Attacks on Android Operating Systems 2

Nir Sasson 212174486
Amir Gillette 324942077

December 2022

1 Introduction

We[1] chose to tackle the classifier presented by McLaughlin et al. [1]. The classifier statically analyses the raw opcode sequences from the disassembled program. The raw opcodes also allow the feature set to be learnt automatically by the network from the textual analysis of the sequences instead of manually choosing the malware features.

2 Preprocessing

The classifier's inputs are the raw opcode sequences, which are analysed by the classifier to define the feature set automatically. The application undergoes a preprocessing stage which consists of two steps - disassembling the application and the extraction of opcodes. The disassembly turns an APK file into Smali files, which can be done by multiple tools, we decided to use APKTool. The second step is the opcode extraction of the Smali files in the disassembly where each line is a sequence of opcodes that represents a method. The Smali instructions are translated into a sequence of raw opcodes which is done by a given opcode table [2]. This table defines a mapping for an operand to its corresponding opcode translation. This flow allow translation of each method to a compatible sequence of opcodes.

3 The Model

Let $O = \{o_1, \ldots, o_n\}$ be the opcode instructions and let $X = \{x_1, \ldots, x_n\}$ be the sequence of opcodes encoded as a vector with D-components, where D is the number of defined opcodes. Our model uses Dalvik dex format for which $D = 224$ [2] but in order to be synchronised with the models paper we used an older version which has $D = 218$. Each vector x_n is a vector of zeros with a '1' in the position corresponding with the n'th integer mapping. Any operands associated with the opcodes were discarded during disassembly and

[1] https://github.com/SassonNir/Deep-Android-Malware-Detection

preprocessing, meaning malware classification is based only on patterns in the sequence of opcodes.

4 Proposed Attack

The model exposes a weak spot by using the raw opcodes in which a malicious APK can modify and tune its disassembled opcodes to avoid detection by the model. We utilise this weak spot by obfuscation. Attacking the raw opcodes allows an attack on generic models which rely on the aforementioned opcodes and allows the APKs to keep their functional state.

There are two general attack vectors regarding the opcodes. Firstly an attack by modification and secondly an attack by addition. The first attack vector changes equivalent sequences of opcodes meaning we preserve the functional behaviour of the app but change interchangeable parts in the code. The second attack vector adds redundancy to the opcodes such as meaningless methods, unused variables and injection of additional methods which have benign context into malicious context and vice versa. This allows changing opcode sequences without changing functionality.

We can perform the first attack vector by changing equivalent instructions such as bit size, 8-bit padding into 16-bit, or in a broader sense changing a method into an equivalent one which achieves the same functionality. This type of attack exploits the model's training and hopes to confuse the model's prediction by introducing a sequence in a different context than seen in the training stage.

The second attack vector uses a pool of predefined redundant code which is injected into the application. The same principle can be achieved by utilising another app context such as a malicious application injecting a non-malicious redundant code from a benign application. Attack by addition exploits weak spots in the textual analysis of the model, this allows for the model to see malicious parts in benign code and vice versa.

References

[1] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail Joon Ahn. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 301–308, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450345231. doi: 10.1145/3029806.3029823. URL `https://doi.org/10.1145/3029806.3029823`.

[2] Dalvik bytecode. `https://source.android.com/docs/core/runtime/dalvik-bytecode`, September 2022. Accessed: 2022-12-20.