0.87	0.87	0.90	0.90	0.00	0.00	0.50	0.50	0.00	0.00	0.50	0.50
Vdloader	0.67	0.67	0.80	0.80	0.00	0.00	0.50	0.50	0.93	0.93	0.95	0.95

mation is used by GinMaster to determine if the device is susceptible to rooting exploits that allow the malware to get root privileges on the smartphones. The write operations (up to 20 operations per file) on files such as `icons_android.ico`, `icons_com.svox.pico.ico` and `installer.xml` are probably performed to attempt privilege escalation and get full access on device data. The `recvsaction USER_PRESENT` corresponds to the registration of a listener used to check whether the Android user is using the Android device in a certain moment (e.g., whether the keyguard is on/off). In this way, `GinMaster` can perform its actions in a deceptive way that goes undetected by the Android phone user. Moreover, from Figure 3 we can observe that all the top 5 features of GinMaster are peculiar of this family, and all the other families have the values of the features set to zero.

Jifake. The fourth row of Figure 3 shows that `Jifake` malware requires some peculiar permissions that are requested rarely by malware families. The permission `EXPAND_STATUS_BAR` allows an app to expand or contract the status bar, and is usually used only by system applications. The same consideration holds for permissions `JiFake` to read and write the calendar. The `READ_FRAME_BUFFER` permissions allows to take screenshots of other applications, whereas `CONTROL_LOCATION_UPDATES` allows it to modify whether phone location is updated or not. We thus observe that though the primary purpose of `Jifake` is to send premium-rate SMS, these permissions are involved with an additional behavior of some Jifake variants that try

to perform aggressive advertisement (Adware) and install unwanted system toolbars.

MobileTx. The fifth row of Figure 3 reports the top 5 distinguishing features of `MobileTx` family, an SMS-fraud malware. Figure 3 shows that three out of the top 5 features involve `sendsms` activity to premium-rate numbers with pre-determined text content – this is captured by dynamic analysis. For example, in the first feature `1065-71090-88877` turns out to correspond to a premium-rate number used to steal money from the victim. (In the whole DREBIN dataset, we have found 68 premium numbers used among several variants). Moreover, all variants of `MobileTx` require the `RESTART_PACKAGES` permission, that allows the malware to kill other processes on the Android device, and is probably used to kill process monitors and antivirus software possibly installed in the device. The histogram in Figure 3 shows that some other malware families (blue bar) also require this permission for the same purpose, but it is not common. Finally, all 21 variants of `MobileTx` have the same `author`=443 (that is the anonymized integer for the SHA256 signature), and hence this feature also helps identify this family.

Geinimi. The `Geinimi` malware family usually steals information such as fine location, device IDs (e.g., IMEI, IMSI) and the list of installed apps. Its top 5 distinguishing feature value distributions are shown in the last row of Figure 3. The first feature is associated with the registration of `AdServiceReceiver` that is a component used to monitor system events on which to trigger malicious behavior (e.g., to send stolen data to remove server). The `crypto` operations identified in the top 5 features use the DES algorithm to *decrypt* URLs of the CnC servers and GET commands which are encrypted to avoid static code analysis, but the malware authors recycled decryption keys and hence we detected them during dynamic analysis. The permission `MOUNT_UNMOUNT_FILESYSTEMS` is used to access data in SD card slots, as some variants of `Geinimi` use SD cards to store and load repackaged malicious applications download from the CnC servers. The `servicestart GoogleKeyboard` represents the launch of a background process named `GoogleKeyboard` (to avoid user detection by looking at the phone task manager) that performs malicious actions, steals information periodically and sends it to the CnC whose URLs are decrypted at runtime.

As a final remark, we note that for some families, static features are more discriminative (e.g., `FakeInstaller` and `JiFake`), whereas some others are better distinguished through dynamic features (e.g., `GinMaster` and `MobileTx`). These statistics can be used by security analysts to define new signatures for malware classification as they automatically reveal some internals of the code obfuscation techniques adopted for a certain malware family. For example, the number of permissions requested in the `FakeInstaller` family (both standard and dangerous) can be used as an indicator, whereas in `MobileTx` family the `author` and the `sendsms` activities can reveal malicious behavior.

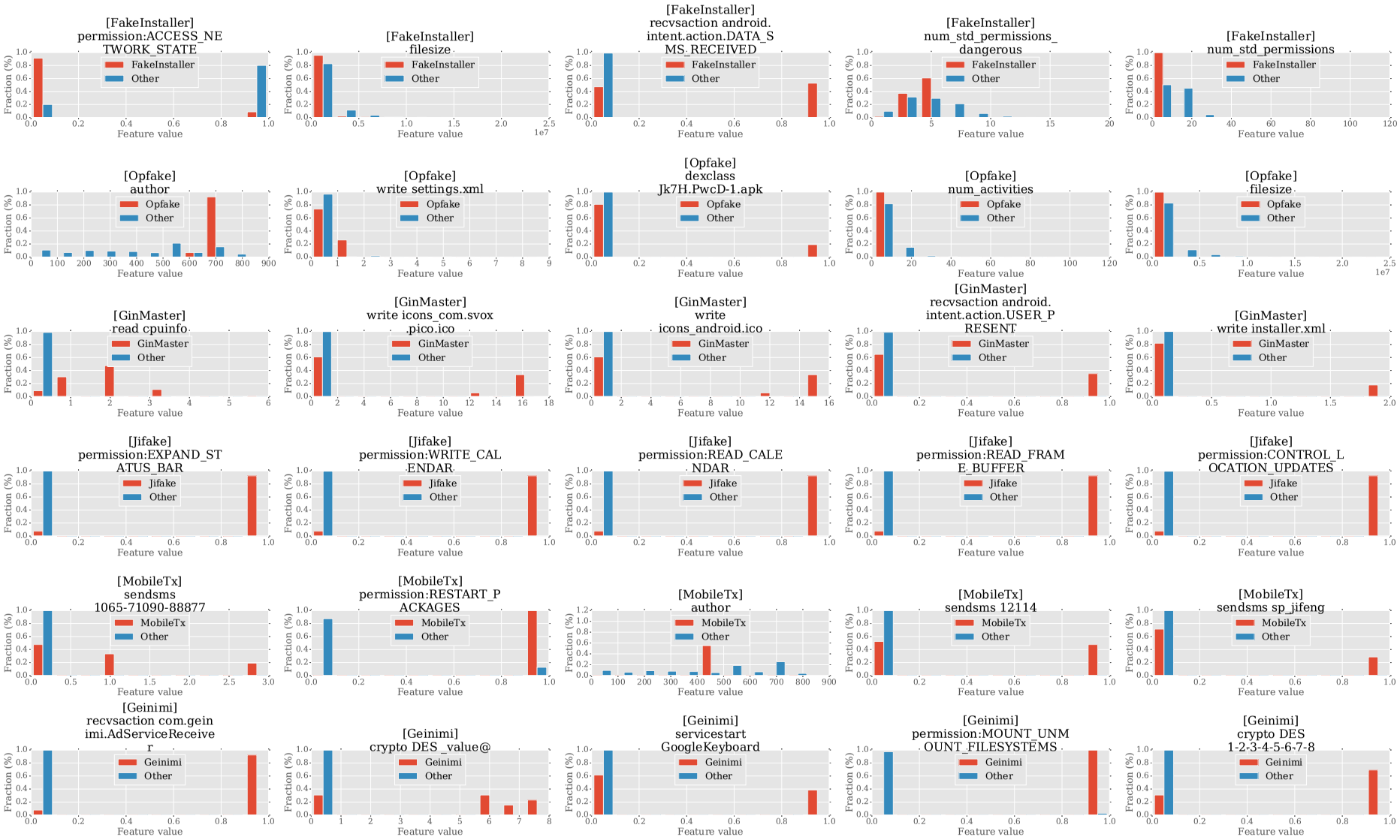


Fig. 3: Feature value comparison between malware families. In particular, we report the histogram of the top-5 feature values for the following malware families (number of samples within parenthesis): FakeInstaller (883), Opfake (607), Geinimi (13), MobileTx (21), GinMaster (304), Jifake (27). Each row corresponds to a family, and the feature value is compared against all other malware families in the dataset.