Cross-Platform Malware Contamination[1]

Limiting Malware Propagation - A Way Forward

Authors
Nicholas Aquilina, MSc (Royal Holloway, 2014)
Konstantinos Markantonakis, ISG, Royal Holloway

1. Abstract

The use of mobile devices in our everyday lives has become widespread. In recent years, the number of smartphones sold globally surpassed the number of desktop computers sold. Malware authors have followed this and developed sophisticated malware addressed both at the mobile device platforms as well as desktop platforms. Particular interest is being shown in the areas of Microsoft Windows operating system for desktop computers and the Android operating system for mobile environments. This article provides a short introduction to the subject of malware and various malware concealment and detection strategies. With this theoretical framework in mind, we put forward a suggestion for the creation and implementation of a hypervisor at a layer below the operating system utilising a behavioural analysis framework. This article finishes off with a discussion of the challenges and barriers for its practical implementation.

2. Introduction

Malware is a general term used to refer to any software that is installed on a machine and performs unwanted tasks. Malware became known to many computer users through widespread infections at the turn of the third millennium. These were based on email attachments as the primary infection vector.

The year 2004 saw the release of the first malicious software, Cabir, aimed at smartphones. This malware spread amongst mobile devices which had Bluetooth enabled in discoverable mode and exploited the limited resources of mobile devices, at that time, battery life.

Infection vectors for malware changed over the years, moving from the traditional desktop platform over to mobile devices. Viruses initially spread through the use of infected floppy discs. When Internet connectivity became more ubiquitous, malware managed to spread using techniques such as mass email lists or web vulnerabilities. Infection vectors which malware uses for mobile devices also had a similar effect, moving from simple SMS or MMS infection vectors to Bluetooth, email and web vulnerabilities.

[1] This article is to be published online by Computer Weekly as part of the 2015 Royal Holloway info security thesis series. The full MSc thesis is published on the ISG's website.

3. Malware Concealment Strategies

Malware concealment strategies serve one purpose: the survival of malicious code. The longer malware can protect itself from detection, the more time it has for replication and infection. In this section, we will discuss the malware lifecycle as well as various malware concealment strategies.

3.1 The Malware Lifecycle

The malware lifecycle is made up of four phases, as shown in figure 1 below:

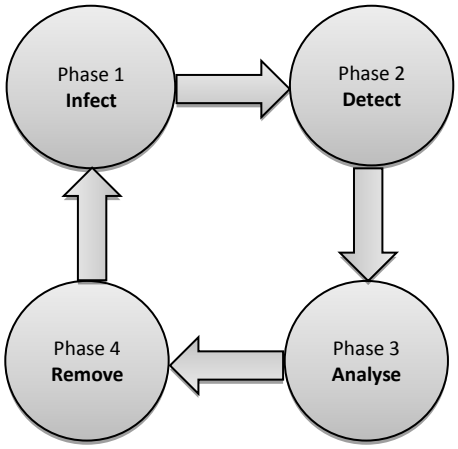

Figure 1 - Malware lifecycle stages

Concealment strategies aim to increase the span of time between the infection and detection phases. In addition, these strategies aim to make analysis of the malware difficult for anti-malware developers. Through concealment strategies, malware authors aim to spread and prosper, hiding their code from plain sight.

3.2 Types of Malware Concealment Strategies

Malware concealment strategies can be divided mainly into two groups: malware which uses a set of techniques to hide itself from detection, such as bypassing anti-virus or anti-malware signature detection, and malware which also actively try to hinder the detection and analysis of the code.

Malware concealment:

- Passively hiding from detection
- Actively hinder detection and analysis

The first group uses passive strategies to evade detection, so that the malware will not attempt to defend itself from the concealment strategies already implemented within the application code. We can view this kind of malware as having a sort of protection shell which, once detected and removed, will reveal the source code behind. The second group uses active strategies to evade detection. In addition to passive concealment strategies, this type of malware will implement aggressive coding techniques for self-protection. Common examples include hindering the detection of a virus using signature-based methods and hindering the analysis of the code. Some of these techniques include:

- **Anti-emulation.** The use of virtual machines is an important aspect of malware analysis across all platforms. Through the use of emulation, a safe and accurate environment is created for

analysing and evaluating real life behaviour of malware and its impact on the targeted system. However, the use of emulation systems can be detected by simple queries directed to the machine. Malware creators have responded by implementing anti-emulation techniques to counter analysis and evade detection.

- **Anti-debugging**. Anti-debugging techniques are used to trick debuggers, making the job of anti-malware analysis much more difficult. A common technique used is code obfuscation, where programme code is deliberately modified to make it difficult for humans to understand or for anti-malware engines to detect. Other techniques deal directly with how the host operating system handles debugging.

- **Anti-disassembly**. Anti-disassembly techniques employ a special code within a file that trick the disassembly and analysis tools into producing incorrect programme listings. These techniques provide an additional layer to the extensive array of armour available for malware creators.

4. Malware Detection Strategies

Malware authors are becoming more aware of the security methods devised, and used, in commercial as well as domestic realms. They have become more skilled in hiding their malicious programmes and operate quietly. These attacks have become so advanced that sometimes they can go undetected for weeks, or even months, as in the case of StuxNet. Some malware detection strategies currently in use are:

- **Static analysis** is a process of extracting static information from a file without actually executing the file. This information is used to create a profile of the file using different techniques such as calculating file hashes, scanning through different anti-virus and anti-malware engines and extracting file information.

- **Static taint analysis** is used for detecting the information flow of a set of instructions that are influenced by the user's input. The basic idea is to identify and label variables that have been 'tainted' with input controlled by the user. Any operation that uses a value from a tainted object to derive another value for another object will taint that object. Simple checks that are done on these variables could indicate possible attacks, such as cross-site scripting, SQL injection and malicious script injection.

- **Dynamic analysis**, or **behavioural analysis**, is the process of analysing the actions of a programme in the course of execution. The main idea is to execute a code sample within a controlled environment (such as a virtual machine), monitoring its behaviour and obtaining further information about its nature and purpose. Through this analysis, the researcher will be able to assess better the threats, and create proper countermeasures.

- **Dynamic taint analysis** is an extension of the static technique described above. It was originally published by Newsome and Song of Carnegie Mellon University in 2005. Data originating or derived from untrusted sources, such as the network, are labelled as tainted. This technique tracks in real time how labelled data impacts other data in a way that might leak the original

sensitive data. The type of tracking, as originally proposed, was performed at the instruction level. Finally, the impacted data is identified before leaving the system, usually at the network interface level.

- **Heuristic analysis** combines a few known facts with experience to make an assumption on the classification at large, and is regarded as part of artificial intelligence. The term 'heuristic' refers to the act of discovering the solution to a problem. These analysis and detection mechanisms employ data mining and machine learning techniques to review, trace and analyse the behaviour of the application code. Through the use of these methods, heuristics look for pieces of programme code that seem to look like a virus, rather than looking for specific virus signatures.

- **Hybrid malware analysis** is a new technique combining both static code analysis as well as dynamic code analysis. This technique combines the benefits of static code analysis with virtual machine analysis.

Detection involves a process of analysing and identifying whether a code is genuinely benign or malicious. Robust malware detection strategies depend on how efficiently obfuscated malware is detected. Simple strategies, such as trusting and installing only digitally signed applications, are one way of limiting malware infection. However, given the vast number of applications that are available on the Internet, especially through the use of peer-to-peer sites, one cannot expect all applications to be digitally signed.

5. Framework Proposal to Limit Propagation

The malware detection strategies reviewed in the previous section are not native to a specific platform. However, they all share a common trait: analysis and detection through anti-malware engines is carried out from within the operating system itself, running as a process. They usually reside at a high level within the operating system stack. Attackers are well aware of this fact and have put in place various active and passive concealment strategies. This makes them as vulnerable as any other application running on the device. If malware manages to subvert the operating system, it will be able to replace various drivers from within the kernel, such as the modem driver. This could have serious repercussions on the user, who risks having to pay excessive fees for unwanted calls or messages.

5.1 A New Way to Look at Malware Detection

A new way in which to look at malware detection is to devise and add a new security level altogether, moving away from the traditional approach of existing detection techniques. The primary aim of this novel model is to protect the device, be it Windows-based or Android-based, with a focused view on limiting malware propagation across these two platforms.

We propose the addition of a new layer between the lowest layer, that is the hardware layer, and the next layer, which is the operating system kernel layer. This additional security layer, which we will denote as our *hypervisor*, can protect and monitor the whole system. It has to be lightweight in terms of consuming system resources, compatible with multiple platforms and hidden to malware that adopts strategies to evade detection.

This hypervisor requires a malware detection technique in order to detect the presence of malicious intentions during programme execution. In our view, the most appropriate and effective malware detection technique to implement on our model is behavioural analysis. Malware can easily evade static malware analysis using polymorphism or metamorphism techniques. In addition, such detection systems are largely based on signatures. The connectivity of the device will affect the frequency with which this signature database is updated, which in turn will affect the effectiveness of the detection system.

Our proposals are based on the assumption that the operating system is running an anti-virus or anti-malware application. In figure 2 below, we collate the various components of Windows-based PC and Android-based devices, noting the similarities, and simplifying them in the following architecture:

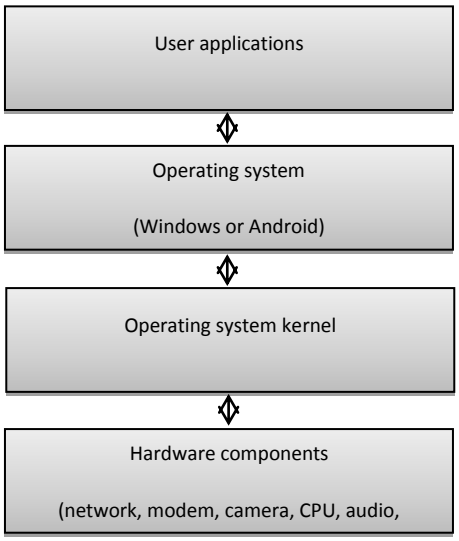

Figure 2 - Common architecture for Windows and Android devices

5.2 The Need for a Hypervisor

We need to devise this additional layer in such a way that malicious behaviour can be monitored, intercepted and acted upon. We also need to be able to define what malicious behaviour is. Additionally, this layer must be able to intercept any hardware calls that may harm the operating system or cause distress for the user. This may include accessing the camera, increasing CPU utilisation thereby consuming battery use, accessing protected memory areas or enabling the computer or mobile microphone. It becomes evident that the deployment of this hypervisor layer must be transparent to the operating system. Our proposed architectural layout for this additional layer is as follows (see figure 3 below):

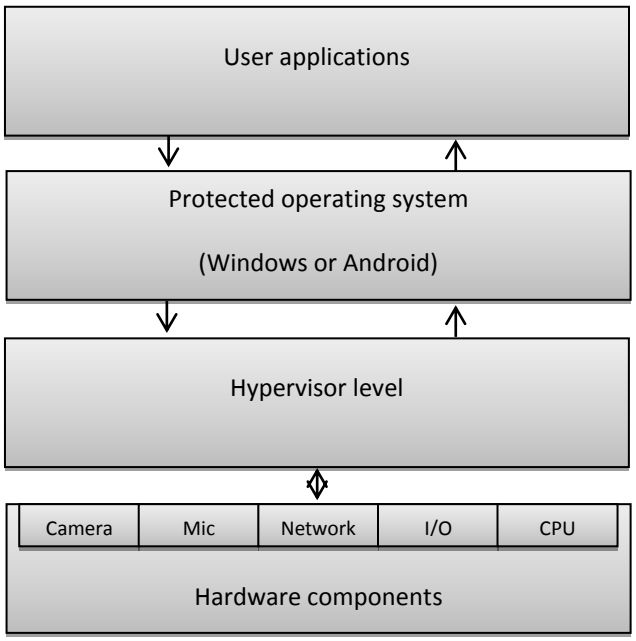

Figure 3 - Proposed architectural layout with the hypervisor

Within the hypervisor layer, we configure two specification sets. One set, S1, will contain the hardware features that will be monitored, such as the GPS sensor, Bluetooth, WiFi, microphone, camera and accelerometer. The other set, S2, will contain the permissions that will be requested for access to S1. Examples of permissions include a request to access the current roll/pitch of the accelerometer, take a picture using the camera, enable Bluetooth connectivity or access the location coordinates from the GPS. These sets will then be validated against user activity/inactivity parameters. Therefore, in our scenario, if the hypervisor detects a request to access the camera (selected from S1) requesting permission to take a snapshot (selected from S2), the request will be allowed if a previous user activity took place, such as using the touchpad or touchscreen. Such activity would be easy for the hypervisor to detect as this is the layer controlling access to the hardware. Below is a graphical overview of this activity.

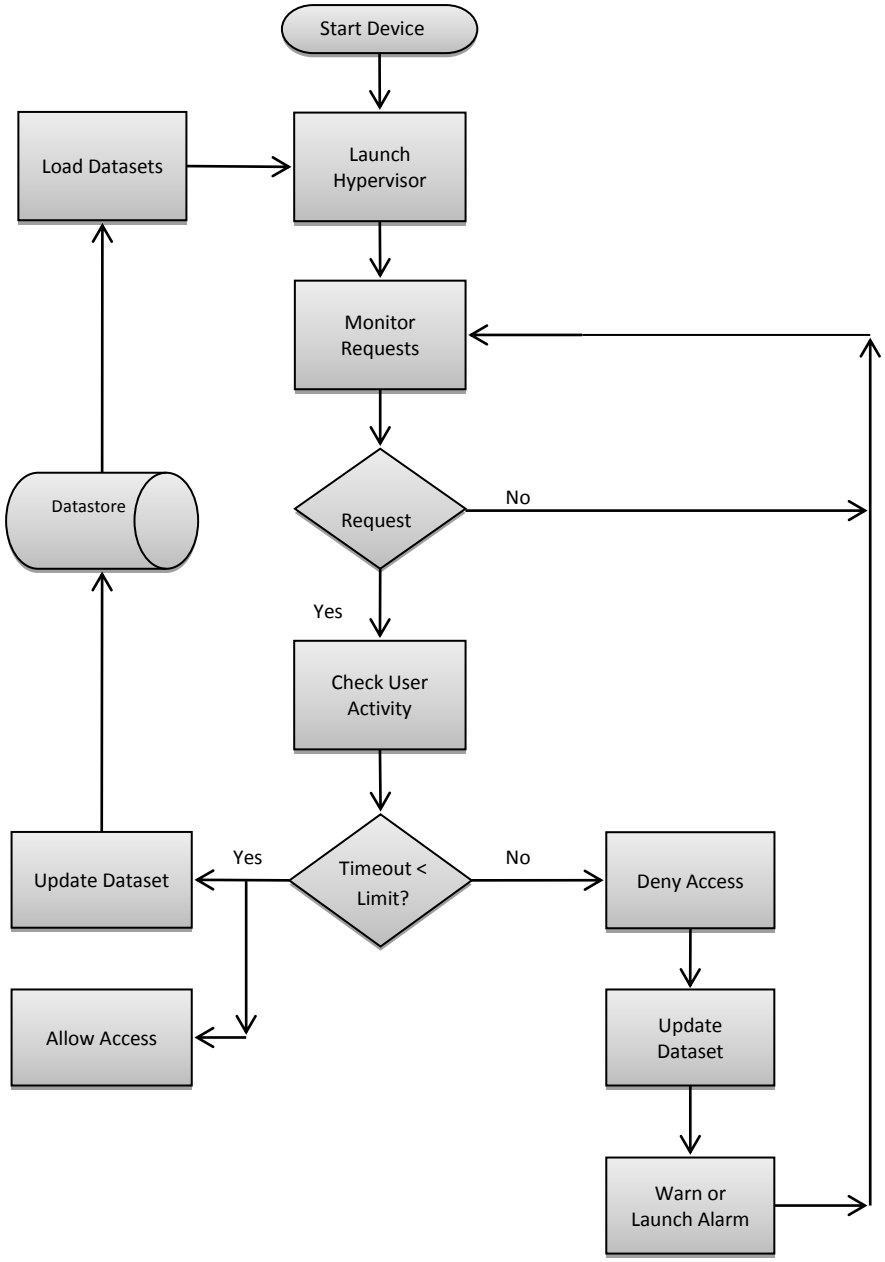

Figure 4 - Flowchart of behaviour analysis

6. Challenges to Implementation

Creating this proposed new layer will pose various challenges, and present barriers for its practical implementation. In the following sections, we note the four major challenges to its implementation.

6.1 Access to hardware drivers

Having a hypervisor interact directly with the hardware layer brings about the major challenge of having access to the various hardware device drivers. Whilst on the Windows platform, the number of hardware manufacturers is well known, the same does not hold for the Android platform. Android is not controlled by a single company, as is the case for iOS installed on Apple devices. It will be a challenge to get device driver source code directly from chip manufacturers. Possible solutions would be to call for a joint effort with a major player in this area, such as a CPU manufacturer.

6.2 Power management

Battery life is a critical factor on mobile devices. With the ever increasing performance in CPU speeds and memory access speeds, battery life is a limiting factor. We do not want our hypervisor to constantly check whether a request is forthcoming or not. Having such behaviour, constantly polling for requests, will hamper battery life. Google implements 'wakelocks' in its Android operating system. Wakelocks are a kernel mechanism which Android uses for power management. When a process or thread holds a wakelock, the kernel will refrain from entering a low-power state. If our hypervisor is not aware when a wakelock is active or not, it will remain in a constantly polling state, thereby contributing to battery consumption. Our hypervisor should be aware of these wakelocks in the layers above.

6.3 Functionality

From a security perspective, functionality has always been a critical factor. Mobile and computer users have high expectations of their device's functionality. They may regard the added benefits of security as a burden, limiting what they can do with their devices. The addition of a hypervisor layer will affect functionality. There are various factors that need to be analysed in more detail, such as processing when an incoming call takes place. What if a malware manages to send an SMS message to a premium rate number and we receive a call back to confirm or accept this? If malware manages to subvert our operating system, it will be able to replace various drivers from within the kernel, such as the modem driver. Thus, any requests such as SMS messages or phone calls could be filtered as required and hidden from the end user. On the other hand, having this additional hypervisor layer will create an additional step for attackers to subvert and control.

6.4 Security review

Security through obscurity is the belief that a system can be secure as long as nobody outside the group that implemented it is allowed to know anything about its internal mechanisms. In 1883, Auguste Kerckhoffs argued that any secure military system must not require secrecy and can be stolen by the enemy without causing trouble. In the academic security community Kerckhoffs' Principle is widely supported. Our hypervisor layer cannot be software with an unknown internal structure (commonly referred to as black-box). On the contrary, it has to be published in such a way that its internal structure and design are known and can be tested by the security community in general. This allows independent assessment of its exposure to risk, makes patching bugs easier and forces software developers to spend more effort on the quality of the code.

7. Conclusion

Despite these challenges, we believe that this proposal brings forth a useful and efficient way to monitor and protect our system. Based on hardware virtualisation, it should provide security against various cross-platform malware in existence. Any practical development of this proposal should be done with specific objectives in mind, namely a hypervisor which is lightweight in operation, transparent to the operating system and provide for added capability enhancements. The weaknesses affecting current malware detection strategies lie at the basis of our departure from the traditional view of how we can protect devices in favour of new methods designed to amplify the effectiveness of existing detection techniques.

Biographies

Nicholas Aquilina is currently responsible for devising and managing corporate systems at IT Services, University of Malta. His role involves supporting University staff in the Finance, Library and Student Registration departments. He is also a key contributor within the Information Security team, responsible for various aspects of user and system security. Prior to joining the University of Malta, Nicholas worked in both private and public sectors for over 15 years. During this time, Nicholas gained hands-on experience on important aspects of Information Security, working for an ISP, online betting company and the State postal operator, where he held a managerial post for 8 years before joining the University of Malta. Nicholas holds an Executive MBA with special focus on Electronic Business from the University of Malta, and is a 2014 MSc graduand in Information Security from Royal Holloway. Nicholas's profile can be found at www.nicholasaquilina.com and he can be contacted via info@nicholasaquilina.com.

Dr Konstantinos Markantonakis M.Sc., Ph.D., MBA is currently a Reader in the Information Security Group. His main research interests include smart card security and applications; secure cryptographic protocol design and analysis, Public Key Infrastructures (PKI), key management, mobile phone security, embedded system security. Since completing his PhD, he has worked as an independent consultant in a number of information security and smart card related projects. He has worked as a Multi-application smart card Manager for VISA EU, responsible for multi-application smart card technology for southern Europe. Following from the he worked for Steer Davies Gleave**,** responsible for advising transport operators and financial institutions on the use of smart card technology. He continues to act as a consultant on a variety of topics including smart card security, key management, information security protocols, mobile devices, smart card migration program planning/project management for financial institutions, transport operators, technology integrators.