Can You See It? -NOP! A Practitioners Study

Diego Soi
Università degli Studi di Cagliari

Leonardo Regano
Università degli Studi di Cagliari

Davide Maiorca
Università degli Studi di Cagliari

Giorgio Giacinto
Università degli Studi di Cagliari

Harel Berger
Georgetown University

Abstract

This study delves into how human intuition detects evasion attacks. Through a suggested online survey with industry and academic practitioners, we will analyze the detection of evasive malware samples by humans, from simple to complex tactics. We wish to emphasize the need for improved training for future cybersecurity experts, to enhance malware detection in human-computer defense systems.

1 Introduction

An evasion attack adds small perturbations to malware samples, such as changing strings values [4, 10] and the flow of the samples [3, 6] to evade detection by malware detection systems. Can the effectiveness of such an attack be gauged by its visibility to human intuition? As mentioned previously by Frederickson et al. [9], there is a tradeoff between the strength of an attack and its detectability by a detection system. In this study, we follow this dilemma in a slightly different form: Can an evasion attack be assessed by its detectability to a human expert? Detection systems are based on human understanding and experience of "what is malware". Therefore, this answer seems straightforward. An evasion attack against malware detection systems is derived from this notion. Generally, adding a non-operational (NOP) command is considered by detection systems as a regular opcode. However, a human expert immediately considers that a red flag, as it does not serve any purpose. Therefore, an attack against a machine that mimics a person, should be affected by human understanding of code. However, it is not so simple. On the one hand, ML/DL may be fooled by attacks that are easily identified by humans. Some models tend to emphasize textual features that can be easily manipulated [4]. An expert can understand the meanings of these textual features based on their names or tags, even if they have not encountered them previously. Even sophisticated models are challenged by the notion of human thought [7, 18], thus attacks against machines are not promised to fool humans. On the other hand, not every attack on ML / DL is visible to the human eye. An attack on face-recognition systems can be achieved, which is invisible to the human eye [20], as it adds infrared perturbation to the picture.

Although this deep understanding of human and computer interaction is of high importance to the community, a limited body of work has been done in this area. Previous work [2] built the fundamentals in this field, using a malware-identification game based on VirusTotal [17] reports. Also, [13, 15] showed different evasive tactics based on the complexity of added code. Therefore, in this study, we pursue a deeper understanding of practitioners of different levels of expertise to identify evasive malware. We devised different tactics for evasion attacks, ranging from a clear addition of useless/dead code to complex entangled loops. We intend to use these attacks to conduct an online study with about 100 practitioners from academia and industry to identify gaps in the human identification of malware samples to express the weaknesses of humanoid malware detection. We hope that the intel from our study will help in educating future security experts in the security community. We also wish to enhance future malware detection models by adding an advanced human view of malware samples.

2 Methodology

Study Design. The analysis focuses on mobile devices, particularly Android apps since it is a widely used OS. The study will be conducted on practitioners with different experience levels in software analysis (i.e., from academic students to industry experts) and with varying studying (working) backgrounds to mitigate the bias related to the studying (working)

USENIX Symposium on Usable Privacy and Security (SOUPS) 2024.
August 11–13, 2024, Philadelphia, PA, United States.

methodology, which may affect the analysis of the data. Participants will be recruited following two criteria: (*i*) students should be from MSc and PhD courses in Computer Engineering who have some knowledge of the topics of this research; (*ii*) experts from academia and companies on the IT fields with a focus on cybersecurity and mobile penetration testing.

The goal of this survey is to answer two research questions:

Q1. What is the correlation between an attack based on NOPs' visibility [15] and the detection rate by an expert?

Q2. What is the correlation between the detection time, practitioner expertise, and success rate?

The survey consists of two questions for each code snippet to accomplish this goal. Initially, a single-choice question is used to determine if the practitioner accurately detects the presence of a NOP (see Appendix A for details). Subsequently, an open-response question is employed to evaluate which part of the code snippet influenced the expert's response to the previous question.

There are two popular languages to generate code snippets from for our case: Smali and Java[1]. The former has been considered because Smali code is the human-readable format for the Android DEX code, and, in general, evasion attacks are made by modifying it. The latter is the main development language for Android apps. There are two primary reasons for selecting Java code. First, the study is interpreted from an *analyst's perspective*, who would most likely look at Java source code using tools like Jadx [14]. Second, using Java snippets allows us to gather a larger number of practitioners whose Smali expertise is limited.

As part of the survey, we will ask for voluntarily disclosing information about the professional demographic and organizational characteristics of the practitioners (e.g., years of experience in software analysis, cybersecurity, and their work), and an introductory section is needed to assess the participants' level of Java knowledge in such a way as to correlate these variables with the results of the survey. Additionally, to assess the correlation between detection time and success rate, the survey has different time slots depending on the complexity of the code snippet. Particularly, participants will have a maximum time limit to analyze and detect, the presence of a NOP.

Data analysis. In conducting the experiments and analyzing the collected data, we will follow the checklist proposed by Wohlin et al. [19]. In particular, we will analyze *quantitatively* in terms of statistics and correlation between data (i.e., visibility vs detection rate, time vs detection rate, expertise vs detection rate). Theoretically, we expect a direct proportion between variables since higher time, expertise, and visibility would increase the detection rate. During the analysis phase, the actual time of response will be gathered and not the maximum slot selected for the snippet. Moreover, a *qualitative* analysis is necessary to study how human experts recognize the presence of an evasive attack. By analyzing the data, we expect a correlation between visibility, time, expertise, and detection rate. Increased visibility should make spotting an NOP easier, while more time and expertise should enhance the ability to identify evasive code.

Threat to validity. Ideally, we aim to employ treatments that evade state-of-the-art anti-malware to reproduce the real scenario accurately. Nevertheless, this can not always be possible since sometimes the time needed to manually evaluate the binary would not be compatible with our setup.

Another potential threat is that some subjects are MSc Computer Engineering students. The discussion on how well students approximate IT experts is active, and no consensus has been reached yet [8, 12]. Nevertheless, significant studies have routinely employed students, e.g., in software engineering [16] and cybersecurity [5, 11], mainly due to the difficulty of involving industrial experts. Following a common practice in such studies, we will test the student's proficiency in Java and correlate it with their results.

This research will adhere to ethical standards by obtaining informed consent, ensuring participant privacy, and maintaining data confidentiality. Participation will be voluntary and anonymous. We will also obtain GDPR [1] consent from participants before starting the survey, and no personally identifiable information (PII) will be collected.

3 Anticipated Contribution

As an expected contribution, we aim to offer insights into the intricate adversarial structures that could be inserted into the source code of Android apps to circumvent ML/DL systems. Specifically, our objective is to offer a practical evaluation of the relationship between evasion attacks and the detection rate by a human analyst.

Moreover, we aim to provide a deeper understanding of the human perception of evasion attacks. This will constitute a first step through a comprehensive evaluation of the differences between human and machine reasoning processes when tasked to evaluate a potential evasive binary malware.

Thus, this contribution has the potential to pave the way for research to improve the detectability of malware evasion attacks by incorporating the "human factor" into the process.

4 Future Work

Moving forward, the primary objective of this preliminary research is to utilize the insights to construct a pipeline in which both automatic approaches, such as LLMs systems, and human analysis are employed to recognize the presence of evasive code structures (i.e., dead code and useless code), which can circumvent learning-based methodologies to identify malicious code. We intend to leverage LLMs, both foundation and fine-tuned code models, such as ChatGPT and Code Copilot, to enhance the automated identification of evasive code.

[1] Kotlin is less supported and spread in the community: https://techaffinity.com/blog/kotlin-vs-java/

References

[1] General data protection regulation (gdpr). https://gdpr-info.eu/, 2016.

[2] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. Humans vs. machines in malware classification. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1145–1162, 2023.

[3] Harel Berger, Amit Dvir, Enrico Mariconti, and Chen Hajaj. Breaking the structure of mamadroid. *Expert Systems with Applications*, 228:120429, 2023.

[4] Harel Berger, Chen Hajaj, Enrico Mariconti, and Amit Dvir. Crystal ball: From innovative attacks to attack effectiveness classifier. *IEEE Access*, 10:1317–1333, 2021.

[5] Mariano Ceccato, Paolo Tonella, Cataldo Basile, Paolo Falcarin, Marco Torchiano, Bart Coppens, and Bjorn De Sutter. Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. *Empirical Software Engineering*, 24(1):240 – 286, 2019.

[6] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

[7] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. A survey of chain of thought reasoning: Advances, frontiers and future. *arXiv preprint arXiv:2309.15402*, 2023.

[8] Robert Feldt, Thomas Zimmermann, Gunnar R. Bergersen, Davide Falessi, Andreas Jedlitschka, Natalia Juristo, Jürgen Münch, Markku Oivo, Per Runeson, Martin Shepperd, Dag I. K. Sjøberg, and Burak Turhan. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empirical Software Engineering*, 23(6):3801 – 3820, 2018.

[9] Christopher Frederickson, Michael Moore, Glenn Dawson, and Robi Polikar. Attack strength vs. detectability dilemma in adversarial machine learning. In *2018 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.

[10] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security*, 51:16–31, 2015.

[11] Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. Empirical validation on the usability of security reports for patching tls misconfigurations: User-and case-studies on actionable mitigations. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 13(1):56 – 86, 2022.

[12] Luka Pavlič, Marjan Heričko, and Tina Beranič. An expert judgment in source code quality research domain—a comparative study between professionals and students. *Applied Sciences (Switzerland)*, 10(20):1 – 13, 2020. Cited by: 0; All Open Access, Gold Open Access.

[13] Sebastian Schrittwieser, Elisabeth Wimmer, Kevin Mallinger, Patrick Kochberger, Caroline Lawitschka, Sebastian Raubitzek, and Edgar R Weippl. Modeling obfuscation stealth through code complexity. In *European Symposium on Research in Computer Security*, pages 392–408. Springer, 2023.

[14] Skylot. Jadx: Dex to java decompiler. https://github.com/skylot/jadx, 2024.

[15] Diego Soi, Davide Maiorca, Giorgio Giacinto, and Harel Berger. Can you see me? on the visibility of nops against android malware detectors. *arXiv preprint arXiv:2312.17356*, 2023.

[16] Toni Taipalus and Hilkka Grahn. Framework for sql error message design: A data-driven approach. *ACM Transactions on Software Engineering and Methodology*, 33(1), 2023. Cited by: 1; All Open Access, Hybrid Gold Open Access.

[17] VirusTotal. Virustotal: Analyze suspicious files and urls to detect types of malware, 2024.

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[19] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[20] Zhe Zhou, Di Tang, Xiaofeng Wang, Weili Han, Xiangyu Liu, and Kehuan Zhang. Invisible mask: Practical attacks on face recognition with infrared. *arXiv preprint arXiv:1803.04683*, 2018.

Appendices

A Code snippets examples

Code snippets employed in the survey are manually implemented by following the same reasonings in the previous work [15] in which some snippets are automatically generated with an adversarial attack.

Listing 1 is an Android Java code with two classes: (*i*) `MainActivity` and (*ii*) `R` whose function `v(int n)` contains a NOP in the form of useless operations. Indeed, the returned value does not depend on the operations before the return statement (i.e., shifts and XORs in lines 18 to 21).

```
1  import android.util.Log;
2
3  public class MainActivity{
4      public static void onCreate(String[]
       args){
5          R rs = new R(3);
6          int r = rs.v(10);
7          Log.i("TAG", "Result: " + n);
8      }
9  }
10
11 public class R{
12     int k;
13     public R(int k){
14         this.k = k;
15     }
16
17     public int v(int n) {
18         int a=n;
19         int b=(a<<this.k);
20         int c=(a^n)^this.k;
21         b = (b^this.k);
22         return (b^this.k)>>this.k;
23     }
24 }
```

Listing 1: A code snippet with useless operations.

Listing 2 shows another kind of NOP. In this case, there is an if that will not be executed. Indeed, the condition is always false (i.e., `c(7,5)=74`), and the code inside the if the branch is dead code.

On the other hand, Listing 3 shows an example of a Java code snippet without NOPs. Indeed, in this case, there are no additional instructions that are not executed or do not change the resulting value.

```
1  public class M {
2      public static void main(String[] args) {
3          int d=8;
4          int a=4;
5          int b=a+2;
6          System.out.println(c(a, b));
7          if(d>2 && c(7,5)==0{
8              int a = 0;
9              for(int i=0;i<5;i++)
10                 a+=i*2;
11             System.out.println("The result
       is " + a);
12         }
13     }
14
15     private static int c(int a, int b) {
16         return a * a + b * b;
17     }
18 }
```

Listing 2: A code snippet with useless operations.

```
1  public class Main{
2      public static void onCreate(String[]
       args) {
3          byte[] b = {44, 112, 119, 108, 113,
       98, 100, 102, 44, 102, 110, 118, 111,
       98, 119, 102, 103, 44, 51};
4          D d = new D();
5          byte[] r = d.d(b);
6          for(int i=0;i<b.length;i++)
7              System.out.println(r[i]);
8      }
9  }
10
11 public class D {
12     public byte[] d(byte[] b) {
13         byte[] r = new byte[b.length];
14         for(int i=0; i<b.length; i++) {
15             r[i] = (byte)(b[i]^3);
16         }
17         return r;
18     }
19 }
```

Listing 3: A code snippet with useless operations.