[title slide]

Hi.

I'm here to talk about one of the biggest risk areas for the security of x86 hypervisors: qemu, in its role as PC emulator.

I'll be looking at it from the perspective of qemu's role in a Xen system. Other Free Software full-PC hypervisors also use qemu, they are structured differently. But in Xen systems qemu plays a smaller role (and is sometimes even absent), so there are different difficulties and different opportunities.

I'm going to talk about some work that Stefano Stabellini and I have been doing to improve the security of one of the most important Xen configurations.

[PV vs HVM slide]

x86 Xen guests can run in two modes: the traditional one is PV (which stands for paravirtualised). This involves modifying the guest to run specifically under Xen. Many Free Software operating systems have been ported to run as a Xen PV guest. This is still, in general, the best mode.

But sometimes its necessary to run unmodified guests. Xen HVM provides an environment that looks to the guest just like a whole PC, complete with PCI bus and the traditional PC peripherals. To do this, Xen needs an implementation of those peripherals: and that is where qemu comes in.

In many Xen HVM configurations, most if not all of the emulated PC is used only during booting: if the guest has suitable Xen-specific drivers, it can load them at a suitable point during boot, and switch over. This is a good idea because it's faster and more scalable. But from a security point of view, a malicious guest can still attack the whole of the emulated PC (although sometimes it might need to reboot itself to do so).

[XSAs slide]

This is a problem because a PC is a very complicated thing to emulate. Inevitably, almost all code has bugs. The more complicated the code, the more bugs. The PC is also a very broad interface: it provides a lot of different facilities. So the attack surface is very large.

That means lots of security problems. Here we see a selection from the past year.

Several of these are only applicable in non-default configurations, and they don't all allow immediate compromise of the host by a guest. But it's still a substantial proportion of the security risk posed to a host by a guest running in a virtual x86 PC.

In principle there is no architectural reason why the PC emulation needs to run with the full privileges of the host. It's that way in most Xen HVM setups because the PC emulation is combined together in the single program qemu. In most non-Xen scenarios where qemu is used, qemu is responsible for most of the configuration and management of the domain, so it is necessarily trusted.

But in Xen most of the other functions, which need privilege, are done by other parts of the Xen system - parts of the Xen system which generally have a much narrower and simpler and therefore safer interface to the guest.

(The main exception to this is the Xen code which deals the astonishing complexity and variety in the x86 architecture. That would be a whole talk by itself.)

This theoretical flexibility is used by some more advanced Xen-based downstreams which a security focus. But the approaches for an ordinary Xen administrator, who gets a fairly vanilla Xen from their operating system distro, are less sophisticated.

[User options]

As an x86 Xen user - that is, an administrator of a Xen system - you have several options for how to deal with the risk from security bugs in the qemu PC emulator.

The best approach to this kind of situation is to simply avoid exposing, to potentially hostile guests, anything which is not strictly necessary. In the case of qemu in Xen, that means running a PV guest. Xen PV guests are still the best choice from a security point of view.

But if you need to run HVM for some reason (maybe your guest operating system doesn't run on Xen PV) then your choices right now aren't brilliant. Currently the default is to simply run qemu as a process, as root in dom0. Any security bug which allows the guest to compromise qemu is a bug which compromises the whole system.

It is still possible to run the device emulator qemu in a separate domain. It then runs only with the privilege of the guest. We call this 'device model stubdomain'.

But there are a number of reasons why most ordinary installations are not able to use this. It is currently only supported with an ancient version of qemu (specifically provided for this purpose by the Xen project). This old qemu (we call it 'qemu-xen-traditional') is no longer receiving any new features.

Most distros find building this too cumbersome and do not want to try to support it, so if you want to do this you will have to build it yourself.

If you can do so then the limited feature set is probably still fine if you only want it for booting. And in that case, bugs in that qemu are not security bugs at all: the stub device model domain runs only with the privileges of (and over) the guest VM.

But it's not practical to ship this as the default configuration, particularly for distros.

[slide with more user options]

We have two alternatives in the works.

The most sophisticated of these is to port a modern qemu to the Rump Kernel project. Rump Kernels are a form of unikernel. Derived from NetBSD, they provide a way to build existing programs which expect a POSIXy kind of environment, to run as a single image directly on Xen (or indeed, on baremetal, or in other hypervisors or environments).

We have got some way with this, but it's quite complicated. The build system, in particular, is exciting. It's essentially a little miniature distribution. And we have to provide a lot of Xen-specific control interfaces to be able to make a qemu device model work in a rump kernel. Sadly this probably isn't going to be ready in the next release of Xen, 4.7.

In the meantime we want to do something to make security bugs in the device model less of a problem. So we have started a miniature project to deprivilege the device emulator within dom0, by running it as a separate Unix user.

This won't do much for resource exhaustion attacks, at least right away, because it's difficult to stop all the ways that a Unix process might starve the rest of the system.

But getting rid of the immediate host compromise is definitely worthwhile. We hope to have done this − or most of it − for Xen 4.7 (which is currently targeting June 2016).

I'm going to take a look under the covers at how we intend to achieve this:

[slide with technical details]

Actually running a device model in dom0 as a non−root user is not entirely trivial.

Firstly, the device model needs to be able to access the underlying resources (such as disks and networks) that it is trying to present to the guest as emulated IDE and ethernet controllers. For example, it will need access to (say) the LVM volume containing the guest disk image.

Fortunately, since these emulated PC devices don't support hotplug, we can have qemu open the relevant devices as root, at startup, and then drop privilege later.

There is an exception: the emulated cdrom, which needs to support insertion and removal. We will deal with this by having the toolstack library, which in a Xen system invokes and controls qemu, open the device, and pass the relevant fd to qemu.

Secondly, the device model needs intimate access to the innards of the guest, the same way that real hardware would have (for example, it needs to be able to do DMA). This involves making hypercalls to access and manipulate guest domain.

If qemu runs as root in dom0, it inherits dom0's whole−system privilege. When it runs in its own domain, it is specifically created as a service domain for the guest, so that Xen knows that it can access that guest.

If we want to deprivilege it in dom0, it's more complicated.

To make qemu's privilege drop effective we need to give it an ability to make hypercalls (and access its guest's memory and so on), but restrict that to hypercalls relating to the management of the specific guest. Only the dom0 kernel can identify and distinguish the qemu process from other parts of dom0, but only the hypervisor understands which hypercalls have which security properties. So, we are developing a small new feature in the hypercall interface that would allow the information about the caller to be presented to Xen along with the hypercall arguments.

qemu will open the dom0 Xen hypercall and memory access devices at startup. Before dropping privilege it make a system call to tell the kernel to from now on always attach the appropriate rider to all its hypercalls. Then Xen can make the right access control decision.

A similar consideration, and a similar approach, applies to xenstore, the low−level structured interdomain communication system which is used as the control plane for domain management and paravirtualisation.

Thirdly, depending on the configuration, in current systems, qemu may be providing paravirtualised devices as well as full system emulation. (The toolstack decides whether PV devices are provided by qemu or by the dom0 or driver domain kernel, according to the requested guest configuration and the configured underlying resources.)

The paravirtualised devices must support hotplug; but they provide a narrower, hypervisor−friendly interface: the paravirtualised protocols are generally the primary security boundary. So they can and should be provided by software with greater privilege to access underlying resources.

To make this possible, we are reorganising the qemu support processes so that a guest might get two qemus: one deprivileged, which does full system emulation; and one which retains privilege but provides only paravirt interfaces.

We have concrete proposals for all of these pieces, but they have not yet been fully agreed, tested, and deployed. Nevertheless, we hope to get this done for this summer's Xen release, 4.7.

[back to user options slide]

So to summarise:

It's still best to use a Xen PV guest if you can.

For those of you who need a full virtual PC, we hope to reduce the security impact of bugs in the qemu system emulation: by running the emulator as a non−privileged dom0 user, by default.

More sophisticated − and more secure − privilege separation will of course continue to be available for Xen−based projects and vendors with a security focus, and in the longer term we hope that full device model stubdomains, with a modern qemu, based on rumpkernels, can become the default.

But until then we will help users exploit the existing Unix user security boundary in their dom0 to help contain the qemu device model.

[questions]