proposal: os/exec: make LookPath not look in dot implicitly on Windows #38736

Closed dholmesdell opened this issue on Apr 28, 2020 · 31 comments

Labels	OS-Windows Proposal Security
Projects	Proposals (old)
Milestone	Proposal

dholmesdell commented on Apr 28, 2020

What version of Go are you using (`go version`)?

```
$ go version
go version go1.14.1 windows/amd64
```

Originally noticed with Go 1.8 though.

Does this issue reproduce with the latest release?

Yes

What operating system and processor architecture are you using (`go env`)?

▶ `go env` Output

What did you do?

Copy “C:\Windows\System32\whoami.exe” to the following program’s directory as “systeminfo.exe” and then run the program:

```
package main

import (
	"fmt"
	"os"
	"os/exec"
)

func main() {
	os.Setenv("PATH", `C:\Windows\System32`)

	cmd := exec.Command("systeminfo.exe")
	out, err := cmd.CombinedOutput()
	if err != nil {
		panic(err)
	}
	fmt.Println(string(out))
}
```

What did you expect to see?

The output of C:\Windows\System32\systeminfo.exe

What did you see instead?

The output of ./systeminfo.exe (my username, since it is a copy of whoami.exe)

If the renamed copy of systeminfo.exe is removed from the test program’s directory, then the output of “C:\Windows\System32\systeminfo.exe” is displayed as expected.

Analysis

os/exec/lp_windows.go contains the following:

```
func LookPath(file string) (string, error) {
…
	if strings.ContainsAny(file, `:\/`) {
		if f, err := findExecutable(file, exts); err == nil {
			return f, nil
		} else {
			return "", &Error{file, err}
		}
	}
	if f, err := findExecutable(filepath.Join(".", file), exts); err == nil {
		return f, nil
	}
```

If the value of ‘file’ is an absolute or relative path, a result is returned. The concern is with a value which is only a name, such as “systeminfo.exe”. One would expect this to search the list of paths found in the PATH environment variable, but before doing so the code explicitly searches the current working directory (“.”). There does not appear to be any means provided to disable this behavior and search only PATH.

I would guess that the intent was to mimic the behavior of the cmd.exe command shell, which searches the current directory first even if it is not specified in PATH. By comparison, the documentation for the Windows CreateProcess API indicates that it does not search PATH at all, but will use the current directory to complete a partial path. (The SearchPath API offers an alternative, though also flawed, option to search the current directory last.)

The problem is that it is not possible to use exec.LookPath, and thus exec.Command, to search the system PATH without searching the current directory. Thus even if diligence is taken to have the program set a secure PATH value, the programmer must be aware of this behavior and avoid using these standard library functions. The documentation of exec.LookPath does not mention the current directory, stating only that it searches "the directories named by the PATH environment variable."

Suggestions

My preferred recommendation would be to remove the explicit search of "." (the second if-clause shown above), in order to provide the best level of security and comply with the documentation. A programmer can add "." to the PATH environment variable value if the behavior is desired, as one would do in a linux/unix program.

If the resulting change in Go behavior/compatibility is not desirable, a workaround could be for exec.LookPath to reference the NoDefaultCurrentDirectoryInExePath environment variable and avoid searching "." if it is set. This is a workaround which Microsoft apparently added in Vista to disable the behavior in cmd.exe.

cc **@FiloSottile**

👍 28 🚀 2 👀 2

as commented on Apr 28, 2020 Contributor

Disabling this will likely break dockerized windows containers relying on the behavior. Amusingly, creating `ping.bat` will make `cmd.exe` run it before the system's ping command, as windows has an extension search list too.

But, if you can copy a file in the same directory as the executable, you can probably rename the original executable (even while its process is running) and replace it with a program that deletes all files on your filesystem next time its executed. The security community treats the filesystem as the wild wild west, but the only way to mitigate hijacking attacks is to properly secure the filesystem in the first place.

Hence, I think Go should not try to solve this problem if LookPath is designed to be compatible with Windows behavior (but NoDefaultCurrentDirectoryInExePath might be a good idea, depending on how it works with modern Windows versions).

👍 7

ALTree added **NeedsInvestigation** **OS-Windows** labels on Apr 29, 2020

FiloSottile added **NeedsDecision** Security labels on Apr 29, 2020

gopherbot removed the **NeedsInvestigation** label on Apr 29, 2020

FiloSottile changed the title ~~Unsecure path search in os/exec~~ **os/exec: LookPath implicitly searches current directory** on Apr 29, 2020

ianlancetaylor added this to the **Backlog** milestone on Apr 29, 2020

aral commented on Jul 23, 2020

I'd consider this desirable behaviour, if consistent cross-platform.

e.g., Use case:

mkcert uses the following call to find certutil if it is installed:

```
exec.LookPath("certutil")
```

I'm going to be bundling pre-built binaries of certutil with future version of Auto Encrypt Localhost. Afaics, with the current behaviour, all I have to do is place the certutil binary and the nss dynamic libraries in the same folder as the mkcert binary. If I had to specify the folder explicitly, I'd most likely have to maintain a fork of mkcert.

mislav commented on Oct 27, 2020 • edited

> I would guess that the intent was to mimic the behavior of the cmd.exe command shell, which searches the current directory first even if it is not specified in PATH.

@dholmesdell You are right that this was the motivation for the feature: `2a876be`

However, I would argue that this was a bad call, as it was made 9 years ago with little discussion about pros & cons. I don't see why the shell feature of `cmd.exe` to execute binaries or batch scripts from the local directory was worth propagating in Go, especially since that feature wasn't kept in PowerShell, which is now de-facto standard shell in Windows.

This behavior of LookPath on Windows absolutely goes against the documentation of the LookPath method, and I find it surprising at best and a security vulnerability at worst.

/cc **@alexbrainman** **@rsc** for comments

rsc commented on Oct 28, 2020 Contributor

The behavior we chose matched the behavior of the default Windows shell at the time.
It seems unlikely we could change it now without breaking many Windows programs.

mislav commented on Oct 29, 2020

@rsc I understand; thank you for chiming in.

Would you be open to a documentation change that points out this behavior on Windows?

dawidgolunski commented on Oct 29, 2020

Hi. As for executing new processes, I thought the Windows standard order (other than cmd that mimics DOS) of searching directories was the one used by the CreateProcessA in win32 API which only searches the current directory as the second option, only if the command was not found in the directory from which the application loaded . See:
https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
I think this is what python's subprocess.Pipe() method mimics on Windows for example.

I personally saw it as a vuln and even requested a CVE for this issue but it seems like the agreement here is that it can't be changed in Go itself and programmers must take care of this individually.
What I think would be helpful in this case, in addition to clarifying the documentation, is adding a separate function to GO that only searches the directories declared in PATH without including cwd/. E.g. StrictPath() in the exec package?
This way existing apps won't be broken and programmers can use this function without re-inventing the wheel when needed.
Just an idea. Would this work ? Could this be added?

ianlancetaylor commented on Oct 29, 2020 Contributor

@mislav Documentation changes are fine.

@dawidgolunski As far as I can tell `exec.LookPath` on Windows does not search the directory from which the application loaded at all. It searches the current directory, then the directories on the environment variable `path`.

I could imagine a `exec.LookPathStrict` proposal. Do you want to write one (https://golang.org/s/proposal)? It could perhaps ignore `.` or any relative directory on the `PATH` / `path`.

👍 2

dawidgolunski commented on Oct 29, 2020

> **@dawidgolunski** As far as I can tell `exec.LookPath` on Windows does not search the directory from which the application loaded at all. It searches the current directory, then the directories on the environment variable `path`.

@ianlancetaylor That's right, `LookPath` function does not do that. I was referring to the `CreateProcessA` function (`processthreadsapi.h`) from the Windows API. The Microsoft article I sent in the previous message defines the executable file search order (if the full path was not provided) as follows:

1. The directory from which the application loaded.
2. The current directory for the parent process.
...
6. The directories that are listed in the PATH environment variable.

The step **1** adds a layer of protection as usually installed applications are loaded from directories owned by admin/SYSTEM user which prevents modification as regular users can't add files into them.

> I could imagine a exec.LookPathStrict proposal. Do you want to write one (https://golang.org/s/proposal)?

Good name I think. As for the proposal, I'll try to have a go at it when I get a spare minute.

alexbrainman commented on Oct 30, 2020 Member

@mislav

I refer you to what **@rsc** said. We made decision based on our knowledge at the time. There was no PowerShell back then. I still do not use PowerShell. Hardly any Windows users use PowerShell.

I also refer you to what **@dawidgolunski** said. CreateProcess is a good authority here. And CreateProcess puts PATH at number 6 - pretty low.

I am still happy with the decision we made.

I agree that LookPath documentation does not mention current directory. We should fix that.

I am not so keen on adding new LookPath function. How would you explain to people which of two functions to use?

I don't see running executable from current directory as security threat. CreateProcess does that too, so that is OK with me.

Alex

dholmesdell commented on Nov 2, 2020 Author

Should it matter what each platform does in its shell or library? As LookPath is Go's own library function and not a syscall passthrough, one could reasonably expect it to have the same behavior on any platform.

My main concern is that it is not possible to implement secure and consistent behavior in Go programs without reimplementing part of the library. There might be an argument for exec.Command to behave similarly to CreateProcess. But since the behavior is actually implemented in exec.LookPath, one cannot use it to obtain an explicit path in order to avoid the "convenience" behavior in exec.Command. There seems to be no option to avoid searching "." without reimplementing one's own version of LookPath.

In other words, from my view having an option to use LookPath (or another similar function, though that seems slightly awkward) to search only PATH (without including ".") so that I can use that result with exec.Command myself would satisfy the need.

alexbrainman commented on Nov 4, 2020 Member

> Should it matter what each platform does in its shell or library? As LookPath is Go's own library function and not a syscall passthrough, one could reasonably expect it to have the same behavior on any platform.

I don't think it is reasonable to have the same behaviour on different platforms. os.LookPath should do what platform users expect, not some "rule invented by Go authors". Something that is reasonable on one platform could be bug or security fault on another platform.

Alex

👍 3 👎 2

gtardif mentioned this issue on Nov 5, 2020

Windows: do not resolve files in current working dir if CWD is not explicitly in PATH. docker/compose-cli#884

Merged

dawidgolunski mentioned this issue on Nov 6, 2020

proposal: os/exec: add LookPathAbs that refuses to return relative paths #42420

Closed

dawidgolunski commented on Nov 6, 2020

@ianlancetaylor I've written the proposal on
#42420
From what I understood, issues can be assigned a Proposal label. Please let me know if this is sufficient.

❤️ 3

ianlancetaylor commented on Nov 6, 2020 Contributor

Thanks, I converted #42420 into a proposal following the guidelines at https://golang.org/s/proposal.

🚀 4

mislav commented on Nov 11, 2020 • edited

> I am not so keen on adding new LookPath function. How would you explain to people which of two functions to use?

@alexbrainman Thank you for adding context and your thoughts.

I agree that adding a new `LookPath*` function could be generally confusing when the two are viewed side-by-side. However, I also find it entirely reasonable that someone would want a utility to only search in PATH and *not* the current directory.

I have created the https://github.com/cli/safeexec module to provide such a `LookPath` implementation for Windows, and I've basically had to copy-paste a non-trivial amount of code from Go's standard library to preserve the parts of `LookPath` logic that I wanted to keep (iterating over `%PATH%`, respecting existing file extension, trying extensions from `%PATHEXT%`) just to avoid the 2 lines of logic that we wanted to avoid.

I was forced to do this because an innocent-looking invocation of `exec.Command("git", args...)` in our GitHub CLI tool was reported as a potential security vulnerability: GHSA-fqfh-778m-2v32

It looks like the `docker-compose` CLI tool also wanted to avoid the security issue, and they had to copy-paste the same code from stdlib into a new package: https://github.com/docker/compose-cli/pull/884/files#diff-3f9e9d4a31ba2c4d9ae604425c9cdca77b4e61f39289fdd35a02ca422c0a6a41

So while I can understand your stance on this and the fact that Go wants to keep backwards compatibility whenever possible, Go programs executing on Windows currently do not have any way of easily running a command found in PATH while also respecting PATHEXT. Our options are:

- Reach for the `exec.LookPath` function that advertises this functionality, and thus unintentionally expose themselves to unwanted behavior of scanning the current directory;
- Implement the logic manually, risking introducing bugs re: extension handling;
- Copy-paste the code from stdlib into an internal module;
- Use a 3rd-party module like https://github.com/cli/safeexec.

None of these options feel great to me. I believe that Go's standard library should make a PATH-based command dispatch accessible without any side-effect.

@ianlancetaylor Thank you for submitting the proposal! ❤️

👍 9

This comment has been minimized. Sign in to view

alexbrainman commented on Nov 20, 2020 Member

> I agree that adding a new `LookPath*` function could be generally confusing when the two are viewed side-by-side. However, I also find it entirely reasonable that someone would want a utility to only search in PATH and *not* the current directory.

Agreed.

> I have created the https://github.com/cli/safeexec module to provide such a `LookPath` implementation for Windows,

Wonderful. Looks good.

> ... and I've basically had to copy-paste a non-trivial amount of code from Go's standard library to preserve the parts of `LookPath` logic that I wanted to keep (iterating over `%PATH%`, respecting existing file extension, trying extensions from `%PATHEXT%`) just to avoid the 2 lines of logic that we wanted to avoid.

I think it is fair price to pay for you to have modified version of `exec.LookPath`. The alternative would be confused users trying to decide which of two `exec.LookPath*` to use. Users who need your version of `LookPath` will find it at `github.com/cli/safeexec`.

> I was forced to do this because an innocent-looking invocation of `exec.Command("git", args...)` in our GitHub CLI tool was reported as a potential security vulnerability: GHSA-fqfh-778m-2v32

I don't doubt standard `exec.LookPath` is not suitable for your project. But your scenario is special. Windows binaries are supposed to be installed in a predefined locations (https://en.wikipedia.org/wiki/Program_Files), and not uncontrollably downloaded into current directory. I suspect that Git authors never considered Windows when designing their system.

> It looks like the `docker-compose` CLI tool also wanted to avoid the security issue, and they had to copy-paste the same code from stdlib into a new package: https://github.com/docker/compose-cli/pull/884/files#diff-3f9e9d4a31ba2c4d9ae604425c9cdca77b4e61f39289fdd35a02ca422c0a6a41

I did not look at these links. But I agree, they are similar to your project. They should use the package you created.

> None of these options feel great to me.

I disagree. Users who need this functionality should use https://github.com/cli/safeexec. Perhaps over time, Windows will change in this regard, and `exec.LookPath` will change too. But today most Windows users should use standard Windows rules.

Alex

👍 2

iamheartypareja commented on Nov 22, 2020

> > I agree that adding a new `LookPath*` function could be generally confusing when the two are viewed side-by-side. However, I also find it entirely reasonable that someone would want a utility to only search in PATH and *not* the current directory.
>
> Agreed.
>
> > I have created the https://github.com/cli/safeexec module to provide such a `LookPath` implementation for Windows,
>
> Wonderful. Looks good.
>
> > ... and I've basically had to copy-paste a non-trivial amount of code from Go's standard library to preserve the parts of `LookPath` logic that I wanted to keep (iterating over `%PATH%`, respecting existing file extension, trying extensions from `%PATHEXT%`) just to avoid the 2 lines of logic that we wanted to avoid.
>
> I think it is fair price to pay for you to have modified version of `exec.LookPath`. The alternative would be confused users trying to decide which of two `exec.LookPath*` to use. Users who need your version of `LookPath` will find it at `github.com/cli/safeexec`.
>
> > I was forced to do this because an innocent-looking invocation of `exec.Command("git", args...)` in our GitHub CLI tool was reported as a potential security vulnerability: GHSA-fqfh-778m-2v32
>
> I don't doubt standard `exec.LookPath` is not suitable for your project. But your scenario is special. Windows binaries are supposed to be installed in a predefined locations (https://en.wikipedia.org/wiki/Program_Files), and not uncontrollably downloaded into current directory. I suspect that Git authors never considered Windows when designing their system.
>
> > It looks like the `docker-compose` CLI tool also wanted to avoid the security issue, and they had to copy-paste the same code from stdlib into a new package: https://github.com/docker/compose-cli/pull/884/files#diff-3f9e9d4a31ba2c4d9ae604425c9cdca77b4e61f39289fdd35a02ca422c0a6a41
>
> I did not look at these links. But I agree, they are similar to your project. They should use the package you created.
>
> > None of these options feel great to me.
>
> I disagree. Users who need this functionality should use https://github.com/cli/safeexec. Perhaps over time, Windows will change in this regard, and `exec.LookPath` will change too. But today most Windows users should use standard Windows rules.
>
> Alex

#38736 (comment)

profclems added a commit to profclems/glab that referenced this issue on Nov 26, 2020

`Ensure only $PATH is searched when switching to external commands` … `51c3e7d`

profclems mentioned this issue on Nov 26, 2020

Ensure only $PATH is searched when switching to external commands profclems/glab#332

Merged

rsc mentioned this issue on Dec 2, 2020

proposal: os/exec: use LookPathAbs by default #42950

Closed

rsc changed the title ~~os/exec: LookPath implicitly searches current directory~~ **proposal: os/exec: make LookPath not look in dot implicitly on Windows** on Dec 2, 2020

13 hidden items
Load more...

zimbatm added a commit to zimbatm/direnv that referenced this issue on Dec 23, 2020

`lp: stop looking in . on Windows` … `1dcb094`

zimbatm added a commit to zimbatm/direnv that referenced this issue on Dec 23, 2020

`lp: stop looking in . on Windows` … `ec2eb71`

zimbatm mentioned this issue on Dec 27, 2020

interp: LookPathDir mvdan/sh#646

Merged

lnu mentioned this issue on Dec 30, 2020

fix/perf: don't call language detection command in enabled() JanDeDobbeleer/oh-my-posh#285

Merged

4 tasks

rsc mentioned this issue on Jan 19, 2021

os/exec: return error when PATH lookup would use current directory #43724

Closed

rsc commented on Jan 20, 2021 Contributor

For anyone following this issue, I think we've figured out a way to avoid surprising or confusing breakages but still correct the behavior. See #43724.

👍 1

KalleOlaviNiemita... commented on Jan 24, 2021

> If the resulting change in Go behavior/compatibility is not desirable, a workaround could be for exec.LookPath to reference the NoDefaultCurrentDirectoryInExePath environment variable and avoid searching "." if it is set.

If you decide to use the environment variable, then Microsoft documentation recommends calling NeedCurrentDirectoryForExePathW instead of reading the environment variable directly.

👍 1

dolmen mentioned this issue on Jan 27, 2021

os/exec: on Windows use NeedCurrentDirectoryForExePathW for LookPath behavior #43947

Closed

rsc commented on Jan 27, 2021 Contributor

Given #43724 it seems like we should simplify down and decline this proposal.

👍 2

rsc commented on Jan 27, 2021 Contributor

Based on the discussion above, this proposal seems like a likely decline.
— rsc for the proposal review group

rsc moved this from **Active** to **Likely Decline** in **Proposals (old)** on Jan 27, 2021

rsc added the Proposal-FinalCommentPeriod label on Jan 27, 2021

rsc commented on Feb 3, 2021 Contributor

No change in consensus, so declined.
— rsc for the proposal review group

rsc moved this from **Likely Decline** to **Declined** in **Proposals (old)** on Feb 3, 2021

rsc removed the Proposal-FinalCommentPeriod label on Feb 3, 2021

rsc closed this as completed on Feb 3, 2021

This comment has been minimized. Sign in to view

netbsd-srcmastr pushed a commit to NetBSD/pkgsrc that referenced this issue on Feb 9, 2021

`hugo: Update to 0.80.0` … bb9710d

jperkin pushed a commit to TritonDataCenter/pkgsrc that referenced this issue on May 12, 2021

`hugo: Update to 0.80.0` … 3f31df9

mrcool1661 mentioned this issue on May 19, 2021

@mislav Documentation changes are fine. #46262

Closed

BurntSushi mentioned this issue on May 29, 2021

[Security] Vulnerability report BurntSushi/ripgrep#1773

Closed

golang deleted a comment on Feb 14

This comment was marked as spam. Sign in to view

This comment was marked as spam. Sign in to view

This comment was marked as spam. Sign in to view

rafasha123 mentioned this issue on Jun 19

cmd/go: unexpected output with go doc std #53446

Open

This comment was marked as spam. Sign in to view

bcmills commented on Jul 8 Member

Proposal #43947 was accepted and implemented for this instead.

bcmills closed this as not planned on Jul 8

golang locked as **resolved** and limited conversation to collaborators on Jul 8

Assignees

No one assigned

Labels

OS-Windows Proposal Security

Projects

No open projects

1 closed project

Milestone

Proposal

Development

No branches or pull requests

40 participants

and others