Designing flexible sandboxing solutions to adapt to new malware trends

Matteo Cafasso and Mathieu Tarral, F-Secure

Every day, security organisations analyse thousands of new files and URLs, identifying the harmful ones to constantly improve their knowledge of computer threats. It is hard to guess whether a file or URL could be harmful or not without executing it, and executing unknown malicious software is dangerous.

Unknown threats

When dealing with unknown threats, such as malicious files or web URLs, one of the best analysis techniques is to execute them in order to study their behaviour. This approach does not come without risks. A typical mitigation technique will see the objects being executed as a test within a sandbox.

Sandboxes are restricted environments carefully designed for safely running dangerous programs and recording the changes that they produce. Their implementation may vary according to the specific use case and the use of hardware assisted virtualisation is a pretty common example. There are many advantages to virtualisation: it’s a mature and stable technology; it allows quick provision of a vast multitude of environments; it scales very well; and it provides a good degree of isolation when running dangerous software. Other examples of sandbox implementation are emulators and containers.

IT security researchers and analysts have been using sandboxes for several years in order to study the behaviour of interesting samples. Many IT security companies rely on automated sandboxes to analyse large amounts of files and web URLs. Despite their popularity among IT security specialists though, commercialising sandboxes seems to be slow to happen.

One of the main reasons for this appears to be the high maintenance cost of this type of solution. Behavioural analysis sandboxes are required to execute a large variety of programs in order to identify the malicious ones. Considering the number of different devices, operating systems, executables and media types that are available today, this is not an easy task.1

Another challenging aspect to consider is the evolutionary growth of malware, which forces forensic experts to constantly perfect their feature extraction tools. Such tools are required to produce the data that provides a verdict on the nature of a sample. These tools must be quickly integrated within the sandboxes themselves without affecting their operational status.

It seems clear that there is a need to reduce the maintenance and operational costs of such solutions. This article explores design patterns and technologies suitable for this kind of problem, aiming to help the security industry bring sandbox technologies closer to their customers. We’ll look at an architectural design aiming to produce a flexible and maintainable sandboxing platform. Then we’ll discuss the introduction of a software development kit (SDK) called Sandboxed Execution Environment. The framework design reproduces what is discussed in the first part. Finally, we’ll evaluate virtual machine introspection techniques as a medium to provide cost effective behavioural extraction features to sandboxes. To conclude this last part, we’ll introduce a virtual machine introspection tool called Nitro.

Requirements

For this research, requirements elicitation was conducted with malware analysts, security researchers and behavioural analysis experts. After several sessions, the requirements were gathered and sorted by relevance. Here follows their brief introduction.

In the past, malware only used to be prevalent on Microsoft Windows operating systems. But in recent years with the introduction of portable devices such as smartphones and tablets, the trend started shifting towards modular malware capable of affecting multiple platforms at once. Operating systems such as Android, macOS and Linux are becoming more and more interesting for attackers. One of the reasons is the adoption of such platforms in enterprises and other organisations, which makes them interesting targets for information exfiltration.

Therefore the support of different operating systems is critical nowadays for the IT security business. A natural consequence is the support for different hardware platforms and devices. Virtualisation technologies are of great benefit for these kinds of problems. Yet
the lack of standards in this field often forces the adoption of multiple virtualisation solutions, rather than a single one. While observing malware analysts at work, it became evident how the use of different forensic tools and technologies plays a key role in the analysis of a sample. Behavioural analysis encompasses multiple fields of expertise, such as memory and file system forensics or operating system internals. Each of these fields allows observation of the behaviour of a program from a specific angle. The co-operation of these competences is of primary importance and should be taken into account when designing these types of solutions. It should be easy for malware experts to integrate their knowledge and skills into the behavioural analysis platform.

Finally, a straightforward requirement is the development of a simple and well documented framework enabling users and developers to achieve what is mentioned above.

A flexible design

The event-driven architectural pattern is based on a simple mechanism consisting of the production and consumption of events. Several entities act as producers and consumers of events that are shared between all the components of the platform. An event is represented as a change of state within the architecture. The transition may trigger internally in some of the platform’s components or externally as a request from a user. Events may be simple entities, such as strings or integers, or more complex messages with headers and containing further metadata – for example, names, time-stamps and payload.

To be a complete Event-Driven Architecture (EDA), the system must share the events between all its components. It requires a common buffer able to dispatch and, if necessary, queue the events in case the consumers are not consuming them quickly enough.

EDAs are loosely coupled and support modular design. In an EDA, the events are neither aware of the causes of their triggering nor of the consequences they might generate once consumed. In addition, the single components of the architecture are free to limit their knowledge based only on their own responsibilities, ignoring all the rest of the system internals.

A good design pattern for implementing the event flow control is the observer pattern. The observer pattern is one of the simplest design patterns for implementing EDAs. An observer-based architecture consists of multiple observer entities subscribing to one or more observable objects.

In this context, the sandbox plays the observable role, whereas the forensic tools are the observers. Each observer can subscribe, through the observable sandbox, different handlers to specific events. Any time a change occurs in the sandbox state, a related event is triggered, allowing the forensic tools to act accordingly. Some examples of events are the powering on of the execution environment, the insertion or removal of a device or a specific API call from the test sample.

A protocol consisting of the formal description of the events and their approximate sequence order is introduced. The protocol is the contract the developers (the forensic analysis experts) need to honour and the only thing they need to be aware of. As the observers are totally decoupled from each other, there is little or no risk of interference.

Such design promotes the development of generic forensic plugins which are well isolated and can be re-used in different contexts. The end user can run multiple tests employing different forensic plugins according to his or her need.

Sandboxed Execution Environment

The Sandboxed Execution Environment (SEE) is a framework that implements what has been previously illustrated.

Developed in the Python language, SEE offers a unified interface for managing different virtualisation technologies. It allows for easy execution and control of multiple sandboxes. Each sandbox is paired with a plugin loading interface and an observer pattern-based event flow controller. The user can describe each test case, specifying the sandbox type and characteristics, as well as the forensic plugins to be used. Once launched, the framework will apply the designed protocol and route the events to the plugins. Each plugin has access to the sandbox resources. The resources consist of the virtual machine memory, the CPU and its registries, the attached devices, the disk and the network interfaces. At any point during the execution, the resources can be inspected to extract relevant information. It is possible, as well, to change the state of such resources. Each state change should result in specific events being triggered, making sure all plugins are aware of what is happening.

Internally, SEE relies on the Libvirt Virtualisation API to offer the support for multiple virtualisation technologies. It currently supports QEMU/KVM, Virtualbox and Linux Containers. More virtualisation technologies can be easily added by exposing the related Libvirt API.

VMI for flexible sandboxing

In this second part, Virtual Machine Introspection (VMI) technologies are evaluated with the goal of improving flexibility in sandboxing solutions. VMI as a concept will be introduced first, followed by the possible applications. Following this is an explanation as to why it reduces the maintenance and development cost of modern sandboxes.

Core concepts

The term VMI first appeared in a publication in 2003 and is described as “Inspecting a virtual machine from the outside, for the purpose of analysing the software running inside it.” This first research was driven by the desire to create a new type of intrusion detection system (IDS), combining the benefits of the attack resistance of a network-based solu-
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات