CODE MODELS FOR **SECURE** AND DEBUGGABLE SYSTEMS

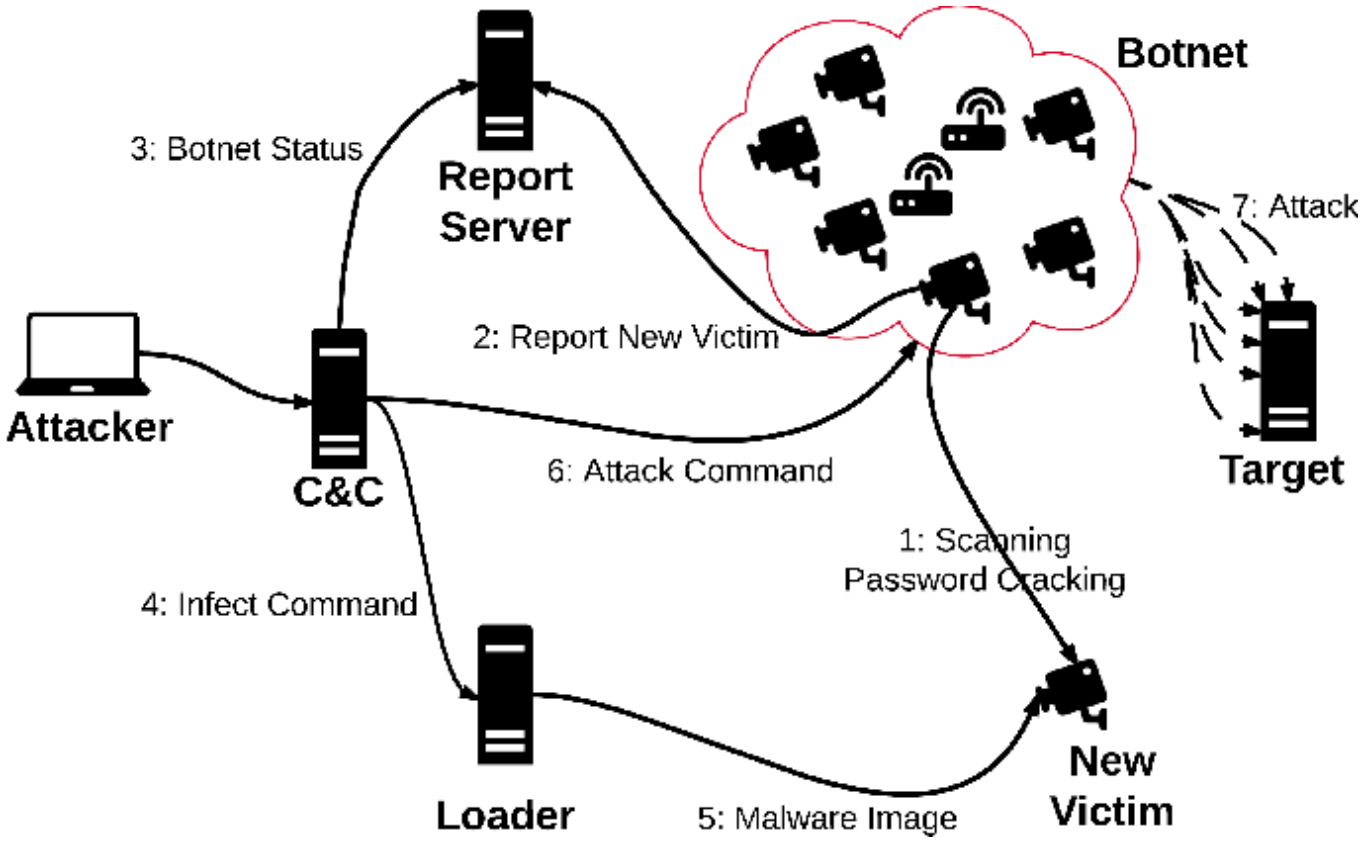
Fiorella Artuso, **Giuseppe Antonio Di Luna**

A DAY IN OUR LIFE

A PANOPLY OF THREATS

- Banking data
- Identity theft
- Surveillance

A PANOPLY OF THREATS



Mirai Botnet:
DDoS attack on 20 September 2016 on the Krebs on Security site which reached 620 Gbit/s.
Ars Technica also reported a 1 Tbit/s attack on French web host OVH.

A PANOPLY OF THREATS

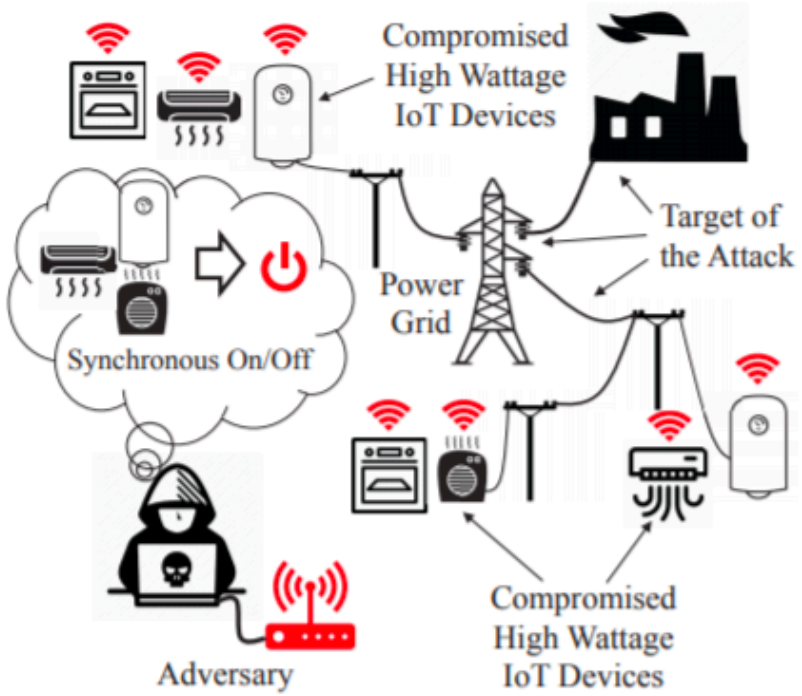
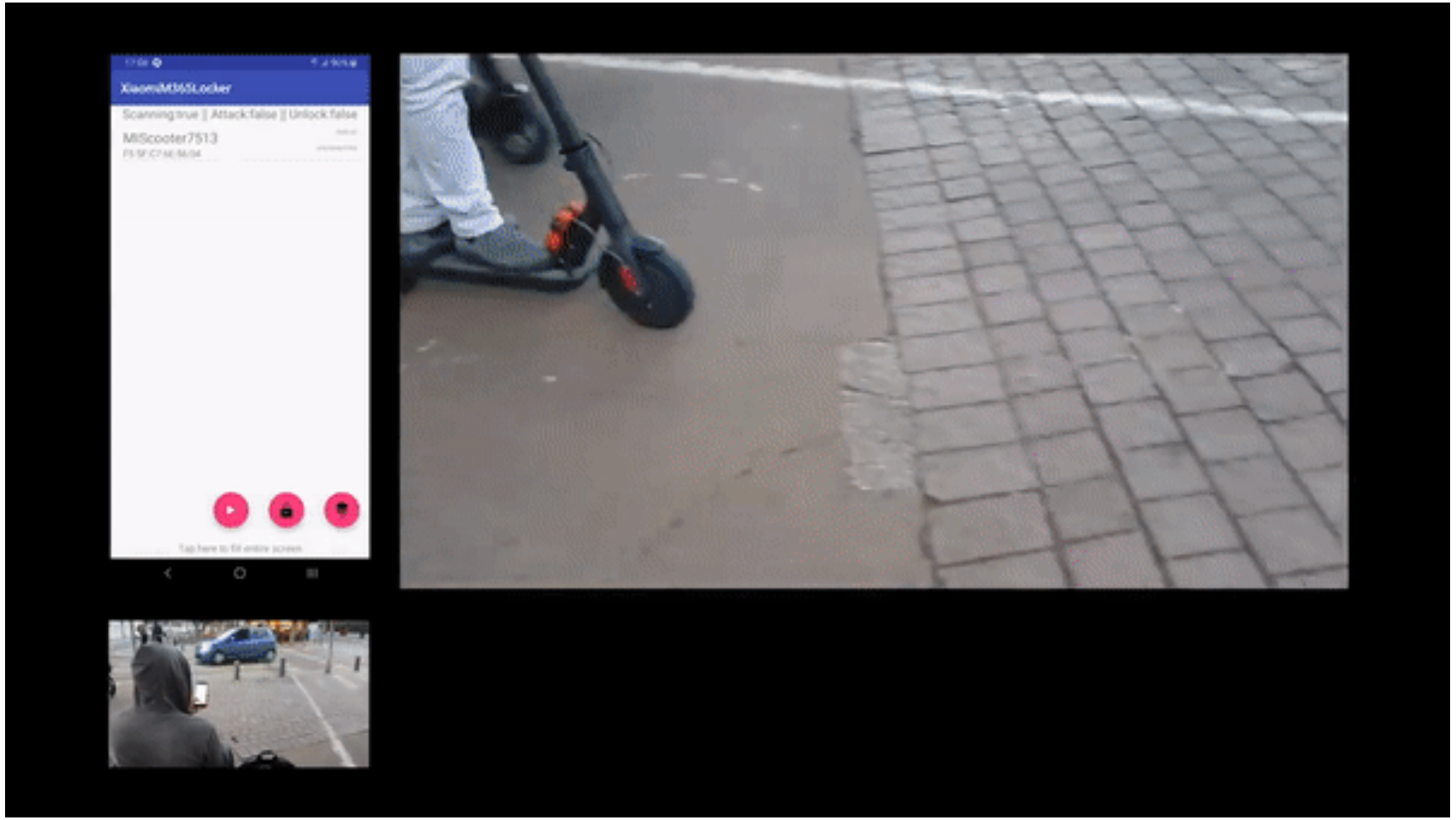


Figure 1: The MadIoT attack. An adversary can disrupt the power grid's normal operation by synchronously switching on/off compromised high wattage IoT devices.

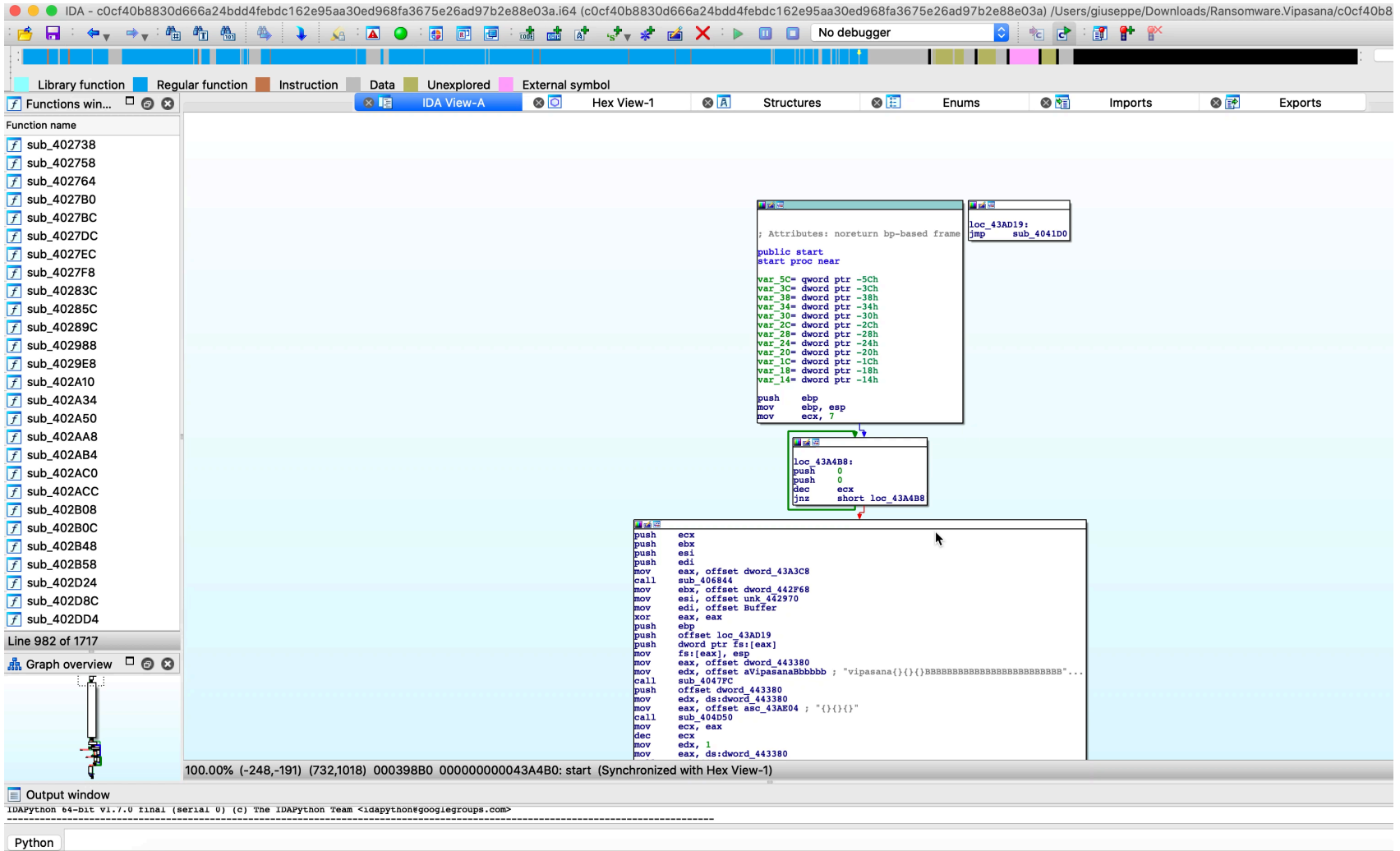
Botnets causing blackouts: how coordinated load attacks can destabilize the power grid. [ACSAC 2017]

A PANOPLY OF THREATS

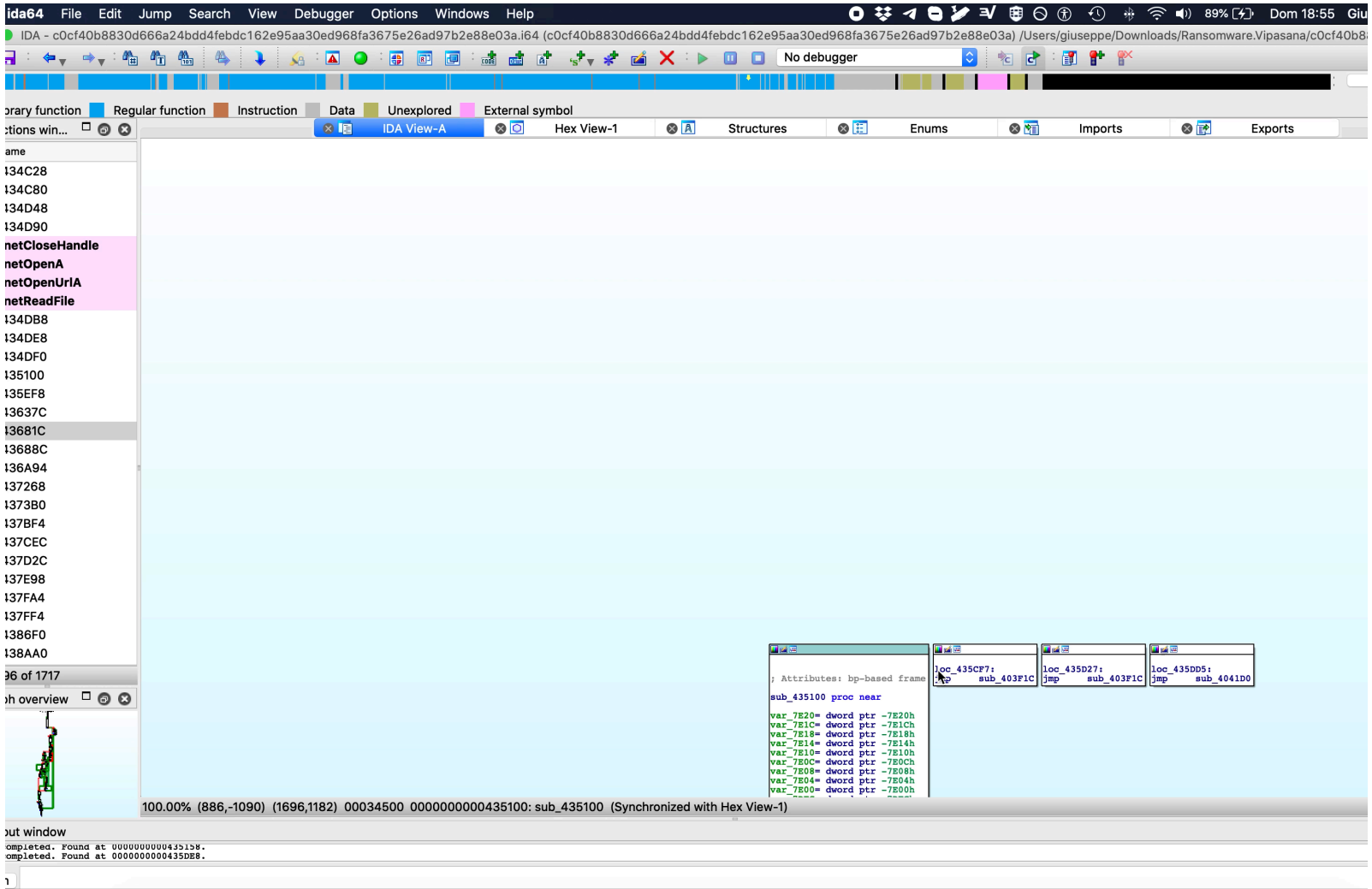


MALWARE, VULNERABILITIES AND FIRMWARES: THE HARD LIFE OF A SECURITY RESEARCHER

REVERSING A MALWARE



REVERSING A MALWARE



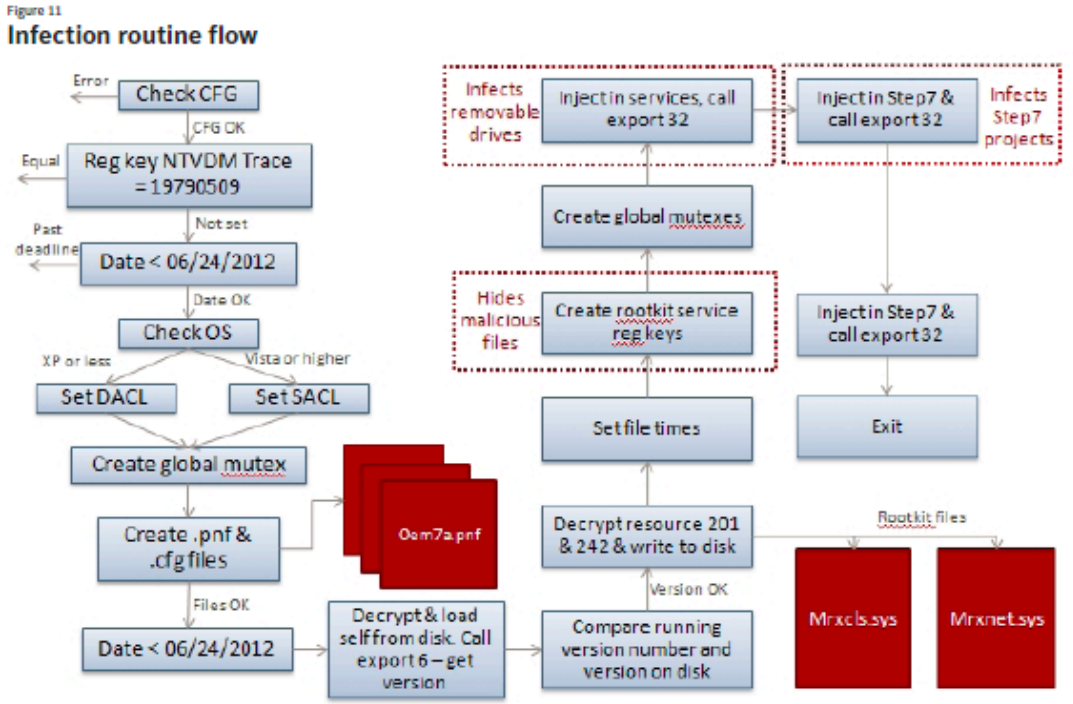
AND THAT WAS EASY....

- Malware binaries are often «stripped» removing functions and variables names;
- The strings inside a binary can be encrypted with a custom encryption algorithm;
- The libraries can be statically imported and can be indistinguishable from malware code;
- Packing;
- The refined ones have inside a virtual machine that runs the actual malware code.

..IN DOUBT RUN IT?

..IN DOUBT RUN IT? NOT SO SIMPLE!

A lot of malware have anti-vm technques
Other are tailored against a target and show a normal behaviour;

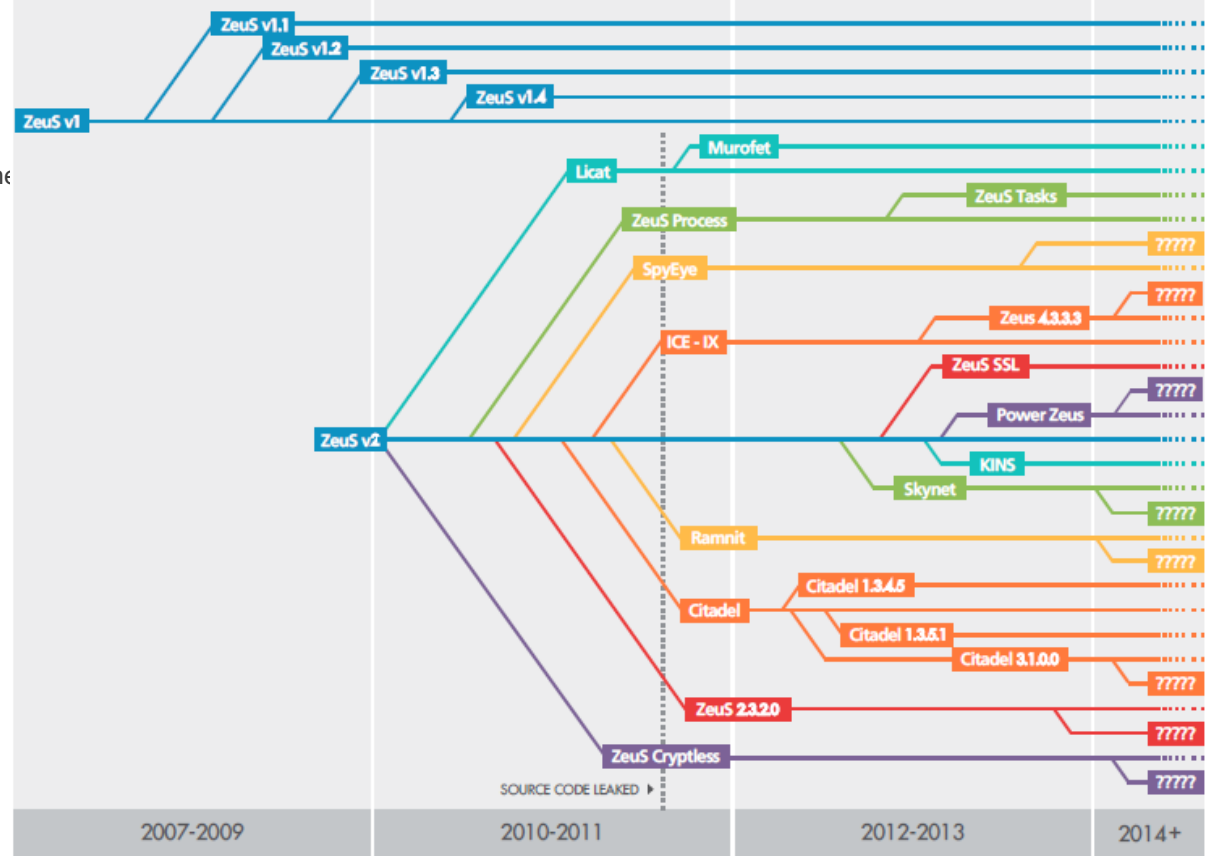


https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf

Understanding Linux Malware https://reyammer.io/publications/2018_oakland_linuxmalware.pdf

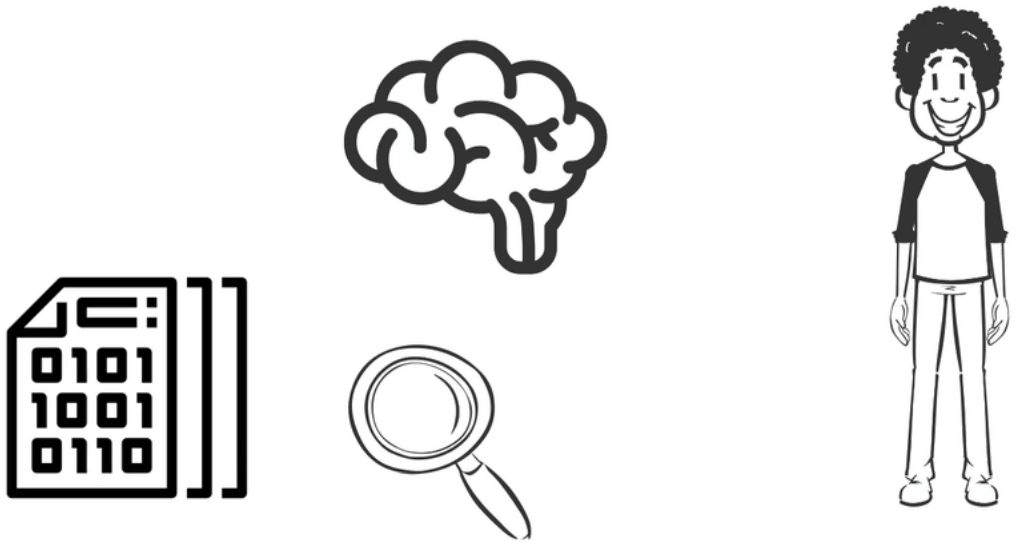
REVERSING REAL-WORLD BINARIES IS A NIGHTMARE

Malware developers produce variants to minimize the



REVERSING REAL-WORLD BINARIES IS A NIGHTMARE

It requires skilled people and a lot, a lot, and I mean a lot of time

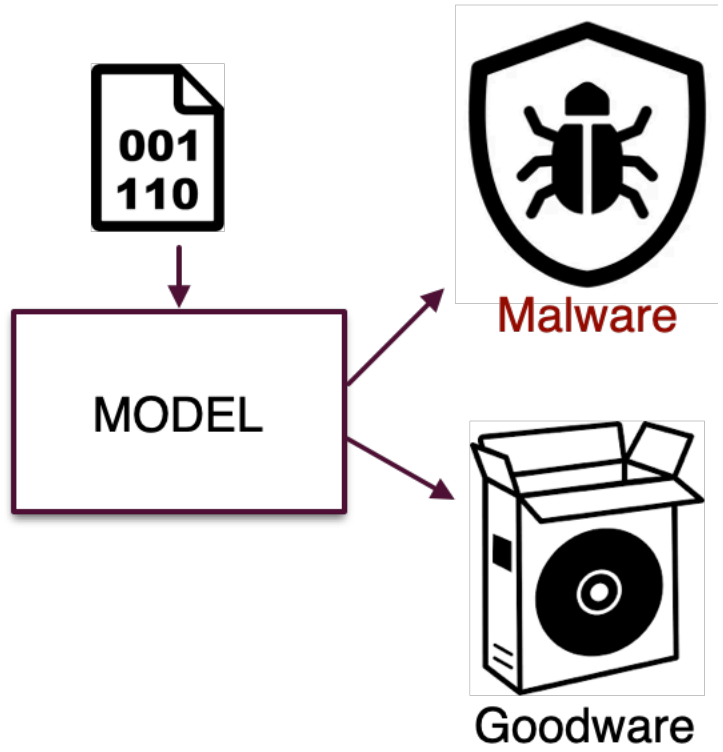


MACHINE LEARNING: A LIFESAVER?

MACHINE LEARNING FOR DETECTING MALWARES

Binary classification problem: Malware or Not

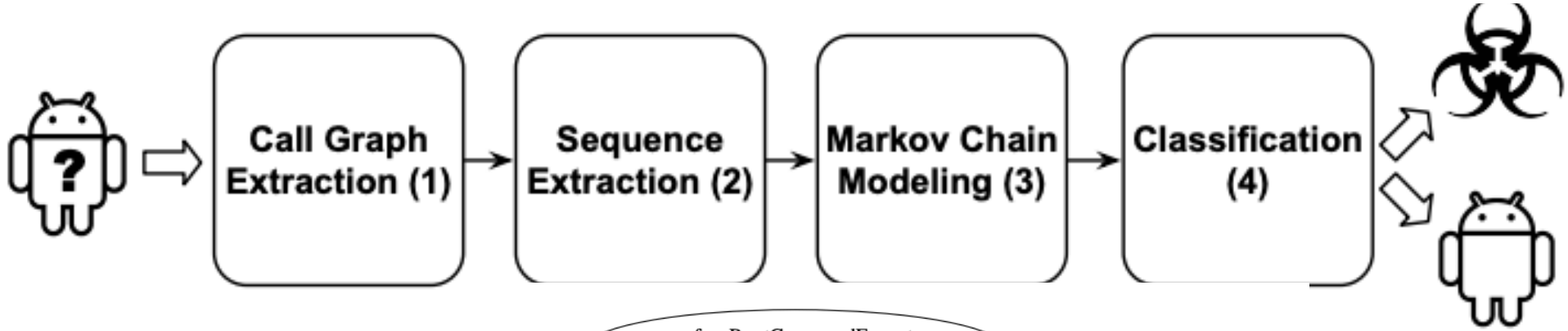
- Thousands of papers using thousands of techniques (DNN, Random Forest, SVM, ...) (see [0] for a recent suvey);
- Static, Dynamic or Hybrid Approach;
- Extracting features[1,2]: system call, functions calls, strings, file name, dns requests, ...
- Without manual features: MalvConv [3]



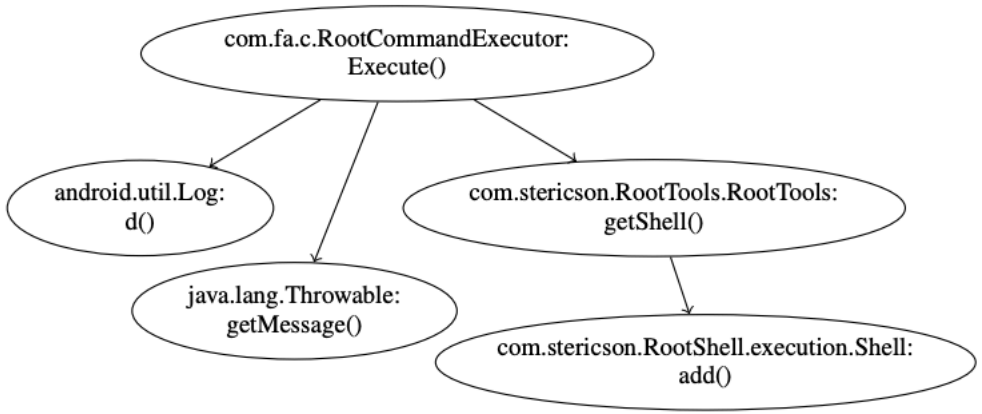
[0] Ucci et al. Survey of Machine Learning Techniques for Malware Analysis. 2018
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017
[2] Arpi et Al.. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS 2014
[3] E. Raff et al.. Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435, 2017

MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



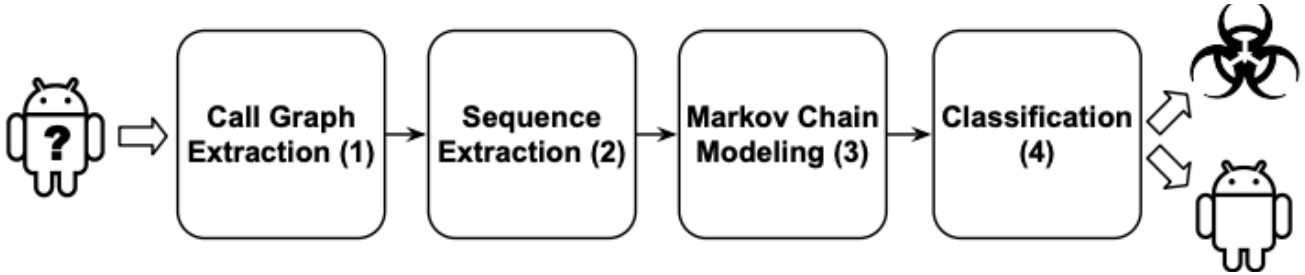
Call Graph Extraction:



Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

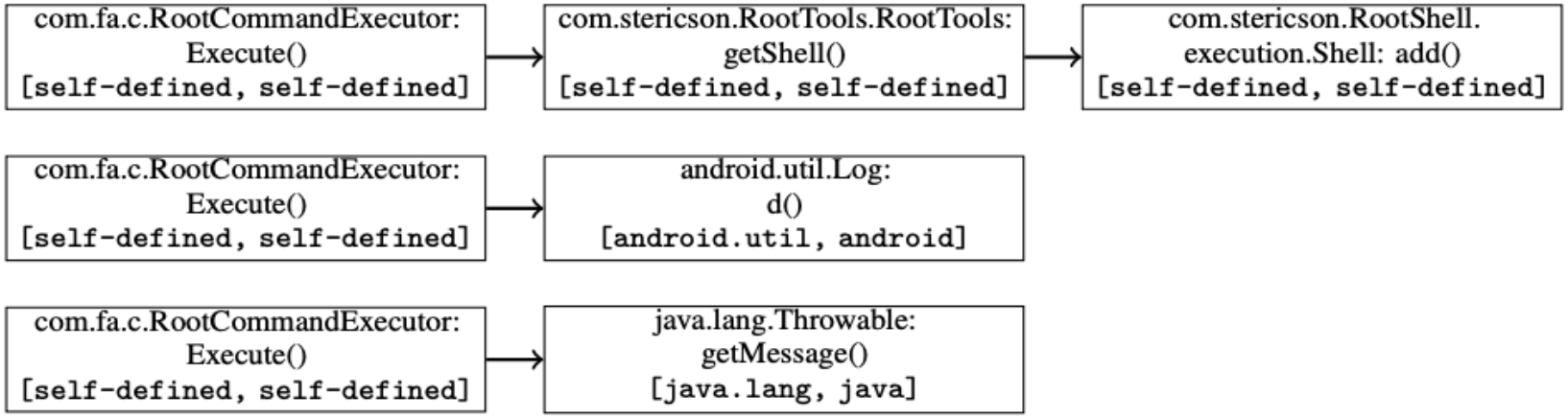
MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



Sequence Extraction:

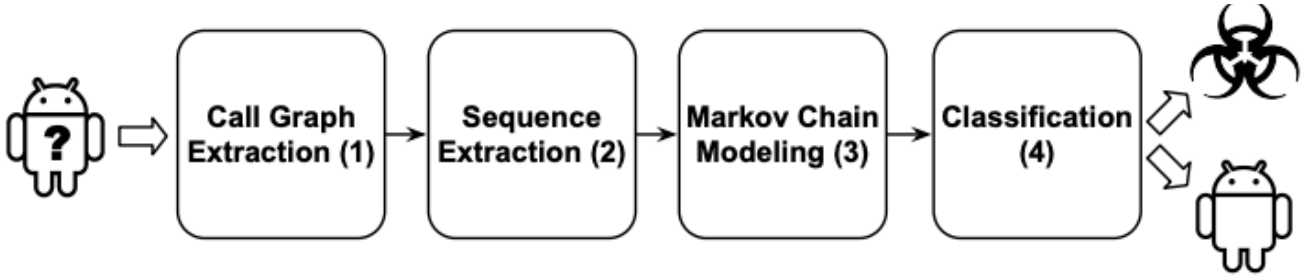
From the call graph we extract Sequences.
The important information is the package.



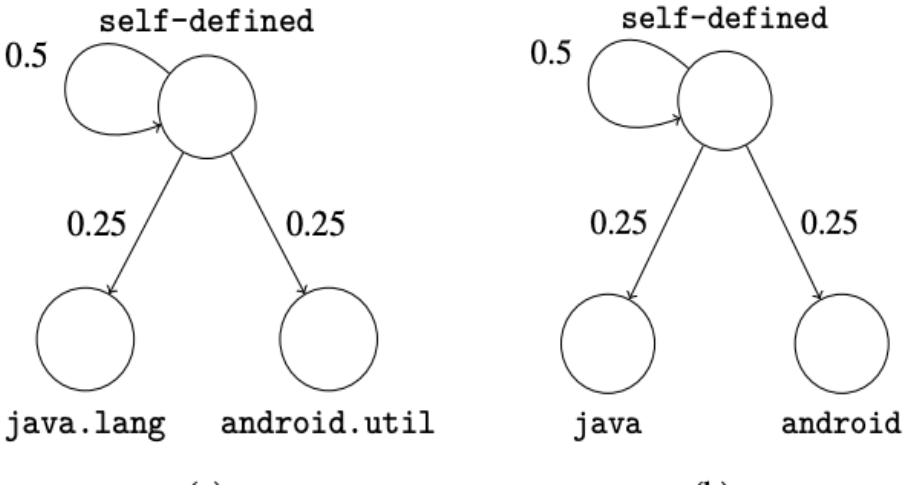
Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



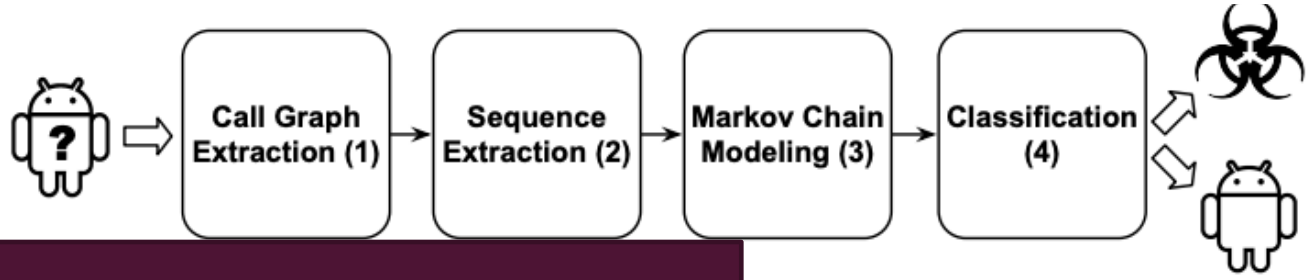
The sequence of calls is transformed in a direct graph. Each sequence becomes a path on the graph.
The graph is weighted, the weight on each edge (A,B) is the normalized frequency the package A calls the package B.
This can be interpreted as a probability measure that A calls B (this graph is a markov chain).



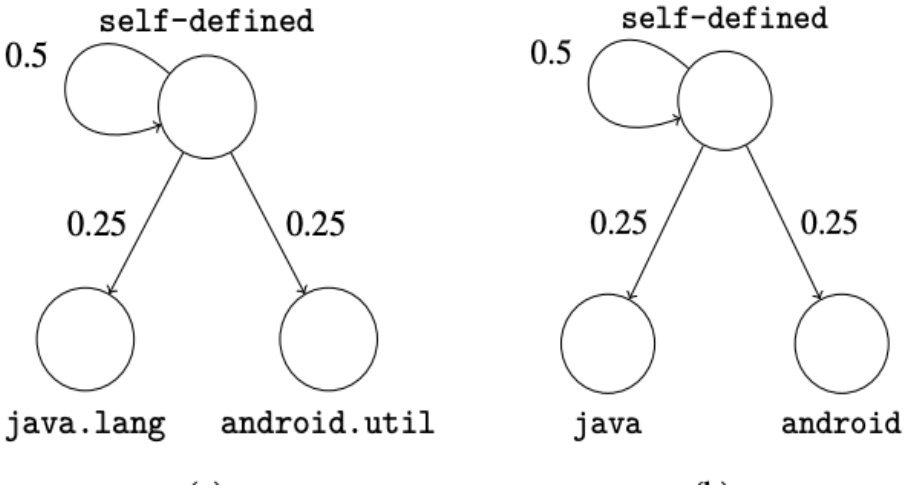
Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



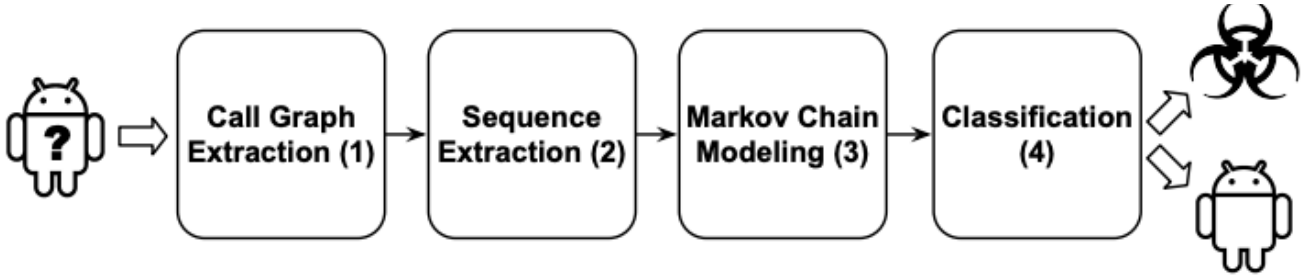
The name of all the packages are normalized by just taking the first name before the point. As example:
System.out. -> System.
This is done to have a manageble number of possible nodes' labels.



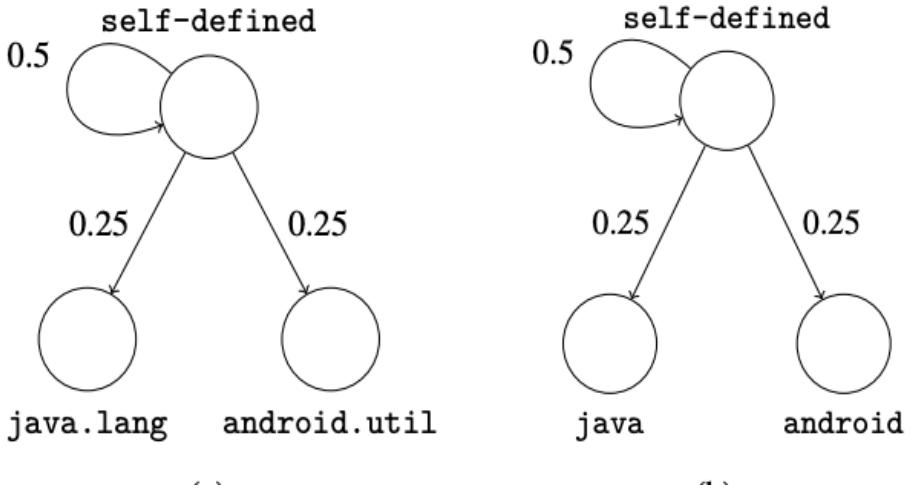
Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



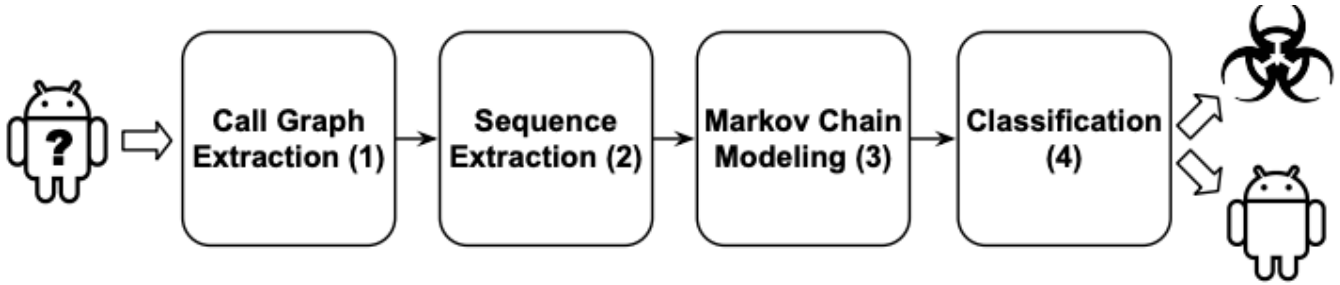
In this way we can have a list of all possible names and represent the graph as a vector. Each component of the vector is a possible edge in the graph.

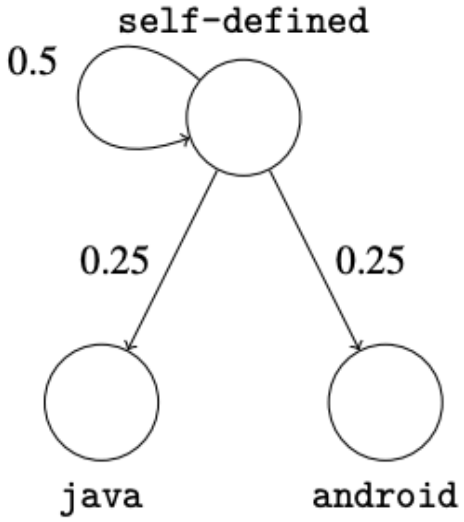


Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

MACHINE LEARNING FOR DETECTING MALWARES: MAMADROID

MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models



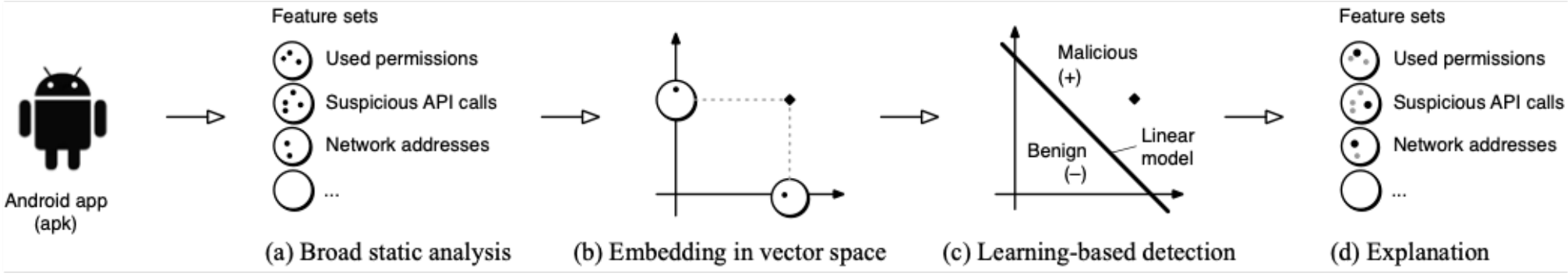


$$(0.5 \quad 0.25 \quad 0.25 \quad 0 \ldots 0)$$

Classifying a vector is easy:
- Random Forest
- SVM
- ...

Images taken from:
[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

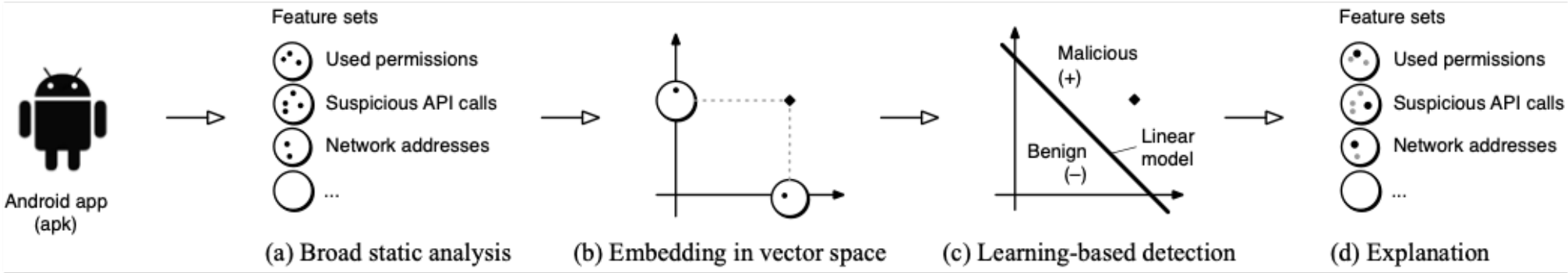
MACHINE LEARNING FOR DETECTING MALWARES: DREBIN



A static analysis extract a set of characteristics that are then embedded into feature vectors:
From manifest: Requested Hardware Components (e.g., camera access), Requested permission, …
Disassembled code: Api Calls, Networking elements,…

[2] Arpi et Al.. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS 2014

MACHINE LEARNING FOR DETECTING MALWARES: DREBIN

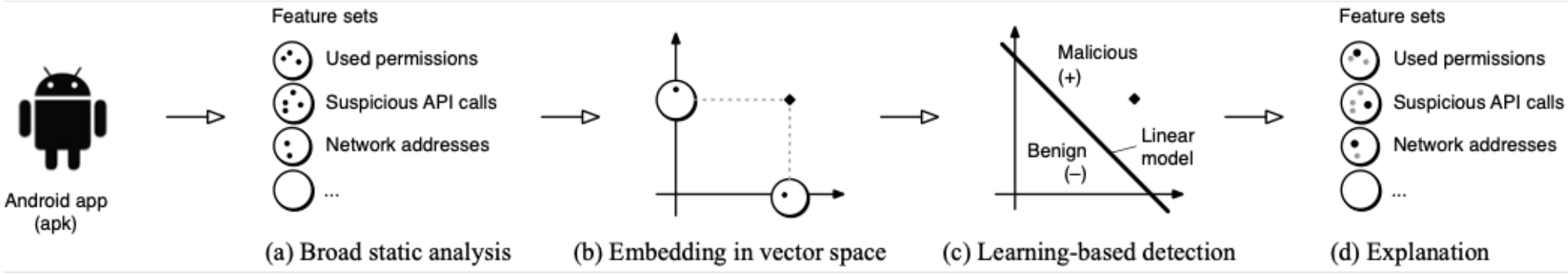


$$\varphi(x) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \\ \cdots \end{pmatrix} \begin{matrix} \cdots \\ \texttt{android.hardware.wifi} \\ \texttt{android.hardware.telephony} \\ \cdots \\ \texttt{SEND\_SMS} \\ \texttt{DELETE\_PACKAGES} \\ \cdots \end{matrix} \begin{matrix} \\ \} S_1 \\ \\ \\ \} S_2 \\ \\ \end{matrix}$$

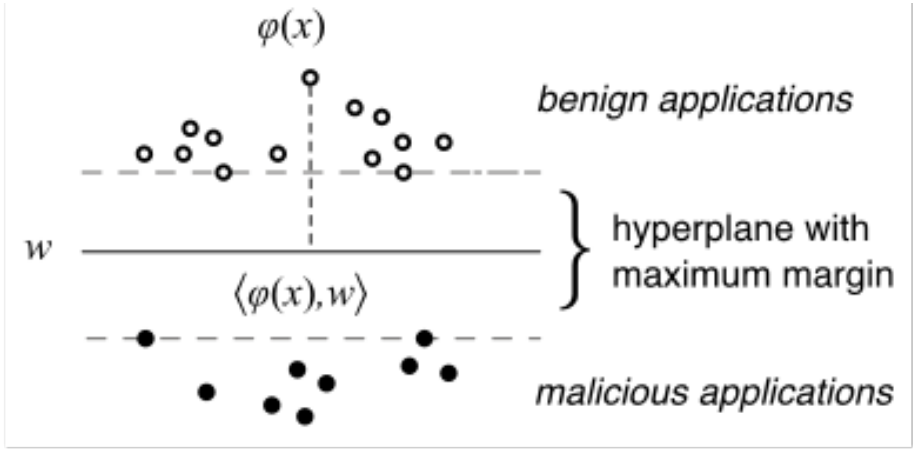
Each characteristic is embedded into a 1-hot vector. The embedding vector of an app is the sum of all these 1-hot vectors.

[2] Arpi et Al.. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS 2014

MACHINE LEARNING FOR DETECTING MALWARES: DREBIN



An SVM classifier is used as classification mechanism.
The model is explainable



[2] Arpi et Al.. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS 2014

SO IS THE PROBLEM SOLVED...

- Mamadroid reaches an F1 score of >0.9 in their experiments.
- Drebin reaches an F1 score of >0.9 in their experiments

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{recall}}$$

SO IS THE PROBLEM SOLVED... OR THERE IS A BIAS?

- In a real scenario a malware detection systems works under these assumptions:
  - (Temporal relationship) You train the system on old malware and goodware and you use it on new malware and goodware
  - (Spatial relationship) You use the system in a scenario where there are a lot of goodware and really few malware.

Experimental setting	Sample dates: Training	Sample dates: Testing	% mw in testing set Ts: 10% (*realistic*) — % mw in training set Tr: 10% — ALG1 [4]	10% — 90%	ALG2 [33]: 10%	90%	90% (*unrealistic*) — % mw in training set Tr: 10% — ALG1 [4]	90%	ALG2 [33]: 10%	90%
10-fold CV	gw: ■■■■■■ mw: ■■■■■■	gw: ■■■■■■ mw: ■■■■■■	**0.91**	0.56	0.83	0.32	0.94	0.98	0.85	**0.97**
Temporally inconsistent	gw: ▒▒▒▒■■ mw: ▒▒▒▒■■	gw: ■■■■▒▒ mw: ■■■■▒▒	0.76	0.42	0.49	0.21	0.86	0.93	0.54	**0.95**
Temporally inconsistent gw/mw windows	gw: ▒▒▒▒■▒ mw: ■■▒▒▒▒	gw: ▒▒▒▒▒■ mw: ▒▒■■▒▒	0.77	0.70	0.65	0.56	0.79	0.94	0.65	**0.93**
Temporally consistent (*realistic*)	gw: ■■▒▒▒▒ mw: ■■▒▒▒▒	gw: ▒▒■■■■ mw: ▒▒■■■■	**0.58**	**0.45**	**0.32**	**0.30**	0.62	0.94	0.33	0.96

The table reports the F1 score
Of two malawre dectetion systems ALG1 (Drebin) and ALG2 (Mamadroid). The F1 score changes if the temporal and
Spatial bias are removed.

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{recall}}$$

Image from: Feargus Pendlebury et al TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. USENIX Security 2019:

THE BIAS....

Experimental set			% mw in testing set Ts: 10% (*realistic*) — % mw in training set Tr				% mw in testing set Ts: 90% (*unrealistic*) — % mw in training set Tr			
							10%	90%	10%	90%
							ALG1 [4]		ALG2 [33]	
10-fold							0.94	0.98	0.85	**0.97**
Temporally inconsis							0.86	0.93	0.54	**0.95**
Temporally inconsis gw/mw windows	mw:	mw:					0.79	0.94	0.65	**0.93**
Temporally consistent (*realistic*)	gw: mw:	gw: mw:	**0.58**	**0.45**	**0.32**	**0.30**	0.62	0.94	0.33	0.96

Manual analiysis is still necessary!

Image from: Feargus Pendlebury et al TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. USENIX Security 2019:

[1] Enrico Mariconti et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. NDSS 2017

WE NEED SECURE SYSTEMS

- Detecting automatically malware is harder than we thought
- Manually analysing malware is hard, so it is unfeasible to manually monitor all the software running in a large system.
- We have to make the system hard to be compromised. There are two main ways that a malware use to compromise a system:
- 1) Humans - This can be fixed with awareness
- **2) Software Bugs - This can be fixed with software engineering.**

DETECTING BUGS 2

- The automated detection of bugs is still at its infancy.
- A manual analysis is still needed.
- How do you manually catch a bug?