intro

why do we need it

we want potentially multiple implementations for example shellcode loader different on mac, nix, windows

for now we will only implement windows, but create stubs for the others

another practical reason sometimes

in my case - i work on mac os

so if i only implement windows, without an interface, i have an issue

the windows-specific shellcode loader doer logic will error out, so i need to use build tags `//go:build windows`

but then, if i do that it essentially "hides" the code from the rest of the application, meaning then when i call these functions in other parts, like orchestrator, they error out since they cannot find them

interface is solution, i use build tags for windows, which "hides" them, but then in orchestrator i don't call the windows functions directly - i call interface - we already saw this!

custom type for ShellcodeResult (models package)

First thing, as shown in previous lesson we have this high-level "generic" results type we will send back to the server

```go
type AgentTaskResult struct {
    JobID         string          `json:"job_id"`
    Success       bool            `json:"success"`
    CommandResult json.RawMessage `json:"command_result,omitempty"`
    Error         error           `json:"error,omitempty"`
}
```

Just as we have a high-level command instruction type that had to have its own field to handle specific arguments related to each command, here we have

`CommandResult`

Why do we need this again? Well different commands will ahve different output. If we run a shell command, say "whoami", well we want results here. If we downlaod a file, we want the base64 data there etc.

This field since it is json.RawMessage is in a sense amorphous - we can place any custom struct inside of it

And so indeed we want one for our shellcode command. This one is fortunately very simple, since nothing really needs to be returned, no output (whoami), or data (download). All we really need is some message - some success message to say it worked etc.

So let's add this to types.go

```go
type ShellcodeResult struct {
    Message string `json:"message"`
}
```

Great now we can create our interface for doer, right now I'll create 2 files

- The interface file where interface is defined
- A stub implementation for mac os

We'll createa a new package at - ./internal/shellcode

interface_shellcode.go

```go
package shellcode

import "workshop3_dev/internals/models"

type CommandShellcode interface {
    DoShellcode(dllBytes []byte, exportName string)
(models.ShellcodeResult, error)
}
```

HERE PLEASE EXPLAIN CODE GIVE A MINI-LESSON

doer_shellcode_mac.go

```go
//go:build darwin

package shellcode

import (
	"fmt"
	"workshop3_dev/internals/models"
)

// macShellcode implements the CommandShellcode interface for Darwin/MacOS.

type macShellcode struct{}

// New is the constructor for our Mac-specific Shellcode command
func New() CommandShellcode {
	return &macShellcode{}
}

func (ms *macShellcode) DoShellcode(dllBytes []byte, exportName string) (models.ShellcodeResult, error) {
	fmt.Println("|❗ SHELLCODE DOER MACOS| This feature has not yet been implemented for MacOS.")

	result := models.ShellcodeResult{
		Message: "FAILURE",
	}
	return result, nil
}
```

Explain this, we will see same essential pattern in windows, difference there really is DoShellcode will be much more complicated - here really just a stub

We need a struct, something to fulfill interaface - macShellcode

New() is just the constructor to instantiate it - we saw that we call this in orchestrator

Then that allows us to call method DoShellcode on it

Again here we can see DoShellcode is just a stub.

test

Since I am on a MacOS, and I did implement a MacOS doer here, we can actually test.

Obvs if you are on Windows or Linux, this will not work, but at least revew output here.

We run server, run client. We run agent

```
❯ go run ./cmd/agent
2025/11/07 08:44:15 Starting Agent Run Loop
2025/11/07 08:44:15 Delay: 5s, Jitter: 50%
2025/11/07 08:44:15 Job received from Server
-> Command: shellcode
-> JobID: job_840709
2025/11/07 08:44:15 AGENT IS NOW PROCESSING COMMAND shellcode with ID job_840709
2025/11/07 08:44:15 |✅ SHELLCODE ORCHESTRATOR| Task ID: job_840709. Executing Shellcode, Export Function: LaunchCalc, ShellcodeLen(b64)=148660
|❗ SHELLCODE DOER MACOS| This feature has not yet been implemented for MacOS.
2025/11/07 08:44:15 |👊 SHELLCODE SUCCESS| Shellcode execution initiated successfully for TaskID job_840709. Loader Message: FAILURE
2025/11/07 08:44:15 |AGENT TASK|-> Sending result for Task ID job_840709 (66 bytes)...
2025/11/07 08:44:15 |RETURN RESULTS|-> Sending 66 bytes of results via POST to https://0.0.0.0:8443/results
2025/11/07 08:44:15 💥 SUCCESSFULLY SENT FINAL RESULTS BACK TO SERVER.
2025/11/07 08:44:15 |AGENT TASK|-> Successfully sent result for Task ID job_840709.
2025/11/07 08:44:15 Sleeping for 2.714526513s
^C2025/11/07 08:44:17 Shutting down client...
```

We can see we get earning it has not yet been plemented - as we expect

Note results are being sent back, remember results are sent back whether it succeeded or failed.

But since there is no /results endpoint yet on our server, we won't yet display this - that will come later.

CONCLUSION

Great we have basic interface in place now, now we can implement our windows doer