# Software [In]security: Assume Nothing

By Gary McGraw, Ivan Arce

Date: Apr 30, 2010

Article is provided courtesy of Addison-Wesley Professional.

Return to the article

According to software security expert Gary McGraw, co-author of Exploiting Software, Microsoft may be forgetting the old mantra of thinking like an attacker by deciding not to patch a vulnerability in the Virtual PC Hypervisor.

## Is Microsoft Forgetting a Crucial Security Lesson?

One of the central lessons of Exploiting Software is that attackers often do whatever they can to uncover and undermine assumptions made by system developers. In fact, the mantra "assume nothing" is an important part of seeing things from the attacker's perspective. Regardless of whether or not developers and architects themselves should all be taught to think like an attacker (an ongoing debate), it is certainly the case that security analysts must! Microsoft seems to have forgotten that lesson of late, with potentially dire consequences going forward.

### Software Security Progress at Microsoft

Before we get started, a big tip of the (white) hat: Microsoft has made a huge amount of progress on software security since the inception of the Trustworthy Computing Initiative way back in 2001 — and their current generation of products reflects that. The difference in security posture between Windows 95/98 (which had no kernel) and Windows 7 is like night and day. Kudos to Microsoft for the very real security progress they have made.

At least a portion of the improved security is due to various security mechanisms that serve to make software exploit harder. Data execution prevention and randomized memory mapping are among those mechanisms. If a given application is mapped into memory differently by the operating system, or if data segments are marked non-executable, then computing offset tables, building attack payloads, and crafting exploits that work every time becomes much harder for attackers. Given the propensity of targets out there, making your software only a modicum harder to attack may well cause the bad guys to move along to something easier to attack.

Combined with static code analysis, world-class penetration testing, threat modeling (or architectural risk analysis, depending on your favorite software security religion), and other software security activities, security mechanisms make a big difference in Microsoft's attack surface and the ease of crafting an exploit. Microsoft has an exceptionally impressive BSIMM result, serving to emphasize the very real progress they have made in software security.

Sound good so far? The problem is that Microsoft appears to be resting on its hard-earned software security laurels. Here's why.

### Time Warp to Broken Security

Earlier this year, Nicolás Economou from Core Security, whose founder and CTO is an author of this article, discovered a design flaw in the Virtual PC Hypervisor made by Microsoft. (A complete description of the problem can be found in the Core Advisory, but we'll provide the technical gist of it here.) The Virtual PC Hypervisor is important because, among many other things, it allows Windows 7 users to run applications designed for previous versions of Windows (*e.g.*, Windows XP SP3). This is a critical factor in enterprise adoption of Windows 7.

A vulnerability in the memory management of the Virtual Machine Monitor (VMM) makes memory pages mapped above two gigabytes (GB) available with read or read/write access to user-space programs running in a Guest operating system. That's bad. Operating systems based on Microsoft Windows NT technologies provide a flat 32-bit virtual address space that describes four GB of virtual memory to 32-bit processes. The address space is used to map a process's executable code and the data it uses during runtime. For performance and efficiency reasons, the process address space is usually split so that 2 GB of address space are directly accessible by a user-mode application process, and the other 2 GB are used to map the code and data of the operating system and are thus only accessible to kernel code. Any attempt from a user-space process to dereference and use memory contents mapped at addresses above the 2GB boundary are supposed to trigger an exception and terminate the offending process. The vulnerability changes all that.

Incorrect memory management by the VMM of Virtual PC makes portions of the VMM worker memory available for read or read/write access to user-space processes running in a Guest OS. Leaked memory pages are mapped on the Guest OS at virtual addresses above the 2GB limit — memory that shouldn't be accessible for user-space programs.

Using this vulnerability, it is possible to bypass security mechanisms of the operating system such as Data Execution Prevention (DEP), Safe Structured Error Handling (SafeSEH) and Address Space Layout Randomization (ASLR) which are designed to prevent the exploit of security bugs in applications running on Windows.

The upshot of this problem is that applications with bugs that are **not exploitable when running in non-virtualized operating systems** become exploitable if they are run on the (very same) Guest OS virtualized in Virtual PC. To give a specific example, an application running on Windows 7 in XP Mode may well be exploitable while the same application running directly on a Windows XP SP3 system is not.

Speaking as security consultants who make decisions about bug severity and which problems should and should not be fixed all the time (yes, sadly there are many known bugs that go unpatched), we are worried. You see, software bugs that in most cases would not be considered security-relevant (because of the security mechanisms we discussed above) and for that reason not prioritized for mitigation will suddenly become exploitable security bugs in the context of this vulnerability. This is like taking a time machine back so you can run on early (and problematic) Windows operating systems that are unprotected.

We do admit that developing an exploit for this situation is one notch more complicated than normal. The particular vulnerability must be used in combination with another, completely distinct application bug in an application running on the Guest OS in Virtual PC. There are two ways to find such applications: 1) find an unpatched version of the application (far fetched and lame) or 2) find a version of the application with an existing bug that was deemed not worth mitigating due to the existence of no-longer-present security mechanisms (much more likely). In the case of 2, what we have is a classic case of assumptions disappearing and security problems popping to the surface like Styrofoam in the ocean.

## Zooming Out to Assume Nothing

The vulnerability that Core found does not seem to worry Microsoft who has decided (so far) not to issue a security bulletin or patch the problem. (The report timeline section of Core's Advisory makes particularly interesting reading.) This may be because using the vulnerability to execute code in the context of the non-virtualized OS (host) does not seem possible. However, use of the vulnerability to implement covert inter-process communications within the virtualized OS or to establish inter-VM communication both seem very likely.

We're less worried about this particular vulnerability than we are about the now-exposed (incorrect) assumption that various security mechanisms will always be in place. It's obvious that a complete re-calibration of exploit potential for uncategorized bugs will become necessary if vulnerabilities like the one described here remain in our fielded systems. Not so good for Windows 7.

In our view, design and architecture decisions made for Virtual PC completely invalidate some basic

assumptions about processes in modern Windows operating systems. Like falling dominos, this in turn invalidates almost all anti-exploit mechanisms that Microsoft has built into their OS over the past decade, which then topples over and turns an entire class of bugs deemed un-exploitable on non-virtualized systems into potential vulnerabilities on virtualized systems. Backwards time warp and a table full of fallen dominos.

The practical knowledge to extract from this story is that modern software always relies on a layered number of software and hardware components "underneath" to run properly. In the design and actual implementation of each one of these layers, assumptions are always made about how the other layers work and what type of facilities are and are not exposed. In this particular case, OS security mechanisms were built with a number of assumptions about how process memory mapping, memory protection, and processor privilege levels interact with each other on Windows operating systems.

We should always try to develop a good understanding of the mechanics of the other components that our software relies on. Even further, we should call out the assumptions we may be implicitly relying on about other components. Both of these activities are crucial to assessing risk and impact accurately when the inevitable cracks begin to appear.

## These Are Not the Droids You're Looking For

Microsoft claims that the Virtual PC problem "isn't a vulnerability *per se*" because the problem described only affects "security-in-depth" mechanisms and attackers would need to find and exploit an actual implementation bug to leverage it. Even if Microsoft is right on that count (which we don't think they are), they are ignoring the bigger issue of assumptions. Bugs previously deemed non-exploitable for anything other than crashing systems are now potentially exploitable under a virtualized OS. Because of the way bugs are slated for mitigation in the real world, a majority of those bugs remain unpatched — a problem of prioritization and the enormity of the bug pile in applications.

Another issue is likely in play as well. Windows 7 is a smash hit, and even a whiff of a security issue will do adoption no good.

Regardless of such considerations, in our view, Microsoft should check its assumptions at the door and fix this problem properly.

---