To appear in an IEEE VizSec 2020

Malware vs Anti-Malware Battle - Gotta Evade ‘em All!

Emily J. Chaffey[*] Daniele Sgandurra[†]

Royal Holloway, University of London

Abstract

The landscape of malware development is ever-changing, creating a constant catch-up contest between the defenders and the adversaries. One of the methodologies that has the potential to pose a significant threat to systems is malware evasion. This is where malware tries to determine whether it is run in a controlled environment, such as a sandbox. Similarly, a malware can also learn how an Anti-Malware System (AMS) decides whether an input program is a malware or in fact benign with the goal of bypassing it. On the other hand, the AMS tries to detect whether a malware sample is performing such evasive checks, e.g. by evaluating the results of Reverse-Turing Test (RTT). This learning process can be viewed as a ‘battle’ between the AMS and the malware, due to the malware attempting to defeat the AMS, where a successful win for the malware would be to evade detection by the AMS and, conversely, a win for the AMS would be to correctly detect the malware and its evasive actions. We propose a visualisation-based system, called *Gotta Evade ‘em All*, that allows cyber-security analysts to clearly see the evasive and anti-evasive actions performed by the malware and the AMS during the battle.

1 Introduction

The amount of new malware samples released every day requires cyber-security analysts to resort to automated processes to analyse malware, relying on the virtue of isolated test environments (sandboxes) [15]. Although this speeds up the analysis of malware, it also introduces additional weakness as the malware may attempt to detect it is being analysed in a sandbox and, if so, it behaves benignly – the so called *sandbox evasion*. This might result in the misclassification of the sample and a potentially harmful program could be allowed to execute. Sandbox evasion can be done via fingerprinting the system via sandbox artefacts (registry keys, loaded libraries, etc.) [14] [8], discrepancy in time among a real and a virtual system [9] and Reverse-Turing Tests [7] [1] [4]. We believe that future malware could use a combination of the previous features and feed them to an embedded ML algorithm to classify the AMS as running on a real host or not [21]. Similarly, the AMS can detect discrepancy among a malware using evasive techniques and a malware not using them [12] [11].

This paper revolves specifically around RTTs, exploring the potential for malware to detect false human interaction by the anti-malware system and respond in a benign manner so that it can avoid being detected. To enable a more visual-oriented representation of the interaction among the malware and the anti-malware system, we propose a system (*Gotta Evade ‘em All*) that displays the ‘battle’ between the machine learning component of the malware-evasion and of the anti-malware program. To this end, *Gotta Evade ‘em All* analyses a malware sample, generates a report and feeds it back to a set of back-end tools to create a visualisation of the battle. If the evasive component of the malware successfully deceives the detection system, it means that it has successfully learnt which features are used by the detection system to determine whether a sample is malicious or not. The status of each battle is visually displayed to the user viewing the program and, once the condition of either the malware successfully evading the AMS, or the AMS detecting the malware, or a set number of battles has occurred, the program terminates by showing the outcome of the battle.

2 Background

The paper [2] discusses an approach based on OpenAI Gym to teach an agent how to bypass an Anti-Malware System (AMS) based on reinforcement learning. The concept of using visualisation (using a breakout-styled game, as seen in Gym-Malware[1]), to demonstrate the ongoing battle occurring in the background between the malware and the anti-malware system stood out as a really interesting method to explain what is occurring to those with less knowledge surrounding malware evasion. The simplicity of the breakout-styled game gave a clear indication of the progress of one side of the battle. However, the game does not provide enough information to the user for them to obtain a solid understanding of what it is trying to explain to them. This is due to the fact that, if a user viewed the

[*]e-mail: Emily.Chaffey.2016@live.rhul.ac.uk
[†]e-mail: daniele.sgandurra@rhul.ac.uk

[1]https://github.com/endgameinc/gym-malware/blob/master/README.md

visualisation aspect of the program in isolation, the user would lack insight into the processes taking place. They could perceive it solely as a game the computer is playing against itself, not comprehending the processes occurring or their complexity.

2.1 RTTs For Sandbox Evasion

The ever-evolving landscape of malware detection means that, in order for a malware to be successful, it needs to be able to evade the detection system. In this context, a RTT has the purpose of using a computer to determine whether an input is made by a *human*, or simulated by another machine – hence, the reverse of a Turing Test [18]. Whilst this can be beneficial to organisations to prevent the receipt of mass automated spam, it can also be exploited by an adversary when designing their malware. By using the theory of RTTs, an adversary can enhance their malware to be able to determine whether it is running in a sandbox or a virtual machine and then act benign if it is suspicious it is being evaluated by an anti-malware system. In the following, we briefly recall some the major different types of RTTs that a malware could use to determine if it is running in a sandbox.

Mouse Pointer Movement. The movement of a mouse cursor can provide important details to a malware evasion system as to whether it is being evaluated in a sandbox. This means that a malware that is tracking mouse movements, in particular their speed, and how algorithmic the movements are, can act benign if it deems that the movements are that of an AMS.

Movement Speed. In a sandbox, simulated mouse movements can be faster than humanly possible [16] due to the algorithm and the speed of the computer. Hence, by calling a method such as `GetCursorPos` repeatedly, at different intervals, interspersed with other instructions, the evasion system can calculate the difference between the mouse position over less than a second. If the difference is above a certain threshold, the system will deem the movement speed too fast to have been made by a human and, therefore, must be simulated behaviour by another computer. The malware evasion system can then take appropriate steps to act benign and avoid detection by the anti-malware system.

Movement Pattern. When a sandbox is emulating the movements of a mouse cursor, it may use an algorithm to generate a path for the cursor to follow. Thus, a malware evasion system could analyse the coordinates of the cursor to determine how ‘human’ it is, as when a user is using a mouse the cursor is not going to perfectly follow an equation. If it seems unrealistic that the pattern is generated by a human, for example too straight, the evasion system would flag its concerns and the system would begin acting benign, as it would deem that it was being monitored by an anti-malware system.

Mouse Clicks. Sometimes a malware will monitor the number of clicks made within a certain period of time. If certain criteria are met, for example there are no mouse movements yet occasional clicking, the malware evasion system may be able to come to the conclusion that a sandbox is simply sending the command for a mouse click event to occur, without a mouse movement. Equally, if there are seemingly too many clicks within the time frame, the system could determine that it was being monitored. With this conclusion, it can decide whether it should continue executing the malware or perform a benign task to avoid detection. Note that malware can also send mouse “click” to the AMS to deactivate its protection [10].

Scrolling Detection. Some malware use scrolling detection to start execution, this is used to perform a RTT as “simulating human interaction with random or preprogrammed mouse movements is not enough to activate its malicious behavior” [17]. In order for the malware to activate, the user would have to scroll to a certain page of a document. This means that if the user never scrolls to that point in the document the malware will never be activated. Whilst this could be viewed as a good technique, it may not be the most effective way of performing a RTT, as it could significantly reduce the number of infected machines and similarly, if the anti-malware system is developed to combat that sort of evasion technique, it may result in an increased number of detections.

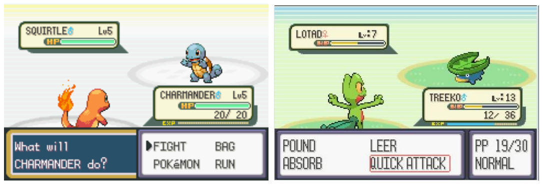
CAPTCHA. A CAPTCHA, for “Completely Automated Public Turing Test To Tell Computers and Humans Apart” [13], is a term to describes a RTT used to automatically determine whether the user is in fact human by producing tests that only a human could pass [19]. Such tests consist of distorted text within an image, where the user would have to determine what is being displayed. A CAPTCHA could aid a malware to evade detection by an AMS as, if the CAPTCHA was used before the execution of the malware, the malware would be able to test whether there is a human using the computer or a computer simulating a user in a virtual machine or sandbox [7]. This would enable the malware to evade detection as, if it determines that the user is not human, it can execute a benign task, so that the AMS does not perceive it as a threat. Additionally, depending on how the malware is programmed, it may result in the AMS being unable to evaluate the threat level of the malware, due to not being able to pass the CAPTCHA [3]. There has also been research into using ML to extract the correct information from the CAPTCHAs to bypass it [5] [20], but there has seemingly been little research in using CAPTCHA solvers in anti-malware systems. This could be very beneficial to the malware-evasion system, as the sandbox may not know how to interact with a CAPTCHA so it could provide a straightforward method for the system to determine whether to act benign or to execute the malware.

3 Gotta Evade ‘em All

We have designed and implemented a game-styled ‘battle’ visualisation system, called *Gotta Evade ‘em All*, where the different ‘moves’ made by the malware evasion and the Anti-Malware System (AMS) are displayed to the user and each move is explained with a short description. We specifically focus on a hypothetical battle among two competing evasion and anti-evasion-based systems, one employed by the AMS to classify a sample as malicious or benign, and the other employed by the malware to classify the host as a sandbox or a ‘real’ system. The main goal of the battle visualisation style is to enable users who are less knowledgeable about malware evasion / detection to understand what is occurring in real-time. There are multiple ‘rounds’ during the battle, where each round is the next ‘move’ the malware performs in an attempt to evade detection, and when the battle ends, the system displays the ‘winner’ of the battle.

3.1 Concept Design

The original design inspiration comes from Pokémon. In fact, although the game may seem a little out of place with regards to malware evasion, from a visualisation perspective, it provides a very clear and simple concept to the viewer as to what is occurring. The use of different option screens, as seen in Figure 1a, allows players to choose whether to: (i) ‘fight’, and are then met with a further option screen to determine which move they wish to use (as shown in Figure 1b), (ii) ‘switch’ Pokémon, which enables them to change the Pokémon they are using during the battle, (iii) ‘access their bag’, so that they can use a special item, such as a potion to heal their Pokémon, or (iv) ‘run’ (which ends the battle).



(a) Pokémon Choice Screen - Pokémon FireRed/LeafGreen Versions

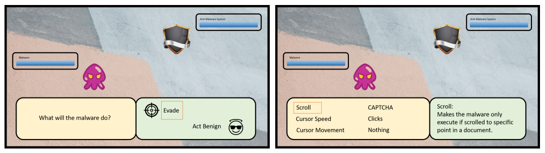
(b) Pokémon Fight Screen - Pokémon Sapphire Version

Figure 1: Examples of Pokémon Screens

3.2 Malware Battle Visualisation

Using the Pokémon ‘fight’ and ‘run’ features as inspiration, *Gotta Evade ‘em All* provides similarly-styled options for displaying the choices made by the malware evasion system using animations. In particular, the ‘fight’ option has been translated to a list of different techniques the malware evasion system could use to evade detection (using each ‘move’ when appropriate). Similarly, if the malware evasion system decides that it is being assessed by an anti-malware system, it could ‘run’, where it would proceed executing a benign process. The same concepts have been used for the other side of the battle (i.e., the anti-malware system). *Gotta Evade ‘em All* provides a more advanced and intuitive way to display to the user what the malware is doing to potentially evade detection and makes the process a lot clearer.

Figures 2a and 2b show an overview of how the visualisation part of the program that displays the malware-evasion system performing an evasion action works. An animation occurs to show the user that the malware has decided to evade (Figure 2a), followed by another animation showing which of the evasion techniques it has chosen (2b) and then the malware ‘performs’ the action (Figure 3) in addition to an animation. On the other hand, if the malware decided to act benign, it would animate the selection of the benign move and perform the relevant animation.



Scroll:
Makes the malware only execute if scrolled to specific point in a document.

(a) Design for Malware Decision Making Screen

(b) Design for Malware Action Choice Screen

Figure 2: Malware Decision Making Screen Designs

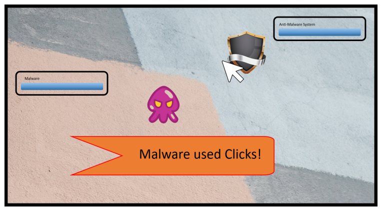
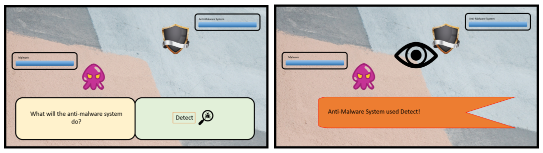


Figure 3: Design for Malware Performing Action Screen

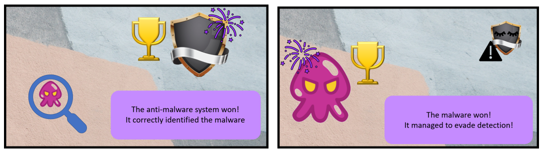
Figures 4a and 4b show a basic plan of how the visualisation part of the program would display the AMS performing a detect action. An animation occurs to show the user that the AMS has decided to detect (Figure 4a) followed by the ‘performance’ of the action (Figure 4b) in an additional animation.


What will the anti-malware system do?
Detect
Anti-Malware System used Detect!

(a) Design for AMS Decision Screen (b) Design of AMS Action Screen

Figure 4: AMS Screen Designs

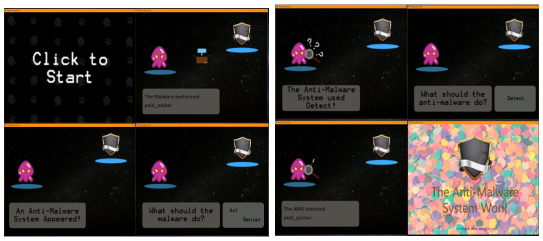
Figures 5a and 5b show the different screens for each final outcome. An animation occurs to show the user that the AMS has won (Figure 5a), where winning consists of the anti-malware determining that the program it has analysed is in fact a malware. Instead, if the malware-evasion system wins, an animation displays to show the user (Figure 5b), where winning is the malware-evasion system successfully evading detection from the AMS.


The anti-malware system won!
It correctly identified the malware
The malware won!
It managed to evade detection!

(a) AMS Winner Screen Design (b) Malware Winner Screen Design

Figure 5: Winner Screen Designs

Gotta Evade 'em All demonstrates to users how an AMS analyses potentially malicious samples. It achieves this in an engaging and familiar manner, by being based upon easily recognisable franchises and having a clean and simple functionality. The ability to run the system without having too much technical knowledge must also be highlighted, as the user only needs to execute one command. Figure 6 shows some screenshots from the start to the end of a real battle.


Click to Start
The Anti-Malware System used Detect!
What should the anti-malware do?
Detect
An Anti-Malware System Appeared!
What should the malware do?
Act
Design
The Anti-Malware System Won!

(a) Start Screen for the Animation and the Encounter View, Followed by the Malware Move Screens (b) AMS Move Screens and the Winner Screen

Figure 6: Screenshots of the Battle Occurring in the Visualisation Program

There is a high degree of automation as the user simply needs to click on a Python script, which starts the virtual environment (an Ubuntu VM) which then automatically opens a Windows 7 VM, takes a snapshot and then closes the VM. This then opens a Cuckoo sandbox [6] and submits the sample executable. Once the analysis has finished and the report has been moved to a shared guest-to-host folder, a parser engine is run and then the visualisation software begins. The code for *Gotta Evade 'em All* and full instructions to run it and use it are available at: https://github.com/EmilyChaffey/GottaEvadeEmAll.

4 Discussion

The *Gotta Evade 'em All* system is a useful tool to educate and explain how an AMS can detect signatures and behaviours of a submitted malware sample. The tool is primarily aimed at security analysts, due to its automation capabilities, however, the visualisation aspects of the tool can be used to inform individuals who have less understanding of the cyber-security aspects. In particular, in an education setting, a knowledgeable instructor could use this as a teaching aid aimed at undergraduate and postgraduate level students to introduce advanced topics such as malware evasion. Similarly, by using a familiar concept based on Pokémon, the proposed system allows the understanding of malware behaviour to be more accessible to college level students. The system can similarly be extended to be used in cyber-security competitions, such as capture-the-flag (CTF) events, to display the status of an ongoing battle, especially in attack-defence CTFs. Finally, this form of malware visualisation can also inspire researchers to address the identified evasive issues using novel and creative anti-malware techniques that in standard settings (e.g., based on program analysis) would be difficult to come up with.

5 Concluding Remarks

This paper has a unique take on a complex and ever-changing subject area, which can provide individuals with the foundations of an understanding of how an Anti-Malware System (AMS) functions, as well as highlighting the potential areas an adversary could exploit. The *Gotta Evade 'em All* system provides security analysts with a tool to aid research, due to its automation capabilities for submitting a malware sample for evasion analysis. Additionally, the system makes the concepts of malware and AMS more accessible to those with less knowledge in those areas and acts as an educational tool which displays the steps taken for analysis and evasion in an abstracted and recognisable format of a battle. It also uses simple-to-understand concepts to enable users to conclude whether evasion was successful or not. Future extensions would be aimed at integrating machine learnings algorithms into *Gotta Evade 'em All* system to improve the analysis and classification of malware based on evasive features.

Credits

There are a few images that were externally sourced, and the credits are: (i) Figure 1a source: Pokémon FireRed and LeafGreen Versions; (ii) Figure 1b source: Pokemon Sapphire; (iii) Malware vector - macrovector; (iv) Confetti background - Ylanite Koppens; (v) Shield vector - freepik; (vi) Starry Sky background - Francesco Ungaro.

Acknowledgements

Daniele Sgandurra's work was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 779391 (FutureTPM).

References

[1] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Comput. Surv.*, 52(6), Nov. 2019. doi: 10.1145/3365001

[2] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, January 2018.

[3] R. Bhatia. How captcha is being used to bypass anti malware security scans and firewalls. *https://www.securitynewspaper.com/2019/09/12/how-captcha-is-being-used-to-bypass-anti-malware-security-scans-and-firewalls/*, 09 2019. URL Access Date: 21/10/2019.

[4] A. Bulazel and B. Yener. A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In *Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium*, ROOTS. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3150376.3150378

[5] E. Bursztein, J. Aigrain, A. Moscicki, and J. Mitchell. The end is nigh: Generic solving of text-based captchas. In *The End is Nigh: Generic Solving of Text-based CAPTCHAs*, 08 2014.

[6] C. T. Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, Mark Schloesser, Koen Houtman, Ricardo van Zutphen, Ben de Graaff. Cuckoo sandbox. *https://cuckoosandbox.org/*. URL Access Date: 01/07/2020.

[7] B. Dolan-gavitt and Y. Nadji. See no evil: Evasions in honeymonkey systems. Technical report, Technical Report, May 2010.

[8] O. Ferrand. How to detect the cuckoo sandbox and to strengthen it? *J. Comput. Virol. Hacking Tech.*, 11(1):51–58, 2015. doi: 10.1007/s11416-014-0224-9

[9] T. Garfinkel and M. Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, p. 20. USENIX Association, USA, 2005.

[10] Z. A. Genç, G. Lenzini, and D. Sgandurra. A game of "cut and mouse": Bypassing antivirus by simulating user inputs. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, p. 456–465. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3359789.3359844

[11] D. Kirat, G. Vigna, and C. Kruegel. Barebox: Efficient malware analysis on bare-metal. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, p. 403–412. Association for Computing Machinery, New York, NY, USA, 2011. doi: 10.1145/2076732.2076790

[12] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti. Detecting environment-sensitive malware. In R. Sommer, D. Balzarotti, and G. Maier, eds., *Recent Advances in Intrusion Detection*, pp. 338–357. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[13] N. J. H. Luis von Ahn, Manuel Blum and J. Langford. The captcha web page: http://www.captcha.net. URL Access Date: 01/07/2020.

[14] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 1009–1024, 2017.

[15] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *CoRR*, abs/1609.03020, 2016.

[16] A. Singh and Z. Bu. Hot knives through butter - evading file-based sandboxes. *https://www.fireeye.com/content/dam/fireeye-www/current-threats/pdfs/pf/file/fireeye-hot-knives-through-butter.pdf*, 2014. URL Access Date: 01/07/2020.

[17] A. Singh and S. O. Vashisht. Threat research - turing test in reverse: New sandbox-evasion techniques seek human interaction. *https://www.fireeye.com/blog/threat-research/2014/06/turing-test-in-reverse-new-sandbox-evasion-techniques-seek-human-interaction.html*, June 2014. URL Access Date: 01/07/2020.

[18] A. M. Turing. I. - computing machinery and intelligence. *Mind*, LIX(236):433–460, 10 1950. doi: 10.1093/mind/LIX.236.433

[19] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *EUROCRYPT*, 2003.

[20] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 332–348. ACM, New York, NY, USA, 2018. doi: 10.1145/3243734.3243754

[21] A. Yokoyama, K. Ishii, R. Tanabe, Y. Papa, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, M. Brengel, M. Backes, and C. Rossow. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, eds., *Research in Attacks, Intrusions, and Defenses*, pp. 165–187. Springer International Publishing, Cham, 2016.