Bromium®

Understanding Bromium® Micro-virtualization for Security Architects

Bromium®

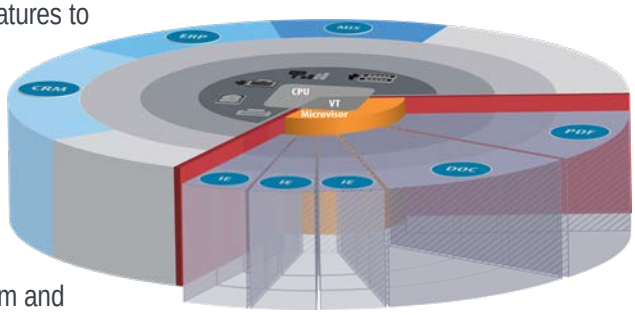
Introduction

Bromium® Micro-virtualization is a powerful construct that enables an endpoint to secure itself "by design" - by hardware-isolating each untrusted user task using built-in features of the Intel® CPU. (For a technology primer on micro-virtualization, please see this whitepaper.) The approach is intuitively appealing, and has been practically validated in publicly available penetration tests including by NSS Labs.

This white paper describes the technology and how it works in terms useful to a security architect who needs to reason about security properties of the system in a more formal way. It dives under the hood of Bromium vSentry to show how micro-virtualization works to make enterprise endpoints inherently secure – without the need for legacy endpoint security technologies that rely on signatures. It shows how the technology allows IT to empower end users to access the power of the web, embrace mobility and new work styles, while protecting the enterprise.

Background

Micro-virtualization relies on the use of Intel CPU features to hardware-isolate individual untrusted user tasks – those that involve code or data from some external system that is being accessed locally, such as browser tabs, documents and media.



These hardware-isolated tasks are contained within micro-VMs, which protect the operating system and other tasks from compromise. Valuable data, networks and devices are not available in a micro-VM – so an attacker cannot steal data or penetrate the enterprise, and execution within a micro-VM is ephemeral, with all changes to system state saved in a throw-away cache, so an attacker cannot persist. Upon the termination of the task, the micro-VM and the throw-away cache are simply discarded together with any malware.

Key Technology Innovations

Recognizing that detection technologies cannot keep up with the rapid evolution of attacks, security vendors, application vendors and industry analysts have begun to advocate execution isolation as a key requirement for next-gen endpoint security. As a first step, it is thus important to place micro-virtualization in the context of existing, well understood isolation technologies.

- Classical OS design implements isolation through separation of untrustworthy user processes from the system kernel, and recent research has focused on improving OS design;
- Sandboxes attempt to retrofit software-based isolation between user space application processes and existing vulnerable operating system kernels, using software;

- Multiple independent operating system instances in VMs can be mutually isolated by a hypervisor; and
- Micro-virtualization isolates individual untrusted tasks within a single OS, mutually isolating them and the desktop in micro-VMs.

Are all Isolation technologies equal?

For the answer to this question, Neil McDonald, a Gartner Fellow in Security, published an analysis and reference architecture that allows security teams to understand and trade off different isolation technologies. Gartner offers crucial insights and develops a framework for understanding the protection afforded by isolation technologies in general.

- Though sandboxing is well established (it's a feature of most applications that process untrusted content), it is also inadequate against a determined attacker. And while a hypervisor offers robust inter-VM (inter-OS) isolation, it can't protect against an attack *within* a VM (such as a virtual desktop).
- Using a hypervisor to isolate each vulnerable application within its own VM (using multiple independent OS instances typically hosted on a desktop system) is impractical for end-users since it impacts the user experience, and it is complex to provision, manage and deploy.
- Finally, it is important for any technology to be useful in the context of today's widely deployed OSes and enterprise applications – without requiring a wholesale forklift.

Gartner's analysis allows a security architect to understand the capabilities and limitations of each kind of isolation technology and the granularity at which it is applied (for example: process, application, or VM).

Protection vs Isolation

Gartner advocates the use of granular isolation of untrusted code to protect enterprise endpoints. This is an intuitively appealing concept: If the untrusted code attacks the system, or crashes, then the system as a whole ought to remain completely protected. But unfortunately it's not quite that simple: "granular protection" is quite different from "granular isolation".

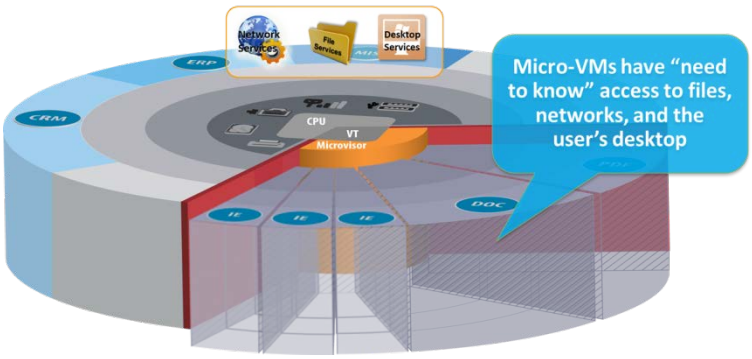
Imagine for a moment that we could isolate the execution of untrusted code down to the application thread level. If a thread of an application were to be compromised, would the fine-grained isolation help to protect the application or the system as a whole? It would not: An application thread has access to all data and resources that the application has access to – devices, the file system, and kernel services such as networking – including an ability to access privileged enterprise networks. An application can read the registry, including the SAM, and it can modify all sorts of program and system configuration data.

The overall security of the system therefore depends on both the granularity of execution isolation, and the granularity of access to security-critical system resources and data. The architectural construct that security analysts use to reason about the latter is the Principle of Least Privilege.

Micro-virtualization: Granular Isolation and Guaranteed Protection

We aim to achieve granular isolation of untrusted code within any endpoint. This requires both granular isolation of execution and granular control over key system resources that must be protected to guarantee system-wide security. We approach these two needs separately, since they are orthogonal.

Granular control over access to system resources results from the rigorous application of the principle of least privilege to an isolated task or user activity. But it works the other way round too: the unit of execution (the untrusted code in a Bromium micro-VM) is selected in such a way as to minimize the set of resources required – according to the principle of least privilege. So the principle of least privilege becomes a constructive principle that allows us to reason about the security of the system, given that any isolated task might be compromised. The constructive principle also leads to a natural user-centric approach to security – one which allows us to secure the system without changing the end user experience in any way.



In the Bromium architecture the Microvisor implements a Least Privilege Separation Kernel between untrusted tasks and the desktop OS. It is the only Separation Kernel that takes advantage of the tiny code-base of a specialized hypervisor to *dynamically* apply Least Privilege at a granular level between tasks within a single running OS instance. Moreover it is the industry's first *general purpose* Separation Kernel that can protect existing, widely deployed OSes and their applications, and that can be deployed and managed using today's management tools (SCCM, AD or a security console like McAfee® ePO).

Hardware Isolated Execution

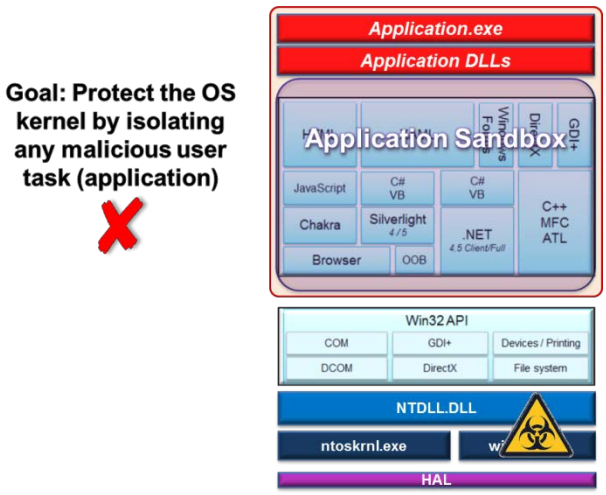
Hardware-isolation is a core tenet of our design because it offers the most robust barrier to attack. Re-stating the problem we posed in the introduction: Assume we can use it to isolate any execution. The key question is *what to isolate* (OS, application, or perhaps even a thread) in order to simultaneously maximize security and deliver an optimal end-user experience. Both traditional virtualization and sandboxing fail to meet these needs:

- Hardware isolation of one or more desktop images in VMs on a single PC or even a server (e.g.: VDI) may deliver manageability benefits, but each desktop is no more secure than a native PC because a

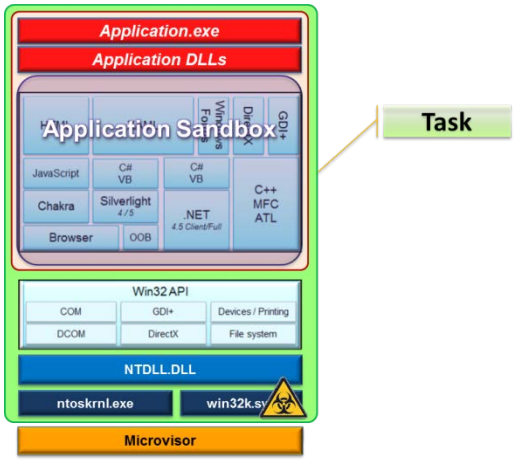
hypervisor cannot provide intra-VM isolation. Moreover virtualization pervades the user experience because the user is explicitly aware of the different contexts of different VMs.

- Running each application within its own VM using a type-2 hypervisor and graphical tricks to make a VM appear to the user to be an application (e.g.: the browser VM is hidden behind a standard looking browser icon on the desktop) doesn't help either. Applications need to inter-communicate (e.g.: cut & paste) and VM isolation won't support this. Moreover the management complexity of this approach is substantial – multiple app-VMs have to be created and managed through their lifecycles, in addition to managing the desktop. Finally, within a single application (such as the browser) one part of the application could be used to attack another (malware in one browser tab could attack a secure site in another tab).
- Application isolation is another approach. Sandboxing is frequently used to protect the OS kernel from a malicious or compromised application– for example a browser. Of course in the case of sandboxing the isolation is software based, but let's assume that it, in combination with other software protection mechanisms are as robust as hardware isolation (One can conceive of application isolation approaches that use VT. For example, see the Microsoft Research Drawbridge project.) But using today's technology, let's assume we use every process isolation technique at once: IE and its sandbox on top of an OS improvement sandbox such as Sandboxie, further strengthened by Microsoft EMET (a powerful free ASLR tool) and running on a system with the full complement of legacy endpoint protection.

Unfortunately, as Rahul Kashyap and Rafal Wojtczuk of Bromium demonstrated at Black Hat Europe 2013, if a malicious application compromises the kernel directly it can bypass all protection. For example, on an unpatched Windows desktop, navigating to a web site that causes the kernel to parse a poisoned font file is sufficient. This exposes the Achilles Heel of the desktop: a long and growing list of kernel logic CVEs, which are impossible to protect against using detection methods and that cannot be prevented using application isolation.



Neither application sandboxes (eg: Adobe, Microsoft, Google) nor OS improvement sandboxes (Invincea, Sandboxie, Trustware Bufferzone, ZeroVulnerabilityLabs) can block such attacks. Although these are powerful technologies that substantially improve system security in the event of a malicious application-to-kernel attack, direct exploitation of a kernel vulnerability completely bypasses this approach, and so sandboxing is unable to meet our requirement for rigorous isolation of untrusted tasks.



What can we conclude from this? We must assume that the kernel can be compromised directly by a malicious task, so if we are to isolate its computation, we must also isolate all of the task's kernel activity too.

Kernel isolation is achieved in the Bromium architecture because the Microvisor hardware-isolates both the user-space and the kernel activity of a task. The system is robust to any kernel execution attack. But what about access to kernel-visible system resources? The moment we start to talk about kernel activity, we need to recognize that a kernel compromise (even if isolated) will expose all visible kernel resources to the malware, so the isolation must extend to all task-relevant, kernel controlled resources. So when the attack succeeds, the set of system resources available to the task must be minimal. Rigorous application of the principle of least privilege allows us to identify the boundaries of a task and the minimal set of resources it needs.

Least Privilege dictates the minimum set of system resources (network, file system, desktop) that any task requires to work correctly: For example, in the context of the browser, a task is an application context defined by the top level domain (the site TLD).

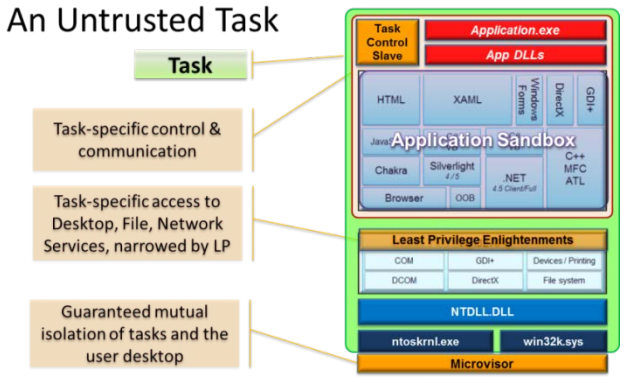
What resources does facebook.com really need? It needs its cookie, and access to the untrusted web. If the browser tab for facebook.com is compromised (the user clicks on a poisoned advertisement), we can tolerate loss of the cookie. We can live with the fact that malware will have access to the untrusted Internet. The system (and the enterprise) will still be safe if:

- The malware cannot see any user keystrokes, mouse input or gain access to the screen (to copy pixels from the display, or display any content to the user).
- The malware cannot access any files other than the Facebook cookie. The malware cannot gain access to any valuable networks or sites (e.g.: SaaS sites, or the Intranet).
- Facebook does not need access to any other files, or USB devices. Least Privilege dictates that it *must not* have access to any other resources unless they are explicitly required, and then only under precise policy control, and only for a short duration. For example (there are many more such examples, but the goal is to demonstrate that we can deliver an unchanged user experience whilst *dynamically enforcing LP*):
    - If the user wants to upload a photo to Facebook, she can select the photo (in the usual way) on the desktop, and then (only) the selected file will be injected into the hardware-isolated task that is rendering the facebook.com browser tab.
    - If the user wants to download a file, it can be allowed to persist outside the confines of the isolated task, but only if we remember the fact that it is untrusted, so that it can only ever be accessed in another hardware-isolated task.

The Microvisor Enforces Dynamic "Need to Know"

The Microvisor implements a dynamic, hardware-backed Least Privilege Separation Kernel between untrusted tasks and the user desktop. It virtualizes access to all shared system resources in such a way as to enforce mutual isolation of all tasks and the desktop.

- It uses Intel VT to hardware-isolate task execution of both user-space and kernel activity, but unlike hypervisor-based virtualization, it does not need to virtualize device hardware, which is driven by the desktop kernel.
    - Instead, the Microvisor virtualizes system resources at a layer that is relevant to tasks, including the file system, registry, network services, and desktop services such as the clipboard, display, and user input. This is achieved using light-weight "task-level enlightenments" that build on standard OS APIs.
    - Execution is non-persistent: all changes (to any system resource) made by a task during execution are saved in an ephemeral, throw-away cache.



Summary

- Micro-virtualization is the only technology that can hardware-isolate all untrusted activity of an application at a granular level (including intra-application isolation, when required), protecting the system or other components of the application context, even when the kernel is attacked directly.
- Its robustness results from hardware-isolated execution and the "throw-away cache" of all changes made during execution to any system resources (memory, files, registry), and granular application of the principle of least privilege to minimize the set of resources available in the context of untrusted execution.
- Just as least-privilege allows us to decide what resources to expose to an untrusted task, it also defines how we should treat data persisted by an untrusted task, namely that it not be trusted and that access shall only be permitted in the context of the task, isolated in a micro-VM.
- Finally, Micro-virtualization can be straightforwardly applied to todays deployed OSes and applications on any modern PC or server.