November 09, 2018

Detecting malware based on mismatch between user interface and computation load

Victor Cărbune

Alexandru Damian

Detecting malware based on mismatch between user interface and computation load

ABSTRACT

Techniques to detect malicious code running underlying a user interface are described. The software application is detected as potential malware if a mismatch is detected between the interface presented to the user and computations performed by the application that presents the user interface. A trained machine learning model is applied for such detection. With user permission, a sequence of rendered images that represent the user interface and a sequence of execution traces sampled from computational operations performed by the application that presents the interface are provided as inputs the model. The model outputs a score indicative of appropriateness of the amount of computation for the user interface.

KEYWORDS

- Malware detection
- Crypto mining
- Computation load
- Interaction sequence
- Execution trace

BACKGROUND

Various forms of malware, e.g., viruses, spyware, crypto mining malware, etc. attempt to utilize computational resources of a user device without user knowledge or permission. For example, malicious actors may surreptitiously run code for certain computationally intensive operations such as cryptocurrency mining, dictionary attacks on user passwords and other cryptanalytic operations, etc. on unsuspecting user devices, while the user is engaged in other activity such as viewing a webpage. For example, a user may spend significant time reading a news website or blog, which allows malware on such websites to deploy code on the user's device for malicious operations. Malware may also be utilized to coordinate network traffic from a large number of user devices, e.g., to launch a distributed denial of service (DDoS) attack.

While some users may detect such malware, e.g., by observing excessive computational load, by use of antimalware tools that detect particular signatures of malicious code, etc., it is difficult for many users to detect and prevent such use of their devices. Some current antimalware tools detect malware by matching computation patterns with known malware patterns.

DESCRIPTION

Techniques described herein detect a mismatch between a user interface and computation load on a user device by applying a machine learning model. The techniques are implemented only if users provide permission to access data on rendered user interface and corresponding computational load. Users are provided with options to deny such permission, provide permission selectively, and to change the permission at any time. For example, permission may be obtained at an initial setup stage (e.g., of a user device or account), and is modifiable by the user.

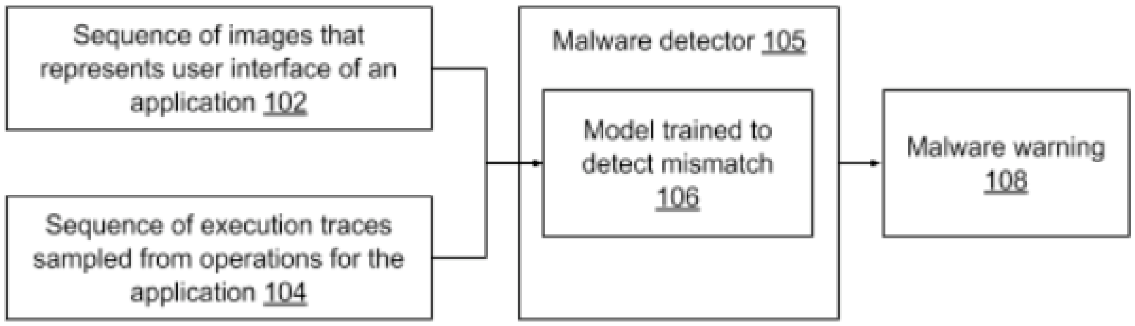
Sequence of images that represents user interface of an application 102
Sequence of execution traces sampled from operations for the application 104
Malware detector 105
Model trained to detect mismatch 106
Malware warning 108

Fig. 1 Malware detection by comparison of UI and execution traces

Fig. 1 illustrates use of a malware detector (105) that utilizes a model trained to detect mismatches (106) between user interface and code execution. The model computes a score based on the determined match of the ongoing computation with the user interface that is presented to a user. When the user permits, a sequence of images that represent the user interface (102) and a sequence of execution traces sampled from operations performed by the application that generates the interface (104) are provided as inputs to the ML model. For example, execution traces may be sampled periodically, e.g., over one to three seconds.

Based on the processing of the pair of sequences, it is determined if there is a mismatch between the information presented on the interface and the computation that supports the interface (106). For example, if the user interface is a simple webpage that waits for user input, e.g., navigation input for scrolling the page, a mismatch is detected if greater than a threshold amount of computation is performed, e.g., greater than typical computational load to render the page itself. The model outputs a score, e.g., between 0 and 1, based on the provided inputs.

If the score indicates a mismatch, various mitigating actions can be taken. For example, a malware warning can be displayed (108) on the user interface. When the underlying computation is characterized, the message can include further details, e.g., “the computational requirements for this application are indicative of unauthorized use of your device for crypto mining.”

The machine learning model can be implemented with common neural network modules as building blocks, for example, convolutional neural networks (CNNs) for processing images of rendered user interfaces, and recurrent neural networks (RNNs) for processing variable-length sequence inputs. To process the execution traces, an embedding layer for different instructions that are present in the execution traces can be used, which is dependent on the position of the model in the software stack. The sequences of execution traces can be provided by the application, a web browser that displays a webpage, when the model is implemented within the browser or as a browser add-on. The model can also be included in the operating system of the user device. When the model is built into the browser, the browser is instrumented to capture execution traces and provide those to the model. If malware detection is provided as a service, the application developer may provide an instrumented version of the code to permit capture of execution traces.

The machine learning model can be trained, e.g., by using synthetically generated cases that combine simple user interfaces with heavy computation loads, on known malicious websites and applications, and by using human-annotated data if available. For example, the training data can comprise pairs of screen-images and execution traces of corresponding under the hood computations.

The techniques described herein utilize the execution trace which is a precise measure of malicious computations. For example, an execution trace may show a signature of a large number of matrix multiplications, or calls to a graphical processing unit (GPU), when the rendered user interface images indicate that no image was displayed that would require such computations. The use of the execution trace makes the techniques described herein advantageous over conventional approaches that use typical load measurements, e.g., CPU load, memory usage, etc. to identify malware, since such measurements may not be granular enough to differentiate between malicious and benign code.

The described techniques are advantageous due to the ability to detect previously unknown malicious code for which the signature is not known. Further, the techniques may also find alternative applications, e.g., to determine efficiency of software code vs. benchmark code that provides similar user interface.

In situations in which certain implementations discussed herein may collect or use personal information about users (e.g., user data, information about a user’s social network, user's location and time at the location, user's biometric information, user's activities and demographic information), users are provided with one or more opportunities to control whether information is collected, whether the personal information is stored, whether the personal information is used, and how the information is collected about the user, stored and used. That is, the systems and methods discussed herein collect, store and/or use user personal information specifically upon receiving explicit authorization from the relevant users to do so.

For example, a user is provided with control over whether programs or features collect user information about that particular user or other users relevant to the program or feature. Each user for which personal information is to be collected is presented with one or more options to allow control over the information collection relevant to that user, to provide permission or authorization as to whether the information is collected and as to which portions of the information are to be collected. For example, users can be provided with one or more such control options over a communication network. In addition, certain data may be treated in one or more ways before it is stored or used so that personally identifiable information is removed. As one example, a user’s identity may be treated so that no personally identifiable information can be determined. As another example, a user’s geographic location may be generalized to a larger region so that the user's particular location cannot be determined.

CONCLUSION

Techniques to detect malicious code running underlying a user interface are described. The software application is detected as potential malware if a mismatch is detected between the interface presented to the user and computations performed by the application that presents the user interface. A trained machine learning model is applied for such detection. With user permission, a sequence of rendered images that represent the user interface and a sequence of execution traces sampled from computational operations performed by the application that presents the interface are provided as inputs the model. The model outputs a score indicative of appropriateness of the amount of computation for the user interface.