View on GitHub

Simplevisor

SimpleVisor is a simple, portable, Intel VT-x hypervisor with two specific goals: using the least amount of assembly code (10 lines), and having the smallest amount of VMX-related code to support dynamic hyperjacking and unhyperjacking (that is, virtualizing the host state from within the host). It runs on both Windows and UEFI.

Download this project as a .zip file Download this project as a tar.gz file

Introduction

Have you always been curious on how to build a hypervisor? Has Intel's documentation (the many hundreds of pages) gotten you down? Have the samples you've found online just made things more confusing, or required weeks of reading through dozens of thousands of lines and code? If so, SimpleVisor might be the project for you.

Not counting the exhaustive comments which explain every single line of code, and specific Windows-related or Intel-related idiosyncrasies, SimpleVisor clocks in at about 500 lines of C code, and 10 lines of x64 assembly code, all while containing the ability to run on every recent version of 64-bit Windows, and supporting dynamic load/unload at runtime.

Additionally, SimpleVisor utilizes a lightweight OS-library for Windows-specific functionality, separating out the hypervisor pieces from the Windows-specific pieces. Leveraging this portable design, a UEFI version of SimpleVisor is also now available. Note however, that it does not have robust support for MP environments due to issues with UEFI, and that loading an operating system will eventually result in a crash as the OS will hit unimplemented code paths due to its re-configuration of processor resources. Virtualizing the entire boot of the operating system from UEFI is beyond the scope of the project.

SimpleVisor can be built with Visual Studio 2015 Update 3, and while older/newer compilers have not been tested and are not supported, it's likely that they can build the project as well. It's important, however, to keep the various compiler and linker settings as you see them, however.

SimpleVisor has currently been tested on the following platforms successfully:

- Windows 8.1 on a Haswell Processor (Custom Desktop)
- Windows 10 Redstone 1 on a Sandy Bridge Processor (Samsung 930 Laptop)
- Windows 10 Threshold 2 on a Skylake Processor (Surface Pro 4 Tablet)
- Windows 10 Threshold 2 on a Skylape Processor (Dell Inspiron 11-3153 w/ SGX)
- VMWare Workstation 11, but without EPT (VMWare does not support 1GB EPTs)
- UEFI 2.4 on an Asus Maximus VII Extreme Motherboard (Custom Desktop)

At this time, it has not been tested on Bochs, but there's no reason why SimpleVisor could not run in such an environment as well. However, if your machine is already running under a hypervisor such as Hyper-V or Xen, SimpleVisor will not load.

Keep in mind that x86 versions of Windows are expressly not supported, nor are processors earlier than the Nehalem microarchitecture, nor is Windows 7. Support for the latter two is easy to add and exists in certain forks.

Motivation

Too many hypervisor projects out there are either extremely complicated (Xen, KVM, VirtualBox) and/or closed-source (VMware, Hyper-V), as well as heavily focused toward Linux-based development or system. Additionally, most (other than Hyper-V) of them are expressly built for the purpose of enabling the execution of virtual machines, and not the virtualization of a live, running system, in order to perform introspection or other security-related tasks on it.

A few projects do stand out from the fold however, such as the original Blue Pill from Johanna, or projects such as VirtDbg and

HyperDbg. Unfortunately, most of these have become quite old by now, and some only function on x86 processors, and don't support newer operating systems such as Windows 10.

The closest project that actually delivers a Windows-centric, modern, and supported hypervisor is HyperPlatform, and we strongly recommend its use as a starting place for more broadly usable research-type hypervisor development. However, in attempting to create a generic "platform" that is more broadly robust, HyperPlatform also suffers from a bit of bloat, making it harder to understand what truly are the basic needs of a hypervisor, and how to initialize one.

The express goal of this project, as stated above, was to minimize code in any way possible, without causing negative side-effects, and focusing on the 'bare-metal' needs. This includes:

- Minimizing use of assembly code. If it weren't for the lack of an __lgdt intrinsic, and a workaround for the behavior of a Windows API, only the first 4 instructions of the hypervisor's entry point would require assembly. As it stands, the project has a total of 10 instructions, spread throughout 3 functions. This is a massive departure from other hypervisor projects, which often have multiple hundreds of line of assembly code. A variety of OS-specific tricks and compiler shortcuts are used to achieve this result.
- Reducing checks for errors which are unlikely to happen. Given a properly configured, and trusted, set of input data, instructions such as vmx_vmwrite and vmx_vmread should never fail, for example.
- Removing support for x86, which complicates matters and causes special handling around 64-bit fields.
- Expressly reducing all possible VM-Exits to only the Intel architecturally defined minimum (CPUID, INVD, VMX Instructions, and XSETBV). This is purposefully done to keep the hypervisor as small as possible, as well as the initialization code.
- No support for VMCALL. Many hypervisors use VMCALL as a way to exit the hypervisor, which requires assembly programming (there is no intrinsic) and additional exit handling. SimpleVisor uses a CPUID trap instead. Relying on little-known OS functions to simplify development of the hypervisor, such as Generic DPCs and hibernation contexts on Windows, or the PI MP protocol on UEFI.
- Supporting EPT/VPID in a very simple fashion, to demonstrate a solid base of the simplest possible implementation of the feature.
- Portability and isolation of OS-specific routines.

Another implied goal was to support the very latest in hardware features, as even Bochs doesn't always have the very-latest Intel VMX instructions and/or definitions. These are often found in header files such as "vmcs.h" and "vmx.h" that various projects have at various levels of definition. For example, Xen master has some unreleased VM Exit reasons, but not certain released ones, which Bochs does have, albeit it doesn't have the unreleased ones! One such example is the usage of 1GB EPT entries, which for example VMWare does not virtualize correctly.

Finally, SimpleVisor is meant to be an educational tool -- it has exhaustive comments explaining all logic behind each line of code, and specific Windows or Intel VMX tips and tricks that allow it to achieve its desired outcome. Various bugs or poorly documented behaviors are called out explicitly.

Installation on Windows

Because x64 Windows requires all drivers to be signed, you must testsign the SimpleVisor binary. The Visual Studio project file can be setup to do so by using the "Driver Signing" options and enabling "Test Sign" with your own certificate. From the UI, you can also generate your own.

Secondly, you must enable Test Signing Mode on your machine. To do so, first boot into UEFI to turn off "Secure Boot", otherwise Test Signing mode cannot be enabled. Alternatively, if you possess a valid KMCS certificate, you may "Production Sign" the driver to avoid this requirement.

To setup Test Signing Mode, you can use the following command:

```
bcdedit /set testsigning on
```

After a reboot, you can then setup the required Service Control Manager entries for SimpleVisor in the registry with the following command:

```
sc create simplevisor type= kernel binPath= "<PATH_TO_SIMPLEVISOR.SYS>"
```

You can then launch SimpleVisor with

```
net start simplevisor
```

And stop it with

```
net stop simplevisor
```

You must have administrative rights for usage of any of these commands.

References

If you would ike to know more about my research or work, I invite you check out my blog at http://www.alex-ionescu.com as well as my training & consulting company, Winsider Seminars & Solutions Inc., at http://www.windows-internals.com.

https://github.com/upring/virtdbg

http://xenbits.xen.org/gitweb/?p=xen.git;a=summary

https://github.com/svn2github/bochs

https://github.com/rmusser01/hyperdbg

http://invisiblethingslab.com/resources/bh07/nbp-0.32-public.zip

https://github.com/tandasat/HyperPlatform

Caveats

SimpleVisor is designed to minimize code size and complexity -- this does come at a cost of robustness. For example, even though many VMX operations performed by SimpleVisor "should" never fail, there are always unknown reasons, such as memory corruption, CPU errata, invalid host OS state, and potential bugs, which can cause certain operations to fail. For truly robust, commercial-grade software, these possibilities must be taken into account, and error handling, exception handling, and checks must be added to support them. Additionally, the vast array of BIOSes out there, and different CPU and chipset iterations, can each have specific incompatibilities or workarounds that must be checked for. ***SimpleVisor does not do any such error checking, validation, and exception handling. It is not robust software designed for production use, but rather a reference code base***.

License

Simplevisor maintained by ionescu007