The failure of pysandbox

By **Jake Edge**
November 20, 2013

Running applications securely, such that they cannot escape confinement and affect other parts of the system, is the general goal of various types of sandboxes. They can be used to safely run untrusted code, for example. Java (in)famously has sandboxing built into the language, and pysandbox was an attempt to do something similar for Python. But the developer of pysandbox, Victor Stinner, recently declared that the project has failed in its goals and warned others away from that style of sandbox. There are some useful lessons in Stinner's experience that others looking at sandboxes will benefit from.

Pysandbox takes the approach of allowing semi-arbitrary Python to be run in a protected sandbox namespace. While the goal might be to allow all of the Python language, that has led to several different ways to escape the confinement. The sandbox does attempt to provide a portable mechanism for running some Python code safely, using the standard CPython interpreter. Unfortunately, as his declaration made clear, Stinner is convinced that pysandbox is the wrong approach.

Stinner's post to the python-dev mailing list is worth reading in full. He starts with a bit of history that includes some of the difficulties faced by the project over the years. As time went on, more and more holes were found in the sandbox, which required restricting various Python language features so that they couldn't be used to escape. Recently, a security challenge targeted pysandbox and found two vulnerabilities in less than a day. This has led him to two conclusions. The first is that "pysandbox is broken" at a fundamental level:

> I now agree that putting a sandbox in CPython is the wrong design. There are too many ways to escape the untrusted namespace using the various introspection features of the Python language. To guarantee the [safety] of a security product, the code should be [carefully] audited and the code to review must be as small as possible. Using pysandbox, the "code" is the whole Python core which is a really huge code base. For example, the Python and Objects directories of Python 3.4 contain more than 126,000 lines of C code.
>
> The security of pysandbox is the security of its weakest part. A single bug is enough to escape the whole sandbox.

He outlined some of the kinds of problems that were found. For example, the `__builtins__` dictionary could be modified in various ways to circumvent the sandbox functions and escape. Any segmentation fault in the CPython executable (there are known "crashers" of this sort) could also be used to break out of confinement.

The two most recent vulnerabilities were some of the most fundamental. One used the `compile()` function to get access to the contents of arbitrary disk files (using a syntax error to print lines as part of the traceback message). The other used a traceback object to unwind the stack frame to one in the trusted namespace, then use the `f_globals` attribute to retrieve a global object. In both cases, the fix limited the usefulness of pysandbox even further.

The second fundamental flaw is that pysandbox "cannot be used in practice", Stinner said. Because so many restrictions have needed to be added, pysandbox cannot be used for

翻译为

anything "more complex than evaluating '1+(2*3)'". Basic language constructs like "`del dict[key]`" have been removed because they can be used to modify `__builtins__` and break out of the restricted namespace.

He notes that various folks had contacted him about using pysandbox in web applications, so he believes there is a real need for the functionality. He ended his message with a call for information on alternative approaches beyond the PyPy sandbox he already knows about. Based on what he has learned, he believes a different approach, outside of the standard CPython interpreter, will be required:

> To build a secure sandbox, the whole Python process must be put in an external sandbox. There are for example projects using Linux SECCOMP security feature to isolate the Python process.

Several developers spoke up to thank Stinner for his analysis and to laud his admission of a defeat of sorts. He set out to create a secure CPython-based sandbox and ended up recognizing that it may well be an unattainable goal. As Python benevolent dictator for life (BDFL) Guido van Rossum put it: "Negative results are also results, and they need to be published."

In addition, several posters were not particularly surprised by the outcome Stinner reported. Nick Coghlan noted the "many JVM vulnerabilities" as one indication that sandboxing is a difficult problem to solve. He continued: "the only ones I even remotely trust are the platform level mechanisms that form the foundation of the various PaaS [platform as a service] services, including SELinux and Linux containers." He is skeptical of trying to have sandboxes that are cross-platform or in-process because of the attack surfaces of both CPython and Java.

The PyPy solution may make sense for some, as Maciej Fijalkowski pointed out, but it is a very different model. A special PyPy interpreter is created that cannot do any library or system calls, but instead sends the operation name and arguments to stdout and waits for a response on stdin. In that way, an outer process completely controls the sandboxed program's interaction with the rest of the system. It may be more difficult to use than was envisioned for pysandbox, however.

Creating a sandbox is clearly a difficult problem—solutions are often quite fragile. But the need to be able to safely run more or less arbitrary code is real. In the thread, Terry Reedy listed a number of sites that accept and run Python code, while Stinner noted a Python shell web application that uses Google's App Engine; all of those use some kind of sandbox—presumably one at the operating system level. But Stinner's tale should serve as a cautionary one to anyone considering a CPython-based solution—or the equivalent for other languages.

(Log in to post comments)

The failure of pysandbox
Posted Nov 21, 2013 21:18 UTC (Thu) by **thomas.poulsen** (subscriber, #22480) [Link]

This paper describes a similar development in the R-language: The sandboxR package disables unsafe functions, where as the RAppArmor package confines the execution from the OS.

Reply to this comment

The failure of pysandbox

翻译为

Posted Nov 21, 2013 21:44 UTC (Thu) by **Trou.fr** (subscriber, #26289) [Link]

Which technique does Google use for App Engine ? They allow fully untrusted code to run on their servers, so their solution's architecture could be used as an inspiration for future works ?

Reply to this comment

The failure of pysandbox
Posted Nov 21, 2013 23:24 UTC (Thu) by **dtlin** (✭ **supporter** ✭, #36537) [Link]

I don't know if this is what Google uses, but there's a paper Language-Independent Sandboxing of Just-In-Time Compilation and Self-Modifying Code describing results from porting V8 and C# (Mono) to NaCl. I imagine (C)Python, which doesn't have a JIT, would be a lot easier.

Reply to this comment

The failure of pysandbox
Posted Nov 22, 2013 9:28 UTC (Fri) by **khim** (subscriber, #9252) [Link]

NaCl is part of the solution, but it's not the whole solution: it's only used as “internal sandbox”, there are also another, OS-level sandbox (ptrace? seccomp-bpf? not 100% sure) plus some additional changes in compiler to make it harder to exploit bugs in NaCl and OS-level sandbox. Although in case of NaCl it's mostly about bugs in CPU, not about bugs in NaCl (so far all AMD and Intel's errata's have not listed anything suitable for NaCl sandbox escape, but couple of these needed additional discussions with AMD/Intel representatives and looked awfully close to a potential sandbox escape).

Initially Google also did some changes to a python interpreter to restrict it, but as was pointed out by Victor Stinner it does not really work and is not used with new, python 2.7-based runtime.

Reply to this comment

The failure of pysandbox
Posted Nov 22, 2013 14:07 UTC (Fri) by **ibukanov** (subscriber, #3942) [Link]

> But Stinner's tale should serve as a cautionary one to anyone considering a CPython-based solution—or the equivalent for other languages.

Sandboxing is much easier when the language itself is safe and sandboxing friendly with mostly functional API (like SafeHaskell [1] demonstrates for Haskell) or when its API is very restrictive to begin with (JavaScript). On the other hand sandboxing-as-afterthought for a rich platform just does not work as SecurityManager for Java demonstrated. Given the complexity of Python ecosystem I am surprised that people have bothered with it.

[1] - https://ghc.haskell.org/trac/ghc/wiki/SafeHaskell

Reply to this comment

The failure of pysandbox
Posted Nov 30, 2013 23:44 UTC (Sat) by **speedster1** (subscriber, #8143) [Link]

翻译为

> Given the complexity of Python ecosystem I am surprised that people have bothered with it.

When the payoff is appealing enough, some will take up the challenge and make an attempt even though risk of failure is high.

It's a very tempting goal to be able to usefully sandbox a standard language like python, since most developers are not concerned enough about security to develop in a sandbox-friendly language like SafeHaskell. Building a good sandbox would have allowed someone with higher security-needs than average to still take advantage of the active python development scene.

Reply to this comment

翻译为