Python offers library functions for establishing secure HTTPS connections between a client and server, but few users are aware that those routines suffer from what some would deem a fatal flaw. By default, Python does not check the validity of the SSL/TLS certificate presented by remote servers, which leaves users vulnerable to man-in-the-middle attacks. The project recently decided to correct this shortcoming, although there was considerable disagreement about whether doing so would break existing applications—and, if it does, whether or not such breakage is acceptable in light of the potential security threat.

On August 29, Alex Gaynor [posted](http://lwn.net/Articles/611465/) Python Enhancement Proposal (PEP) 476 to the Python development list. Currently, he explained, the standard Python library does not actually check that the server in an HTTPS connection has an SSL/TLS certificate that is signed by a certificate authority (CA) in any trust root (such as the operating system's certificate database), and it does not check that the Common Name on the certificate matches the server name. The result, of course, is that all programs using the standard library are vulnerable to man-in-the-middle attacks—and, moreover, users are not made aware of this vulnerability. The application developer may reasonably expect that if the SSL/TLS connection is established, then all of the proper steps were taken during the handshake and setup stages. If the user notices the connection at all, he or she, understandably, might also assume it was established securely.

The fix proposed by Gaynor in PEP 476 is straightforward. Python would attempt to verify the certificate presented by the server by querying the system's certificate database. Failure to locate the database would be handled by raising an exception. For situations where an unverified certificate should be trusted (e.g., a self-signed certificate, which would not be signed by a certificate in the system's database), developers (or users) would be able to manually modify their application to accept the certificate.

Earlier proposals had suggested bundling a certificate database into Python (specifically, Mozilla's). But relying on the system's database removes the necessity for Python maintainers to keep their copy of the database up to date, and it simplifies matters for corporate Python deployments that include internal CAs in their system databases.

Gaynor's original proposal suggested making the fix in both Python 3 and Python 2, although the specifics of that part of the plan later became a far more involved debate.

**Defaults**

The list participants were overwhelmingly in support of PEP 476, with one small caveat: the original PEP wording suggests that the change should be enacted immediately, changing the default behavior in the next Python release. Marc-Andre Lemberg, among others, [pointed out](http://lwn.net/Articles/611477/) that a transition plan would be wiser than introducing a backward-incompatible change without warning. He suggested adding certificate-validation failures as a warning in 3.5, then making them an exception in 3.6. In addition, he suggested making it possible to turn the validation behavior off with a command-line switch or environment variable, and perhaps making it possible to pass in certificates not in the system's trust store from the command line.

The primary justification for allowing such workarounds is that real-world Python applications tend not to be deployed in pristine network conditions: corporate intranets, embedded devices, and third-party content-delivery networks all routinely sport problematic security settings. Users may not be able to modify the Python code in question, nor to add certificates to the operating system's trust store. As Nick Coghlan [observed](http://lwn.net/Articles/611478/), corporate intranets may be rife with internal servers running HTTPS because of company-wide edicts that have little connection to the intranet's real security needs.

In addition, Python developers have a real need to test their code with self-signed certificates at times. Providing a workaround allows the developer to test the application in more normal conditions—in essence, not triggering exceptions caused by the network environment. But in such cases, installing a self-signed certificate into the operating system's trust store is rarely the ideal approach.

Few parties in the discussion, however, thought that allowing validation checks to be bypassed by a command-line switch or environment variable was a safe approach. There are just too many ways such a feature could be exploited, not to mention the temptation that would exist for developers to misuse the switch and write code that depends on it.

Christian Heimes [suggested](http://lwn.net/Articles/611480/) one possible solution: making site-wide configuration for certificate validation possible through an "sslcustomize" module akin to the existing sitecustomize. It would allow corporate system administrators to properly configure machines for the vagaries of the intranet, as well as allow individual users to add a personal certificate store to the validation process (without necessarily modifying the platform's built-in certificate database). Heimes's plan seemed reasonable to others, although Coghlan eventually [decided](http://lwn.net/Articles/611481/) that it deserved to be written up as a separate PEP, since many of its potential uses are orthogonal to the specifics of PEP 476.

**When to throw the switch**

Implementing a flexible method for users to tweak SSL behavior to fit the peculiarities of their network answers many of the questions about *how* Python should transition to a validate-by-default stance, but it does not address *when* such a change should be rolled out. Donald Stufft [argued](http://lwn.net/Articles/611482/) in favor of moving the change up from the 3.5 release to the next update for 3.4, noting that "otherwise it’s going to be 2.5+ years until we stop being unsafe by default."

Specifically, Stufft suggested introducing certificate-validation warnings in Python 3.4.2, and changing the default behavior (thus throwing exceptions for validation failures) in 3.5. Not everyone was on board with that time frame; Glyph Lefkowitz [argued against](http://lwn.net/Articles/611483/) the need for a one-cycle warning period, saying that Twisted implemented a similar fix in 14.0, without warning, and had literally received no complaints from its users.

At the crux of the timing question, of course, is determining what Python's responsibility is when users' code stops functioning because of an untrusted SSL/TLS certificate. The responsibility is crystal-clear when the untrusted certificate is actually a man-in-the-middle attack, of course: users need to be alerted to the security threat without delay. But things are more nebulous, again, when the corporate intranet deployment is considered.

The consensus that emerged was that Heimes's sslcustomize solution is viable for Python 3 users, but it soon became apparent that there was a push to backport the fix to Python 2 as well. As Coghlan [pointed out](http://lwn.net/Articles/611484/), Python 2's status as a maintenance release mandates an even better migration strategy: introducing a break in backward-compatibility is a far worse move in an maintenance branch (particularly one with a large installed base).

Lefkowitz again [advocated](http://lwn.net/Articles/611485/) an immediate switch-over, saying "this is not a break in backwards compatibility, it's a bug fix. Yes, systems might break, but that breakage represents an increase in security which may well be operationally important." Antoine Pitrou [disagreed](http://lwn.net/Articles/611486/) with that notion, responding: "saying it doesn't make it magically true. Besides, it can perfectly well be a bug fix \*as well as\* a break in backwards compatibility. Which is why we sometimes choose to fix bugs only in the feature development branch."

There were, of course, several parties in each camp, which led to a rather lengthy discussion. Ultimately, though, the backport camp managed to convince Guido van Rossum, effectively ending the debate. Van Rossum [decided](http://lwn.net/Articles/611487/) that certificate validation by default should be backported to the next Python 2.7 release:

I don't want to start preaching security doom and gloom (the experts are doing enough of that :-), but the scale and sophistication of attacks (whether publicized or not) is constantly increasing, and routine maintenance checks on old software are just one of the small ways that we can help the internet become more secure. (And please let the PSF sysadmin team beef up \*.python.org -- sooner or later some forgotten part of our infrastructure \*will\* come under attack.)

Heimes [pointed out](http://lwn.net/Articles/611488/) that the *next* 2.7 release (which would be numbered 2.7.8) was not a viable option, since there would not be any way to support a workaround for self-signed and intranet certificates. A fix for that issue is underway, but it will not land until Python 2.7.9. Van Rossum [agreed](http://lwn.net/Articles/611489/), but reiterated his decision that the validate-by-default behavior should be ported to the next possible Python 2.7 update—that update will simply be one or two releases later.

As it stands now, then, the 3.4.2 release will issue warnings when an SSL/TLS certificate does not validate. In addition, 3.4.2 will add a simple workaround ([described](http://lwn.net/Articles/611490/) by Coghlan) for code needing to bypass certificate validation: urllib.request.urlopen() will support a new "SSL context" parameter with which developers can opt-in to the old behavior on a per-call basis. The exact details are still under discussion, naturally, but such a bypass operation might look like:

urllib.request.urlopen(context=ssl.\_create\_unverified\_context())

as [one example](http://lwn.net/Articles/611510/) described it. An officially recommended monkeypatching fix has also been discussed, which would be used to tweak application behavior. 3.5 will introduce the more full-featured sslcustomize module. "SSL Context" will also be added to Python 2.7.9 and, assuming that goes well, certificate validation could be switched on by default as early as 2.7.10.

In all likelihood, there will be users (and perhaps developers) whose first encounter with any of this work will be when they have a previously working Python application unexpectedly fail with a scary-looking exception about untrusted security certificates. Some of those exceptions will be fixable with workarounds, albeit frustrating ones. But the point of the whole endeavor is that there will be other exceptions, too: users who have been exposed to bad or even malicious SSL/TLS servers and simply never heard about it in the past.

([Log in](https://lwn.net/login?target=/Articles/611243/) to post comments)

**Python decides for certificate validation**

Posted Sep 11, 2014 5:35 UTC (Thu) by **Cyberax** (**✭ supporter ✭**, #52523) [[Link](http://lwn.net/Articles/611530/)]

The Python developers should pull their head out of their asses and do a real Python 2.8 release. Breaking back compat in a minor release is the worst thing they can do.

**Python decides for certificate validation**

Posted Sep 11, 2014 16:52 UTC (Thu) by **josh** (subscriber, #17465) [[Link](http://lwn.net/Articles/611611/)]

Python's 2.7.\* seems to have become the equivalent of the Linux kernel's 2.6.\*.

**Python decides for certificate validation**

Posted Sep 11, 2014 14:43 UTC (Thu) by **mathstuf** (subscriber, #69389) [[Link](http://lwn.net/Articles/611600/)]

Is there an API for providing your own trust root? If I know my target, trusting anyone for the root certificate is still non-optimal.

**Python decides for certificate validation**

Posted Sep 11, 2014 17:41 UTC (Thu) by **leoluk** (subscriber, #97665) [[Link](http://lwn.net/Articles/611616/)]

The python-requests library supports SSL certificate validation:

[http://docs.python-requests.org/en/latest/user/advanced/#...](http://docs.python-requests.org/en/latest/user/advanced/#ssl-cert-verification)

Nice to hear that they'll implement verification in the standard library.

**Python decides for certificate validation**

Posted Sep 12, 2014 15:11 UTC (Fri) by **DavidS** (subscriber, #84675) [[Link](http://lwn.net/Articles/611740/)]

After having had the "opportunity" to have to review http APIs in python I'm really wondering why anyone would prefer the abomination that is urllib over python-requests.

In the same time I spent checking how I could (not) solve my TLS use case with urllib I had it already implemented and running with requests. Never looked back.

**small correction**

Posted Sep 11, 2014 21:08 UTC (Thu) by **gutworth** (subscriber, #96497) [[Link](http://lwn.net/Articles/611663/)]

Actually the next release of Python 2.7 will be 2.7.9. 2.7.8 is already released.

**Python decides for certificate validation**

Posted Sep 18, 2014 6:18 UTC (Thu) by **thedevil** (guest, #32913) [[Link](http://lwn.net/Articles/612344/)]

What about protocols other than HTTP on Python 2.7?

I am thinking of smtplib.SMTP\_SSL and imaplib.IMAP4\_SSL in particular.  
Or have we really reached the point where "networking" == "HTTP"? If  
so, I quit.

**Python decides for certificate validation**

Posted Sep 18, 2014 17:47 UTC (Thu) by **raven667** (subscriber, #5198) [[Link](http://lwn.net/Articles/612459/)]

We passed that point a while ago, when was the last time you saw a new major internet-scale protocol that wasn't run over HTTP(S)?

**Python decides for certificate validation**

Posted Sep 18, 2014 23:43 UTC (Thu) by **ssokolow** (guest, #94568) [[Link](http://lwn.net/Articles/612497/)]

\*nod\* Far too many over-paranoid corporate firewalls to implement a new protocol.

It's bad enough that many of them break cross-domain web fonts by being so hyper-paranoid about HTTP headers that they strip CORS headers.

**Python decides for certificate validation**

Posted Sep 19, 2014 3:35 UTC (Fri) by **mathstuf** (subscriber, #69389) [[Link](http://lwn.net/Articles/612517/)]

Why is breaking web fonts such a bad thing?</snark> I'm happy NoScript blocks them too :) . Though RequestPolicy gets the cross-domain ones first.

**Python decides for certificate validation**

Posted Sep 19, 2014 3:42 UTC (Fri) by **thedevil** (guest, #32913) [[Link](http://lwn.net/Articles/612519/)]

I prefer to edit ~/.fonts.conf and substitute my default font but it  
does mean I have to do it again for every new one. One of these days  
I'll follow your example :-P

**Python decides for certificate validation**

Posted Sep 22, 2014 3:42 UTC (Mon) by **ssokolow** (guest, #94568) [[Link](http://lwn.net/Articles/612751/)]

Aside from getting a screen full of unreadable toolbar buttons on sites like GitHub which use icon fonts?

Some sites do REALLY stupid things. The worst being Scribd, which prevents downloading documents without creating an account by applying a substitution cipher to the text beyond the Google-visible preview snippet and then using a dynamically generated web font to unscramble the glyph mappings for display.

**Python decides for certificate validation**

Posted Sep 22, 2014 3:47 UTC (Mon) by **mathstuf** (subscriber, #69389) [[Link](http://lwn.net/Articles/612752/)]

The only problems I've seen on github are the icons in the list and something weird with the avatars. The latter is probably due to RequestPolicy and/or some JS not running. Luckily, I have little need to use scribd myself. Though I have run into PDFs like that before. The tr command is your friend :) .

I've gotten used to sites just barfing text everywhere without their CSS's CDN access, so garbled icons isn't that bad.

**Python decides for certificate validation**

Posted Sep 22, 2014 10:51 UTC (Mon) by **ssokolow** (guest, #94568) [[Link](http://lwn.net/Articles/612762/)]

I have no problem with using tr. In fact, it wouldn't be too difficult to do frequency and dictionary analysis to automate the whole process... that doesn't change the fact that I don't really have much against web fonts but blocking them is a ton of hassle.

(Maybe it's just that RequestPolicy is a hair too far on the side of hassle for me so I stick to stuff like RefControl and NoScript with some custom ABE rules to block 3rd-party requests on sites I actually visit like Twitter.)

**Python decides for certificate validation**

Posted Sep 19, 2014 3:36 UTC (Fri) by **thedevil** (guest, #32913) [[Link](http://lwn.net/Articles/612515/)]

Well, I think you misunderstood me. It was probably due to my joking  
tone for which I apologize; my point was serious. I wasn't asking about  
new protocols but about existing code using protocols like IMAP and SMTP  
over SSL. That code will break with the new change just like HTTP will,  
but I see no mention of a fix equivalent to urlopen(context=...) for  
them. So what's the plan, just abandon that code or force it to be  
rewritten in Python 3 (where the new sslcustomize module is or will be  
available) ?

**Python decides for certificate validation**

Posted Sep 21, 2014 11:59 UTC (Sun) by **intgr** (subscriber, #39733) [[Link](http://lwn.net/Articles/612714/)]

> existing code using protocols like IMAP and SMTP over SSL. That code will break with the new change just like HTTP will

I assume those protocols will continue to ignore certificate validation like in previous versions. Why do you think they will break?

**Python decides for certificate validation**

Posted Sep 23, 2014 17:19 UTC (Tue) by **thedevil** (guest, #32913) [[Link](http://lwn.net/Articles/612889/)]

I thought they would break because the change (make validating be the  
default) is in the underlying SSL code, on which all the individual  
protocol modules depend. Was this a misunderstanding?

**Python decides for certificate validation**

Posted Sep 19, 2014 2:33 UTC (Fri) by **mathstuf** (subscriber, #69389) [[Link](http://lwn.net/Articles/612410/)]

But backwards compatibility![1] I don't see it as a break anyways since those functions may have always failed, but just never did. Anyone not checking for failures already is playing with fire. Or are we just going to continue to pretend CPython is the only Python implementation?

[1]Ignore the Python3 problems though, please.