[P] | Prelude preludesecurity.com

Learn the basic syntax of a Verified Security Test

01 Endpoint.go

All security tests import the Endpoint module, which contains a collection of common functions for engaging with an endpoint. Functions include the ability to read and write files, locate installed applications or run shell commands.

```
14
15    Endpoint "github.com/preludeorg/test/endpoint"
16 )
17
```

02 Build flags

In Go, we may create build flags to make compilation easier. They act as build restrictions, ensuring that the code is only built on the designated operating system.

```
1  //go:build windows
2  // +build windows
```

03 Starting a test

Every test starts in the main function, where Endpoint.Start(..) is passed the test function to run - and optionally a clean up function to reverse the effects of the test. The Start function sets a 10 second timer to ensure all tests finish within this time. If a test exceeds 10s it will be stopped and a time out exit code will be sent to Detect.

```
47 func main() {
48     Endpoint.Start(test)
49 }
```

04 Embedded malware

Go has a unique keyword called "embed" that allows you to include an arbitrary file inside the compiled Go file. The file contents are available in a []byte variable during the code's runtime (called "malicious" in the example here).

```
18 //go:embed mimikatz.exe
19 var malicious []byte
```

05 Log statements

Tests can contain print statements that write to standard out or standard error. These statements are helpful for debugging but will also appear in the Detect log file (prelude.log) if the probe used an installer.

```
26  println("[+] Extracting file for quarantine test")
27  println("[+] Pausing for 3 seconds to gauge defensive reaction")
28  if Endpoint.Quarantined("mimikatz.exe", malicious) {
29          println("[+] Malicious file was caught!")
30          Endpoint.Stop(105)
31          return
32  }
33
34  println("[-] Malicious file was not caught")
35
36  command := supported[runtime.GOOS]
37  println("[+] Executing Mimikatz")
38  _, err := Endpoint.Shell(command)
39  if err != nil {
40          println("[+] Execution was prevented")
41          Endpoint.Stop(100)
42  }
43  println("[-] Mimikatz was not blocked")
```

06 Quarantine check

Some tests rely on the Quarantined function, which writes the embedded malware to disk, opens a file handle to it - in an attempt to trigger the defense, and then waits a few seconds before checking if the file was scooped up by any endpoint defense. If a test was quarantined it will exit with a 105, representing a quarantine event.

```
28  if Endpoint.Quarantined("mimikatz.exe", malicious) {
29          println("[+] Malicious file was caught!")
30          Endpoint.Stop(105)
31          return
```

07 Technique execution

Certain tests implement specific procedures in the event that the Quarantine test does not stop the test. Some of these procedures will be composed in pure Go language, while others will leverage the Shell function. This function accepts an array of commands and carries them out as a shell process. In the event that execution is blocked, the test will terminate, resulting in an exit code of 100.

08 UNPROTECTED!

If a test is not blocked at any stage of the attack, it will exit with a 101 code - meaning the endpoint failed the antivirus check.

```
43  println("[-] Mimikatz was not blocked")
44  Endpoint.Stop(101)
45 }
```