Prior to the training starting

- Ensure you have Go installed on your system
- A VM has been supplied with Go installed. Find it here: https://drive.google.com/file/d/12mYGwp9nutRYjz9VrIR3wfS2UuZFXgON/view?usp=sharing
  - **Don't worry about any updates to Ubuntu**
- A lot of the sample code, as well as solutions to exercises can be found here: https://github.com/Haydz/bsidestogo101
- Double check internet connection :)

Flow of the training

- Minimal troubleshooting at beginning of the day
- Flexible on delivery based on what students preferred
- Questions allowed at any time (depending on how many questions)
- Small breaks at the top of every hour
- Let us know if we are going too fast

Welcome to the training!!!!

GO AND ATT&CK

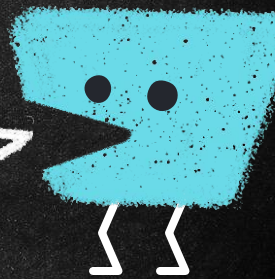
An introduction to programming with Go
(with advanced topics at the end)

What to expect

This course will introduce you to the basics of the Go programming language while expanding upon some of the reasons why it's gaining popularity among tech. startups and threat actors alike.

We'll cover the basics of:

- Structuring Go programs
- Data types and values
- Boolean logic
- Conditional logic
- Structs, methods, and interfaces
- Functions
- Pointers
- Error handling
- Channels
- Goroutines



Level of training

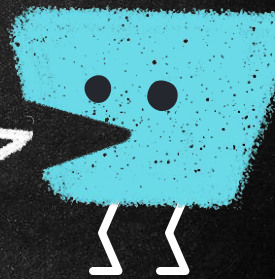
This is not a computer science course and will not cover every topic.

The aim is to get students comfortable with learning further on after this class.

Tyler is a developer, but Haydn is more of a hobby coder.

In Depth computer science, memory, or advanced questions will be for Tyler.

We are industry professionals volunteering our time to create and conduct the training. Please be nice :)



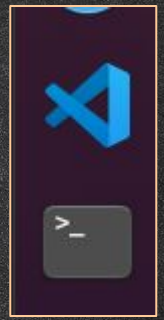
Setup

Prerequisites

- Go
- Your favourite text editor
- A network connection
- A can-do attitude

A Virtualbox VM has been supplied

If not already downloaded, please download from here:
https://drive.google.com/file/d/12mYGwp9nutRYjz9VrlR3wfS2UuZFXgON/view?usp=sharing



The Ubuntu VM includes:

- ➜ Go 1.16.6
- ➜ VSCode

For this training you will need the minimum:

- ➜ Go installed
- ➜ A text editor (e.g. Sublime Text)

VM credentials:
Username: bsidesto

Password: bsidesto

Installing Go

If you don't use the VM, please have Go installed.

For installing Go use:

- ➜ https://golang.org/doc/install

It should look something like:

bsides_ubuntu (Snapshot 1 - pre guest additions) [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Visual Studio Code

helloworl

File Edit Selection View Go Run Terminal Help

EXPLORER

CODE

pre_bsidesto

go.mod

helloworld.go

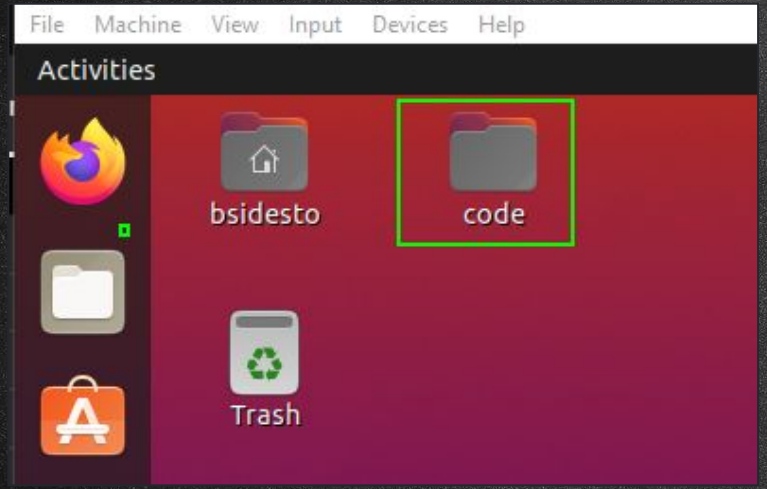
helloworld.go

pre_bsidesto > helloworld.go

```go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

VM Structure

There is a folder called **'code'** on the desktop.




```
bsidesto@bsidesto-go:~/Desktop/code/pre_bsidesto$ ls
go.mod  helloworld.go
```

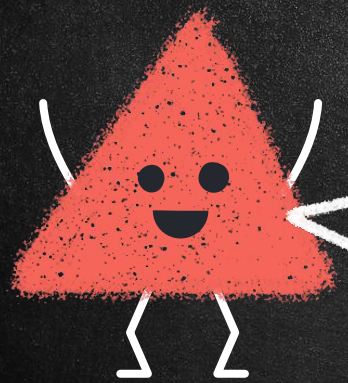
Github for reference

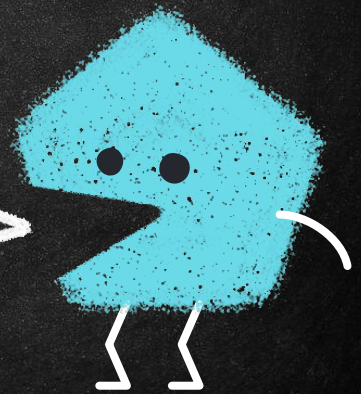
https://github.com/Haydz/bsidestogo101

Has Go code from the training. Including answers to some of the exercises.

Did the class download the VM?

Or install Go locally?





As a backup

The go playground is a web service that runs on golang.org's servers.

It can do most things, but only print to standard output and standard error. It cannot create any network communication.

https://play.golang.org/

Can be used up until the web requests section.

All about Go

What is GO?

- Go is a statically typed, compiled programming language that was initially developed at Google with the goal of making it easier for programmers to write fast, and efficient code while remaining as productive as possible.

- Go achieves this by abstracting away the technical complexity of memory management, garbage collection, and concurrent programming by managing memory for you, and providing a simple paradigm for carrying out multiple tasks at the same time.

- That is, Go handles the types of operations that are better left to machines, rather than software engineers.

https://golang.org/doc/faq#What_is_the_purpose_of_the_project

Why Go?

- Go is a fast, simple, “batteries-included” programming language that comes with everything that you need to get started

- Provides the readability and usability of loosely typed, interpreted languages like Python, or JavaScript while remaining strongly-typed
  - Value types can be explicitly declared or inferred at compile-time using Go’s type inference system
  - By omitting explicit type references and inferring types at compile-time via type inference, you end up less code
  - Less code = more readable code

https://golang.org/doc/faq#What_is_the_purpose_of_the_project

Why Go?

- Provides native support for concurrent programming using lightweight “goroutines”
- Goroutines are executed asynchronously within the scope of the calling process rather than across multiple threads or processes which avoids the need to perform context switching between threads and processes
- Go’s internal scheduler is responsible for scheduling work between goroutines and defers control to another thread if a given goroutine is blocked

https://golang.org/doc/faq#What_is_the_purpose_of_the_project

Why Go?

- Provides native support for concurrent programming using lightweight coroutines (“***goroutines***”)
- ***Goroutines*** are executed asynchronously within the scope of the calling process
- ***Channels*** are used to share information between ***goroutines***
- ***Channels*** may be ***buffered*** or ***unbuffered***
  - **Buffered channels**: multiple values can be written up to a set capacity - reads can be either *blocking* or *non-blocking*
  - **Unbuffered channels:** only one value can be written to the channel at a time - reads are always *blocking*
- Non-blocking reads from buffered channels are achieved by selecting from a given channel

“Don’t communicate by sharing memory; share memory by communicating” - Go proverb

https://golang.org/doc/faq#What_is_the_purpose_of_the_project

Go for Security??

- Becoming more popular in malware development
- Used in DevOps

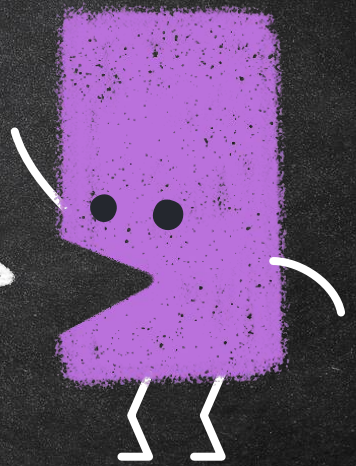
Advantages over Python?

- Lower detection rate against security software
- Compile code into all major OS, Linux, WIndows, Mac
- Can be compiled into a single file
- Performance is better

Basic Go

“

Lets get setup and run some Go code!



Content to cover

- Go Program Structure
- How to Run Go code
- Format of a Go File
- Go commands
- Everyone running their first file

Program Structure

A standalone Go program must have:

- A main function included in the main package
- A main function that is the entry point of execution

```go
package main

func main() {

}
```

Braces & no semicolons

Go uses brace braces for statement grouping, a syntax familiar to any language in the C family.

Semicolons (;) are injected automatically - compile time

```
func main() {

}
```

https://golang.org/doc/faq#semicolons

Standard library packages

Packages contain many fundamental building blocks to write modern software

Use an import statement to access the package.

Standard Go Library can be found:
https://pkg.go.dev/std

```go
package main

import "fmt"

func main() {

}
```

```go
import (
    "fmt"
    "math/rand"
  )
```

Directories

- archive
- bufio
- builtin
- bytes
- compress
  - bzip2
  - flate
  - gzip
  - lzw
  - zlib
- container
- context
- crypto
- database

https://pkg.go.dev/std
https://pkg.go.dev/fmt

Comments

Single Line Comments:

Use //

Block Comments:

Start with: /*

End with: *

```
// This is the single line comment
```

```
/* A multi line comment
Can be done here
*/
```

HelloWorld.go

- package main is defined
- Import fmt package from Go's standard library
- fmt.Println is used to print to standard output

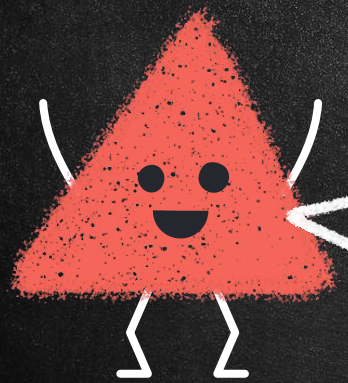
```go
package main
// This is the single line comment
import "fmt"
/* A multi line comment
Can be done here
*/
func main() {
    fmt.Println("Welcome to Go! - BsidesTo 2021 Go 101 training!")
}
```

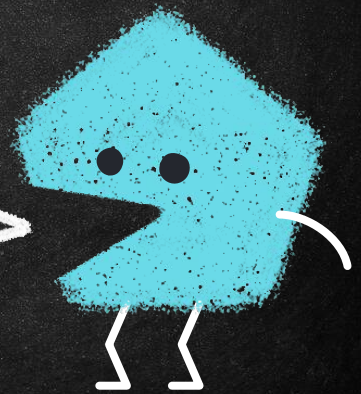
https://github.com/Haydz/bsidestogo101/blob/main/helloworld.go

We good to continue?

Is everyone setup?

Lets try to run a file





Go mod init

Modules refer to a grouping of related packages that are versioned to each other as one unit.

This is a requirement in Go, even when writing your own code.

To start, use the 'go mod init' command with a path, we used github just as a place holder and the main folders name. This saves us having to initialize each folder during the training.

```
go mod init github.com/bsidesto
```

A go.mod file will be created and contain something like:

```
PS C:\bsidesto_go> cat .\go.mod
module github.com/bsidesto

go 1.17
```

```
PS C:\bsidesto_go> pwd

Path
----
C:\bsidesto_go


PS C:\bsidesto_go> go list -m
github.com/bsidesto
PS C:\bsidesto_go>
```

Running a Go program

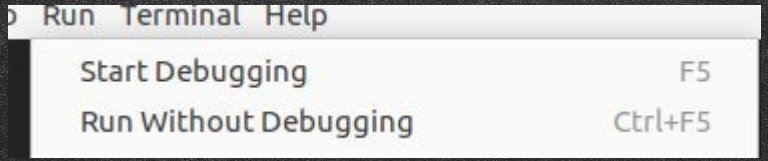
go run helloworld.go

```
PS C:\bsidesto_go> go run .\helloworld.go
Welcome to Go! - BsidesTo 2021 Go 101 training!
```

VSCode:

ctrl + f5

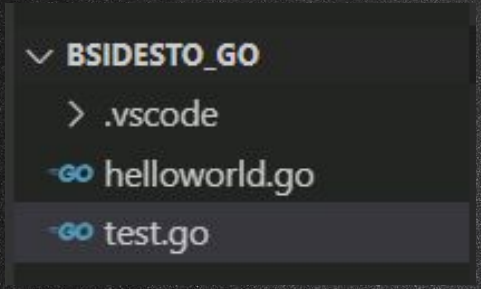
run>run without debugging



https://github.com/Haydz/bsidestogo101/blob/main/helloworld.go

Trying to run a 2nd file

What happens if we attempt to run test.go?




```go
package main

import "fmt"

func main(){
    fmt.Println("This is the 2nd file")
}
```

Trying to run a 2nd file

An error is received!

```
# _/c_/bsidesto_go
c:\bsidesto_go\test.go:5:6: main redeclared in this block
        previous declaration at c:\bsidesto_go\helloworld.go:5:6
Build process exiting with code: 2 signal: null
```

Why:

Go programs are organized into packages.

A package is a collection of source files in the SAME directory that are compiled together.

Items in one source file are visibility to all other source files within the same package

Run different files in a project

Place each file in a different folder, this makes each file a different ‘project’.

```
C:.
├───.vscode
├───helloworld
│       helloworld.go
│
└───test
        test.go
```

```go
package main

import "fmt"

func main() {
    fmt.Println("test")
}
```

https://github.com/golangci/golangci-lint/issues/1370

Go Commands

"**go**" is a tool for managing Go source code.

"**go run**" = compile and run a Go program

"**go build**" = compile packages and dependencies into an executable

** We will cover Go Build further into the training"

https://pkg.go.dev/cmd/go
https://astaxie.gitbooks.io/build-web-application-with-golang/content/en/01.3.html

For Today's training

If using the VM, use the **“code”** directory on the desktop.

Use the **go mod init** command from the code directory:

```
go mod init github.com/code
```

Create a folder for each Go file from within the “**code**” directory.

This will avoid many errors

```
bsidesto@bsidesto-go:~/Desktop/code/pre_bsidesto$ ls
go.mod  helloworld.go
```

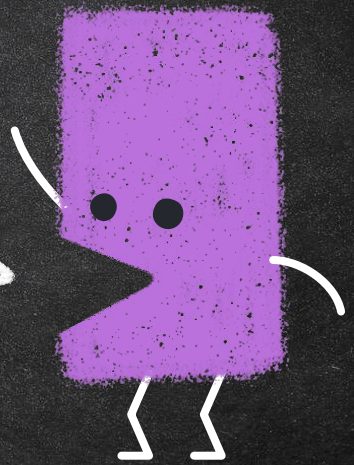
Go Syntax

Content to cover

- Data Types
- Assigning Variables
- Arrays
- Slices
- Strings
- Printing to screen
- For Loops
- Relational and Logical Operators
- Math
- If Statements
- Functions
- Structures
- Modules

“

Learn about the basic programming concepts in Go



To remember

Every file starts with the package information. Which in the main code is:

```
package main
```

Then packages are imported, which again looks like:

```
import "fmt"
```

A main function is used in package main. This is run automatically

```
func main() {
    fmt.Println("test")
}
```

```
package main

import "fmt"

func main() {
    fmt.Println("test")
}
```

Code organization

It is a good habit to write code as if it will be published.

A repository contains one or more modules. A module is a collection of related Go packages that are released together.

Each modules path serve as the import path for its packages, but also where the go command looks to download it.

golang.org/x/tools converts to https://golang.org/x/tools

```
bsidesto@bsidesto-go:~/Desktop/code/pre_bsidesto$ ls
go.mod  helloworld.go
```

Data Types

Integers = data type used to represent real numbers that do not have fractional values.

```
var number int
```

```
number := 5
```

String = ordered sequences of characters, letters, numbers etc

```
var word string
```

```
word := "Hello world"
```

Float (or Real) = for numbers that contain decimal points, or for fractions.

```
var adv_number float32
```

```
adv_number := 2.25
```

Boolean (or bool). Used where data is restricted to True/False or yes/no options.

```
var true_false bool
```

```
true_false := true
```

https://www.geeksforgeeks.org/data-types-in-go/

Strings

Most programming languages have a data type called a string, which is used for data values that are made up of ordered sequences of characters

```
cat_name := "Muffins"
dog_name := "Max"

fight := cat_name + " fought " + dog_name
fmt.Println(fight)
```

```
Muffins fought Max

Program exited.
```

Assigning Variables

A variable is a storage location, with a specific type and an associated name

`var` declares 1 or more variables

```
var x string = "Hello World"
```

```
var x string
x = "Hello World"
fmt.Println(x)
```

```
food := "apple"
```

https://gobyexample.com/variables

Variables - Examples

```
cat_name := "Muffins"
dog_name := "Max"
total_receipt := 28.50
colour_blue := true
```

Arrays

Store multiple of the same data type in one variable. Requires a length and element type

Is an index of the values starting from Zero

```
array_one := [6]int{1, 2, 3, 4, 5, 6}
```

```
array_two := [4]string{"hello", "welcome", "to", "bsides"}
```

array_two:

Index	0	1	2	3
Value	Hello	Welcome	to	bsides

array_one looks like this:

Index	0	1	2	3	4	5
Value	1	2	3	4	5	6

```go
package main

import "fmt"

func main() {
    array_one := [6]int{1, 2, 3, 4, 5, 6}
    array_two := [4]string{"hello", "welcome", "to", "bsides"}

    fmt.Println(array_one)
    fmt.Println(array_two)

}
```

```
[1 2 3 4 5 6]
[hello welcome to bsides]
```

Arrays – how to access

Access array elements by referring to the index number

```go
func main() {
    array_one := [6]int{1, 2, 3, 4, 5, 6}

    fmt.Println(array_one[0])
    fmt.Println(array_one[1])
    fmt.Println(array_one[2])

}
```

```
1
2
3

Program exited.
```

Index	0	1	2	3	4	5
Value	1	2	3	4	5	6

Arrays – question

Given the below array number below, what value is array_two[2]???

Or what value is at index 2

```
array_two := []string{"hello", "welcome", "to", "bsides"}
```

```
fmt.Println(array_two[2])
```

https://www.golangprograms.com/how-to-assign-and-access-array-element-values-in-go.html

```go
package main

import "fmt"

func main() {
    array_two := [4]string{"hello", "welcome", "to",
"bsides"}
    fmt.Println(array_two[2])
}
```

```
to
```

Slices

Arrays are constant, cannot be changes. Slices are more common and flexible. Does not contain a size, so can grow or shrink.

Store multiple of the same data type in one variable.

As shown below, the size is not included:

```
slice_one := []int{1, 2, 3, 4, 5, 6}
```

```
slice_two := []string{"hello", "welcome", "to", "bsides"}
```

https://go.dev/blog/slices-intro

```go
package main

import "fmt"

func main() {
    slice_one := []int{1, 2, 3, 4, 5, 6}
    slice_two := []string{"hello", "welcome", "to", "bsides"}
    fmt.Println(slice_one)
    fmt.Println(slice_two)

}
```

```
[1 2 3 4 5 6]
[hello welcome to bsides]

Program exited.
```

Slices – Appending or Adding

Can add to a slice by using the append() function.

“Technically creating a new slice from the old slice with the extra value.

```
slice_one := []int{1, 2, 3, 4, 5, 6}
slice_one = append(slice_one, 7)
```

https://www.tutorialspoint.com/go/go_slice.htm

Slices – Appending or Adding

```go
package main

import "fmt"

func main() {
    slice_one := []int{1, 2, 3, 4, 5, 6}
    slice_one = append(slice_one, 7)
    fmt.Println(slice_one)

}
```

```
[1 2 3 4 5 6 7]

Program exited.
```

Printing to the screen

Printing to the screen (console) is an easy way to see output from your code.

Might want to check:

- ➜ How the program runs
- ➜ What values are variables holding
- ➜ The logic of your loops or if statements

```
import (
    "fmt"
)
```

Need the "**fmt**" package

```
fmt.Println()
```

Println() = will print all arguments to the console

```
fmt.Printf()
```

PrintF() = used for printing formatted strings to console

https://sunitc.dev/2018/11/22/how-to-print-to-console-in-go/

Printing to the screen

```go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, Bsides Toronto Students!")
}
```

```
Hello, Bsides Toronto Students!

Program exited.
```

Println vs Printf

fmt.Println()

Multiple arguments require a space and comma.

Prints a New Line character at the end

```
number := 10
words := "stuff"

fmt.Println(number, words)
```

fmt.Printf()

More options for formatting.

Uses formatting verbs:

Integer: %d

String: %s

So much more

```
number := 10
words := "stuff"

fmt.Printf("%d, %s", number, words)
```

Println vs printf

```go
package main

import (
    "fmt"
)

func main() {

    number := 10
    words := "stuff"

    fmt.Println(number, words)
    fmt.Println(number, words)
    fmt.Printf("%d %s", number, words)
    fmt.Printf("%d %s", number, words)
}
```

New line character, 2nd Println is on a new line

```
10 stuff
10 stuff
10 stuff10 stuff
Program exited.
```

Printf does not include a new line character

FOR Loops

Go only has the concept of `for` loops.

For loops execute blocks of code over and over again. The advantage to a for loop is we know exactly how many times the loop will execute before the loop starts.

This example loops over a range:

```
for initialisation; condition; post {
}
```

```
nums := []int{1, 2, 3, 4}

    for _, number := range nums {
        fmt.Println(number)
    }
```

```
1
2
3
4

Program exited.
```

https://golangbot.com/loops/
https://yourbasic.org/golang/for-loop/

Loops advanced

"Three component Loop" / The iteration loop

"While a number is less than x, continue to loop, at the end of each loop increase x.

```
for i := 1; i < 5; i++ {
    fmt.Println(i)
}
```

The "While" Loop

While x is not y, keep looping

```
number := 1
for number < 5 {
    number += 2
}
```

Exiting Loops

The continue command will execute the next iteration of the innermost for loop.

In a single for loop, the code skips the rest of the loop and starts again, from the next iteration of i++.

```
for i := 1; i < 5; i++ {
    if i != 2 {
        fmt.Println("i: ", i)
        continue
    }
    fmt.Println("We triggered on #2 ")
}
```

```
i:  1
We triggered on #2
i:  3
i:  4
```

The break statement leaves the innermost for loop.

In a loop with 1 for loop, it exits the loops

```
for i := 1; i < 5; i++ {
    if i != 2 {
        fmt.Println("i: ", i)

        break
    }
    fmt.Println("We triggered on #2 ")
}
```

```
i:  1

Program exited.
```

For loops - exercise

Create a for loop that iterates to 10, and prints “found the number 5” on the 5th iteration, and breaks on the 9th iteration For every iteration that is not the 5th or 9th, print the iteration number.

Effectively combining the 2 concepts on the previous slide.

For Loops Answer

```go
package main

import (
    "fmt"
)

func main() {

    for i := 1; i < 11; i++ {
        if i == 9 {
            break
        } else if i == 5 {
            fmt.Println("WE FOUND 5")
            continue

        } else {
            fmt.Println("i: ", i)
        }
    }

}
```

Math Operations

As with all programming languages, math operations are built in

```go
func main() {

    num1 := 5
    num2 := 5

    add_nums := num1 + num2
    fmt.Println(add_nums)
}
```

```
10

Program exited.
```

Relational & Logical Operators

RELATIONAL

Greater than

Less Than

Not equal

Equal

```
fmt.Println(5 > 10)
fmt.Println(5 < 10)
fmt.Println(5 == 10)
fmt.Println(5 != 10)
```

LOGICAL:

AND

OR

```
nums := []int{1, 2, 3, 4}
for _, number := range nums {
    if number == 2 || number == 4 {
        fmt.Println(number)
    }
}
```

Relational & Logical Operators

```go
nums := []int{1, 2, 3, 4}
for _, number := range nums {
    if number == 2 || number == 4 {
        fmt.Println(number)
    }
}
```

```
2
4

Program exited.
```

Exercise: Math

Calculate:

A: (2 x 59) / 4

Calculate:

B: 40 x 40 x 40

Is (2 x 59) / 4 greater than 40 x 40 x 40 ?

```go
package main

import (
    "fmt"
)

func main() {

}
```

ANSWER

```go
package main

import (
    "fmt"
)

func main() {

    answer_a := (2 * 59) / 4
    fmt.Println(answer_a)

    answer_b := 40 * 40 * 40
    fmt.Println(answer_b)

    fmt.Println(answer_a > answer_b)
}
```

If statements

If statements are decision-making options that guides a program to make decisions based on specific criteria.

Based on if a set of code is **TRUE** or **FALSE**.

“If the traffic light is red, stop the car”

```
if number == 2 || number == 4 {
            fmt.Println(number)
```

IF ELSE

If the statement is false do this instead (execute a different block of code).

“If the traffic light is red, stop the car, otherwise keep driving.”

```
number := 199
if number > 250 {
    fmt.Println("Higher")
} else {
    fmt.Println("The number is less than 250")
}
```

IF ELSE

```
The number is less than 250

Program exited.
```

```go
number := 199
if number > 250 {
    fmt.Println("Higher")
} else {
    fmt.Println("The number is less than 250")
}
```

BUT – CAREFUL

- What if the number is 250?
- Which statement will execute?

Is 250 > than 250???

```
The number is less than or equal to 250

Program exited.
```

To fix this, use greater than or equal to (>=).

This is specific to this function, but servers as something to check if you have a logic issue.

```
250 or higher

Program exited.
```

```go
func main() {
    number := 250
    if number >= 250 {
        fmt.Println("250 or higher")
    } else {
        fmt.Println("The number is less than
250")
    }

 }
```

Exercise: Loops & If

Loop through the numbers 1 - 10

If the number is 3:

Print “its a 3”

Else:

Print “Not a 3”

```go
package main

import (
    "fmt"
)

func main() {

}
```

Suggestions

Can loop through an array of numbers 1 to 10.

Or create an incrementing loop (C style)

Answer

```go
package main

import (
    "fmt"
)

func main() {
    number := 1
    for i := 0; i < 11; i++ {
        if number == 3 {
            fmt.Println("Is a 3")
        } else {
            fmt.Println("Is not a 3")
        }
        number += 1
    }
}
```

Functions in Go

Functions (also called 'procedures' in some programming languages) are a set of instructions bundled together to achieve a specific outcome.

- Alternative to having repeating blocks of code.
- They break long programs up into smaller components
- Can be shared and used by other programmers

```go
func print_name(name string) string {
    return "Hello " + name
}
```

https://gobyexample.com/functions

```go
package main

func print_name(name string) string {
    return "Hello " + name
}

func main() {
    name := "Haydn"
    test_function := print_name(name)
    println(test_function)
}
```

```
Hello Haydn

Program exited.
```

Requires the type of variable and type or return variable to be declared

```go
func print_name(name string) string {
    return "Hello " + name
}
```

Requires an explicit return statement

Structs / Structures

A struct/structure is compared with the class in Object-Oriented Programming.

A structure is used when you need to define a schema made of different individual fields. An object can be created from the schema.

Helpful in parsing JSON

```
type StructName struct {
    field1 fieldType1
    field2 fieldType2
```

https://medium.com/rungo/structures-in-go-76377cc106a2

```go
package main

import "fmt"

type fighter struct {
    name   string
    health int
}

func main() {
    var chun_li fighter
    chun_li.health = 100
    chun_li.name = "Chun Li"

    fmt.Println(chun_li)
    fmt.Println("Chun Li Health: ", chun_li.health)
}
```

Defining the fields

Initializing a struct and setting values

Accessing parts of the struct

```go
package main

import "fmt"

type fighter struct {
    name   string
    health int
}

func main() {
    var chun_li fighter
    chun_li.health = 100
    chun_li.name = "Chun Li"

    fmt.Println(chun_li)
    fmt.Println("Chun Li Health: ", chun_li.health)
}
```

```
{Chun Li 100}
Chun Li Health:  100

Program exited.
```

Create your own structure

Let's take 5 mins and just create a structure.

Create a structure that called "car":

- ➜ Add a model
- ➜ Add a make

```go
package main

import "fmt"

type fighter struct {
    Name   string
    Health int
}

func main() {
    var chun_li fighter
    chun_li.health = 100
    chun_li.name = "Chun Li"

    fmt.Println(chun_li)
    fmt.Println("Chun Li Health: ", chun_li.health)
}
```

Answer

```go
package main

import (
    "fmt"
)

type car struct {
    Make  string
    Model string
}

func main() {

    var fast car
    fast.Make = "Toyota"
    fast.Model = "Supra"

    fmt.Println(fast)
}
```

```
{Toyota Supra}
```

Installing Packages

Packages allow the use of additional functionality. Previously we quickly went through the standard library packages.

Such as

```
import (
    "fmt"
    "time"
)
```

It is possible to install 3rd party packages, those that are not in the standard library. These allow specific functions beyond the standard library.

Use the “go get” command to install packages. For example, to install the gopacket library use:

```
go get https://github.com/google/gopacket
```

```
import (
"github.com/google/gopacket
)
```

Building executables

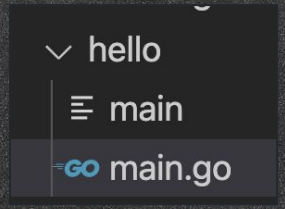
GO BUILD

Use the “go build” command to build an executable of a Go program

“go build main.go” = builds an executable called main

Go executables can be compiled for different operating systems and architectures.

```
% go build main.go
```




```
hello % ./main
Hello Students!
```

https://www.digitalocean.com/community/tutorials/how-to-build-go-executables-for-multiple-platforms-on-ubuntu-16-04

GO BUILD

Go executables can be compiled for different operating systems and architectures.

```
hello % file main
main: Mach-O 64-bit executable x86_64
```

“GOOS” = the Operating System

“GOARCH” = the architecture

Example

GOOS=linux

GOARCH=amd64

```
env GOOS=linux GOARCH=amd64  go build main.go
```

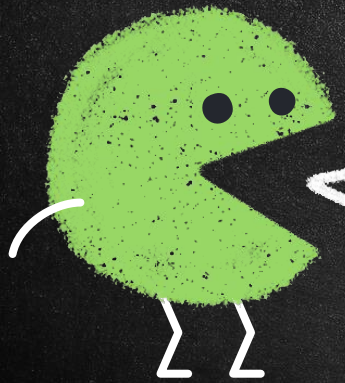
```
hello % file main
main: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
GyTbWNS_G/5azj9Jhdp30nCb7Uw8gI, not stripped
```

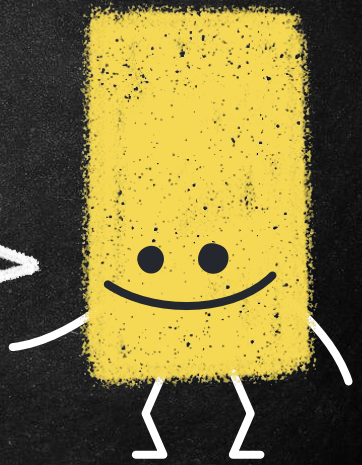
Examples

GOOS - Target Operating System	GOARCH - Target Platform
android	arm
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64
freebsd	arm
linux	386
linux	amd64
linux	arm
linux	arm64

https://www.digitalocean.com/community/tutorials/how-to-build-go-executables-for-multiple-platforms-on-ubuntu-16-04

More Go





Content to cover

- Pointers
- Error Handling
- Goroutines

Pointers

A pointer is simply a variable containing the address of another variable. Every variable has a location in memory

"&" = accessing the location where its value is stored in memory

"*" access the values held at a location in memory

```go
i := 42
fmt.Println(&i)

p := &i
fmt.Println(p)

fmt.Println(*p)
```

Accessing the memory location of i

Placing the memory location of "i" in "p"

Acces the value of the memory location of "p", which is actually the memory location of "i", si it prints the value of "i"

```
memory loc: 0xc000126000
memory loc: 0xc000126000
value: 42
```

https://medium.com/@annapeterson89/whats-the-point-of-golang-pointers-everything-you-need-to-know-ac5e40581d4d
https://tour.golang.org/moretypes/1

Error Handling

Different to other programming languages.

Errors indicated something went wrong in the code. For example, writing to a file might not work and it is good to be notified.

```go
    file, err := os.Open("fake.log")
    if err != nil {
        fmt.Println(err)

    }
```

```
open fake.log: The system cannot find the file specified.
```

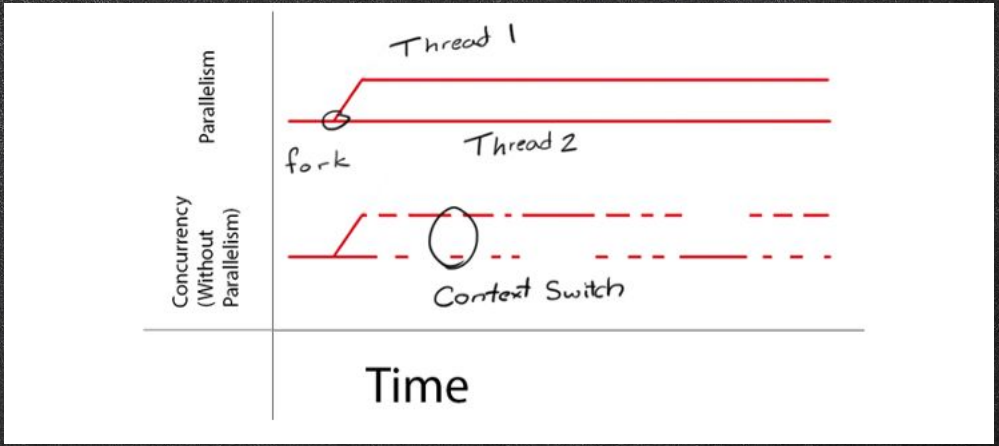
https://medium.com/rungo/error-handling-in-go-f0125de052f0

Goroutines

Concurrency = working on different tasks, but only 1 at a time

Parallelism = multiple tasks being executed at the same time

There is where gorountines come in. Different goroutines can be created and executing different tasks at the same time with goroutines enables concurrency.



https://medium.com/technofunnel/understanding-golang-and-goroutines-72ac3c9a014d

Goroutines

- Goroutines communicate using channels, this prevents any race conditions when accessing shared memory.
- Goroutines are tiny compared to threads, only a few kb in stack size

"Go" infront of a function or method makes a Goroutine. Control does not wait for the function to finish, it continues on with the rest of the program.

```go
func hello() {
    fmt.Println("Hello world goroutine")
}
func main() {
    go hello()
    time.Sleep(1 * time.Second)
    fmt.Println("main function")
}
```

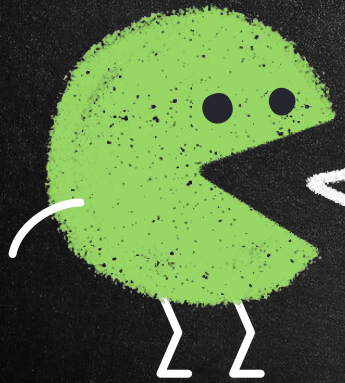
https://golangbot.com/goroutines/

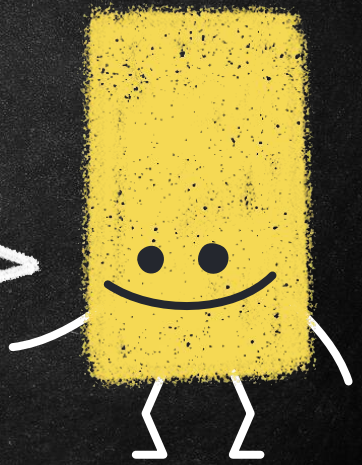
Goroutines – example

```go
package main

import (
    "fmt"
    "time"
)

func numbers() {
    for i := 1; i <= 5; i++ {
        time.Sleep(250 * time.Millisecond)
        fmt.Printf("%d ", i)
        fmt.Println("1st routine")
    }
}
func alphabets() {
    for i := 'a'; i <= 'e'; i++ {
        time.Sleep(400 * time.Millisecond)
        fmt.Printf("%c ", i)
        fmt.Println("2nd routine")
    }
}
func main() {
    go numbers()
    go alphabets()
    time.Sleep(3000 * time.Millisecond)
    fmt.Println("main terminated")
}
```

```
1 1st routine
a 2nd routine
2 1st routine
3 1st routine
b 2nd routine
4 1st routine
c 2nd routine
5 1st routine
d 2nd routine
e 2nd routine
main terminated
```

Even More Go





Content to cover

- Writing to a File
- Reading a File
- Web Requests
- JSON Parsing

Writing to a File

In simplest terms

os.Create() = creates a file with desired name

file.WriteString() = writes data to a file

defer = used to delay closing a file until all data has been written

But can do more.

- ➜ Write bytes to a file
- ➜ Write strings line by line
- ➜ Append

```
file, _ := os.Create("file.txt")
defer file.Close()
file.WriteString("A file being created for students to learn")
```

https://golangbot.com/write-files/

```go
package main
import (
    "fmt"
    "log"
    "os"
)
func main() {
    file, err := os.Create("file.txt")

    if err != nil {
        log.Fatalf("failed creating file: %s", err)
    }
    defer file.Close()
    _, err = file.WriteString("A file being created for students to learn")
    if err != nil {
        log.Fatalf("failed writing to file: %s", err)
    }
    fmt.Printf("\nFile Name: %s", file.Name())
}
```

Reading a File

Use “**ioutil.ReadFile**”

body, _ioutil.ReadFile (“filename.txt”)

String_body := string(body)

fmt.Println(body)

```go
package main

import (
    "fmt"
    "io/ioutil"
)

func main() {
    data, err := ioutil.ReadFile("fun.txt")
    if err != nil {
        fmt.Println("File reading error", err)
        return
    }
    string_data := string(data)
    fmt.Println("Contents of file:", string_data)
}
```

https://zetcode.com/golang/readfile/

Reading a File – Line by line

Use the “**bufio**” package
Use “**scanner**” for reading data that is newline-delimited.

- ➜ Open the file
- ➜ Create a “new scanner” (default splits by new line)
- ➜ Defer closing the file
- ➜ Loop through scanner, select text and print to console or store in a variable

```go
func main() {

    file, _ := os.Open("file.txt")

    defer file.Close()
    scanner := bufio.NewScanner(file)

    for scanner.Scan() {
        fmt.Println(scanner.Text())
    }

    file.Close()
}
```

Reading & Parsing exercise

Read in the file “access.log” and search for the words bsides_is_not_real (or short version) and print the line to the console (screen).

This will require:

- ➜ Reading in the file
- ➜ Storing the data
- ➜ Looping through the data and searching for the words bsides_is_not_real
    - Tip: You will need a new package to search in a string

```go
package main

import (
    "bufio"
    "fmt"
    "log"
    "os"
    "strings"
)

func main() {


    file, err := os.Open("access.log")
    if err != nil {
        log.Fatalf("failed to open")

    }
```

Opening the file

Creating Scanner

Variable to store the text

```go
    scanner := bufio.NewScanner(file)

    var text []string

    for scanner.Scan() {
        text = append(text, scanner.Text())
    }

    file.Close()

    for _, each_ln := range text {
     //     fmt.Println(each_ln)
        if strings.Contains(each_ln, "bsides") {
            fmt.Println("Found the line with bsides:", each_ln)
        }
    }
}
```

Using the scanner and storing each line in the array “text”

Can print each line to screen to ensure code is working

Looping through the array “text” and if a line has the word “bsides”, print to the screen

https://github.com/Haydz/bsidestogo101/blob/main/read_parse_ex_1.go

Web Requests

HTTP (S) REQUESTS – GET

Need the **"net/http"** package. Provides all the utilities to make HTTP requests.

The Get function takes in a URL and returns a response of type pointer to a **struct** and an **error**.

```
resp, _ := http.Get("https://www.bsidesto.ca/")
```

Attempting to print the response will show the header and properties of the request made.

```
&{200 OK 200 HTTP/2.0 2 0 map[Cf-Team:[0d4aba6d36000053fb0110e40
ate:[Fri, 22 Oct 2021 17:51:46 GMT] Server:[gunicorn/20.0.4] Via
000e0420}
```

Need **ioutil.ReadMe** to access the **Body** property of the request.

This also returns a body and an error.

```
body, _ := ioutil.ReadAll(resp.Body)
```

Printing the body will give bytes of data:

```
[60 33 68 79 6
6 101 109 112
114 32 102 114
```

Must convert the response body to string.

```
body_string := string(body)
```

```
<!DOCTYPE HTML>
<!--
        Typify by TEMPLATED
        templated.co @templatedco
        Released for free under the
```

https://blog.logrocket.com/making-http-requests-in-go/

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)
func main() {
    resp, err := http.Get("https://www.bsidesto.ca/")
    if err != nil {
        log.Fatalln(err)
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalln(err)
    }
    body_string := string(body)
    fmt.Println(body_string)
}
```

Error handling

HTTP Exercise

Write a go program to read a webpage, parse the data and find the duplicate letter from the alphabet.

Web page:

https://bsidesto2021.s3.ca-central-1.amazonaws.com/bsidesto_2.txt

What is needed:

- ➜ Http request to read the webpage
- ➜ Storing the body of the webpage
- ➜ Looping through the webpage variable
- ➜ ... <no spoilers>

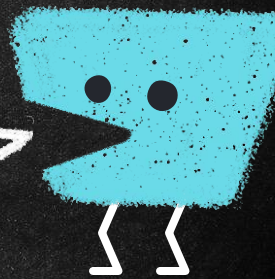
```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

func main() {
    resp, err := http.Get("https://bsidesto2021.s3.ca-central-1.amazonaws.com/bsidesto_2.txt")
    if err != nil {
        log.Fatalln(err)
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalln(err)
    }
    alphabet := string(body)

    for index, letter := range alphabet {
        for index_check, letter_check := range alphabet {
            if letter_check == letter && index_check != index {
                fmt.Println("duplicate found: ", string(letter))
            }
        }
        fmt.Println(string(letter))

    }
}
```

JSON & ~~XML~~ PARSING

What is JSON?

JSON text is text only, an open standard file format that is human readable.

Used to transmit data objects consisting of attribute-value pairs and arrays.



JSON

It looks like these examples.

```
{"firstName":"John", "lastName":"Doe"}
```

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

```
{
    "employees":[
        {"firstName":"John", "lastName":"Doe"},
        {"firstName":"Anna", "lastName":"Smith"},
        {"firstName":"Peter", "lastName":"Jones"}
    ]
}
```

https://www.w3schools.com/whatis/whatis_json.asp

JSON – Parsing

We need to use the **“encoding/json”** module.

```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

The **Unmarshal** function parses raw JSON data in the form of **[]byte variables**.

```
import "encoding/json"
myJsonString := `{"title":"json"}`
json.Unmarshal([]byte(myJsonString), &myStoredVariable)
```

Storing the data in the variable we want

JSON Example

- Need to receive or have JSON data
- Have a struct to hold the JSON data
- Read the data and unmarshall the JSON data
- Print to screen (do something with the data)

```go
resp, err := http.Get("https://jsonplaceholder.typicode.com/todos/1")
```

```json
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

```go
type Book struct {
    UserID    int
    Id        int
    Title     string
    Completed string
}
```

```go
json.Unmarshal([]byte(body), &book_1)
```

```go
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

type Book struct {
    UserID    int
    Id        int
    Title     string
    Completed string
}
```

```go
func main() {
    resp, err := http.Get("https://jsonplaceholder.typicode.com/todos/1")
    if err != nil {
        log.Fatalln(err)
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalln(err)
    }
    body_string := string(body)
    fmt.Println(body_string)

    var book_1 Book
    json.Unmarshal([]byte(body), &book_1)
    fmt.Println(book_1)

    fmt.Println("Book Title:", book_1.Title)
}
```

https://github.com/Haydz/bsidestogo101/blob/main/json_parsing.go

Exercise 1

Task: To download and parse some basic json data.

- ➜ Download the JSON data from a website
- ➜ Parse the JSON data
- ➜ Print the fields

Website: "https://bsidesto2021.s3.ca-central-1.amazonaws.com/json_ex_1.json

What you will need:

- ➜ HTTP Request
- ➜ Reading the data
- ➜ Unmarshalling the data **(a "byte")**

```go
fmt.Println(book_1)

fmt.Println("Book Title:", book_1.Title)
```

```
{1 1 delectus aut autem }
Book Title: delectus aut autem
```

XML

Used much less.

Read data in

Define Structs

Unmarshall XML

MITRE
ATT&CK!

What is the MITRE ATT&CK framework ?

MITRE ATT&CK

- The MITRE ATT&CK framework was developed in order to provide security researchers with a uniform means of expressing:
  - The relationships between **groups**, **tactics** and **techniques** (i.e. the high-level *goals* of a particular group, and the means by which that group can achieve their goals)
    - e.g. a **threat actor** may use **scheduled tasks** to **execute code,** achieve **persistence**, and perform **privilege escalation**

  - The relationships between **malware families** or **tools** and **groups**, **tactics**, and **techniques**

https://medium.com/mitre-attack/att-ck-101-17074d3bc62
Free training:
https://attack.mitre.org/resources/training/cti/

ATT&CK®

Getting Started | Take a Tour
Contribute | Blog
FAQ | Random Page

ID: T1082
Sub-techniques: No sub-techniques
Tactic: Discovery
Platforms: IaaS, Linux, Windows, macOS

Embed | View on Twitter

ATT&CK Matrix for Enterprise

layout: side | show sub-techniques | hide sub-techniques

Reconnaissance (10 techniques)	Resource Development (7 techniques)	Initial Access (9 techniques)	Execution (12 techniques)	Persistence (19 techniques)	Privilege Escalation (13 techniques)	Defense Evasion (40 techniques)	Credential Access (15 techniques)	Discovery (29 techniques)	Lateral Movement (9 techniques)	Collection (17 techniques)	Command and Control (16 techniques)	Exfiltration (9 techniques)	13
Active Scanning (2)	Acquire Infrastructure (6)	Drive-by Compromise	Command and Scripting Interpreter (8)	Account Manipulation (4)	Abuse Elevation Control Mechanism (4)	Abuse Elevation Control Mechanism (4)	Adversary-in-the-Middle (2)	Account Discovery (4)	Exploitation of Remote Services	Adversary-in-the-Middle (2)	Application Layer Protocol (4)	Automated Exfiltration (1)	Accou
Gather Victim Host Information (4)	Compromise Accounts (2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Brute Force (4)	Application Window Discovery	Internal Spearphishing	Archive Collected Data (3)	Communication Through Removable Media	Data Transfer Size Limits	Remo
Gather Victim Identity Information (3)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (15)	Boot or Logon Autostart Execution (15)	BITS Jobs	Credentials from Password Stores (5)	Browser Bookmark Discovery	Lateral Tool Transfer	Audio Capture	Data Encoding (2)	Exfiltration Over Alternative Protocol (3)	Data
Gather Victim Network Information (6)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (5)	Build Image on Host	Exploitation for Credential Access	Cloud Infrastructure Discovery	Remote Service Session Hijacking (2)	Automated Collection	Data Obfuscation (3)	Exfiltration Over C2 Channel	Data for Im
Gather Victim Org Information (4)	Establish Accounts (2)	Phishing (3)	Inter-Process Communication (2)	Browser Extensions	Create or Modify System Process (4)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Remote Services (6)	Browser Session Hijacking	Dynamic Resolution (3)	Exfiltration Over Other Network Medium (1)	Defac
Phishing for Information (3)	Obtain Capabilities (6)	Replication Through Removable Media	Native API	Compromise Client Software Binary	Domain Policy Modification (2)	Deploy Container	Forge Web Credentials (2)	Cloud Service Discovery	Replication Through Removable Media	Clipboard Data	Encrypted Channel (2)	Exfiltration Over Physical Medium (1)	Disk W
Search Closed Sources (2)	Stage Capabilities (5)	Supply Chain Compromise (3)	Scheduled Task/Job (6)	Create Account (3)	Escape to Host	Direct Volume Access	Input Capture (4)	Cloud Storage Object Discovery	Software Deployment Tools	Data from Cloud Storage Object	Fallback Channels	Exfiltration Over Web Service	Endpo of Ser
Search Open Technical Databases (5)		Trusted Relationship	Shared Modules	Create or Modify System Process (4)	Event Triggered Execution (15)	Execution Guardrails (1)	Modify Authentication Process (4)	Container and Resource Discovery	Taint Shared	Data from Configuration Repository (2)	Ingress Tool Transfer		Firmw Corru
Search Open Websites/Domains (2)		Valid	Software Deployment Tools	Event Triggered		Exploitation for Defense Evasion	Network	Domain Trust Discovery		Data from Information	Multi-Stage		Inhibi Recov
			System Services (2)			File and Directory Permissions		File and Directory Discovery					Netw Servic
													Reso

ATT&CK v10 has been released! Check out the blog post or release notes for more information.

Execution

The adversary is trying to run malicious code.

Execution consists of techniques that result in adversary-controlled code running on a local or remote system. Techniques that run malicious code are often paired with techniques from all other tactics to achieve broader goals, like exploring a network or stealing data. For example, an adversary might use a remote access tool to run a PowerShell script that does Remote System Discovery.

ID: TA0002
Created: 17 October 2018
Last Modified: 19 July 2019

Version Permalink

Techniques

Techn

ID	Name	Description
T1059	Command and Scripting Interpreter	Adversaries may abuse command and script interpreters to execute commands, scripts, or binaries. These interfaces and languages provic of interacting with computer systems and are a common feature across many different platforms. Most systems come with some built-in command-line interface and scripting capabilities, for example, macOS and Linux distributions include some flavor of Unix Shell while Wind installations include the Windows Command Shell and PowerShell.
.001	PowerShell	Adversaries may abuse PowerShell commands and scripts for execution. PowerShell is a powerful interactive command-line interface and scripting environment included in the Windows operating system. Adversaries can use PowerShell to perform a number of actions, includin discovery of information and execution of code. Examples include the `Start-Process` cmdlet which can be used to run an executable and `Invoke-Command` cmdlet which runs a command locally or on a remote computer (though administrator permissions are required to use PowerShell to connect to remote systems).
.002	AppleScript	Adversaries may abuse AppleScript for execution. AppleScript is a macOS scripting language designed to control applications and parts of via inter-application messages called AppleEvents. These AppleEvent messages can be sent independently or easily scripted with AppleScr These events can locate open windows, send keystrokes, and interact with almost any open application locally or remotely.
.003	Windows Command Shell	Adversaries may abuse the Windows command shell for execution. The Windows command shell (cmd) is the primary command prompt o Windows systems. The Windows command prompt can be used to control almost any aspect of a system, with various permission levels r for different subsets of commands. The command prompt can be invoked remotely via Remote Services such as SSH.

What is it useful for?

- Using as a guide for detections. What stages of an attack can you, or can't you detect on.
  - Mapping Defensive Controls
- Threat hunting
  - Threat hunt based on the gaps identified
- Helps with investigations
  - If you see one IOA, what others would be similar
- Red teaming
  - Offensive security can test attacks that are *known* to be executed in real environments
- Purple Teaming
  - Internal teams can use it to help with testing known gaps in defensive controls
- Management Reporting
  - It is very valuable to be able to report on the 'state of security' to management
    - Known gaps, visible improvements, budget requests

https://www.w3schools.com/whatis/whatis_json.asp

ATT&CK v10 has been released! Check out the blog post or release notes for more information. | MITRE ATT&

layer ×

selection controls | layer controls | technique control

Reconnaissance 10 techniques	Resource Development 7 techniques	Initial Access 9 techniques	Execution 12 techniques	Persistence 19 techniques	Privilege Escalation 13 techniques	Defense Evasion 40 techniques	Credential Access 15 techniques	Discovery 29 techniques	Lateral Movement 9 techniques	Collection 17 techniques	Command and Control 16 techniques	Exfiltration 9 techniques	In 13 te
Active Scanning (0/2)	Acquire Infrastructure (0/6)	Drive-by Compromise	Command and Scripting Interpreter (0/8)	Account Manipulation (0/4)	Abuse Elevation Control Mechanism (0/4)	Abuse Elevation Control Mechanism (0/4)	Adversary-in-the-Middle (0/2)	Account Discovery (0/4)	Exploitation of Remote Services	Adversary-in-the-Middle (0/2)	Application Layer Protocol (0/4)	Automated Exfiltration (0/1)	Accoun Remova
Gather Victim Host Information (0/4)	Compromise Accounts (0/2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (0/5)	Access Token Manipulation (0/5)	Brute Force (0/4)	Application Window Discovery	Internal Spearphishing	Archive Collected Data (0/3)	Communication Through Removable Media	Data Transfer Size Limits	Data De
Gather Victim Identity Information (0/3)	Compromise Infrastructure (0/6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (0/15)	Boot or Logon Autostart Execution (0/15)	BITS Jobs	Credentials from Password Stores (0/5)	Browser Bookmark Discovery	Lateral Tool Transfer	Audio Capture	Data Encoding (0/2)	Exfiltration Over Alternative Protocol (0/3)	Data En for Imp
Gather Victim Network Information (0/6)	Develop Capabilities (0/4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (0/5)	Boot or Logon Initialization Scripts (0/5)	Build Image on Host	Exploitation for Credential Access	Cloud Infrastructure Discovery	Remote Service Session Hijacking (0/2)	Automated Collection	Data Obfuscation (0/3)	Exfiltration Over C2 Channel	Data Manipu
Gather Victim Org Information (0/4)	Establish Accounts (0/2)	Phishing (0/3)	Inter-Process Communication (0/2)	Browser Extensions	Create or Modify System Process (0/4)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Remote Services (0/6)	Browser Session Hijacking	Dynamic Resolution (0/3)	Exfiltration Over Other Network Medium (0/1)	Defacer
Phishing for Information (0/3)	Obtain Capabilities (0/6)	Replication Through Removable Media	Native API	Compromise Client Software Binary	Domain Policy Modification (0/2)	Deploy Container	Forge Web Credentials (0/2)	Cloud Service Discovery	Replication Through Removable Media	Clipboard Data	Encrypted Channel (0/2)	Exfiltration Over Physical Medium (0/1)	Disk Wi
Search Closed Sources (0/2)	Stage Capabilities (0/5)	Supply Chain Compromise (0/3)	Scheduled Task/Job (0/6)	Create Account (0/3)	Escape to Host	Direct Volume Access	Input Capture (0/4)	Cloud Storage Object Discovery	Software Deployment Tools	Data from Cloud Storage Object	Fallback Channels	Exfiltration Over Web Service (0/2)	Endpoi Service
Search Open Technical Databases (0/5)		Trusted Relationship	Shared Modules	Create or Modify System Process (0/4)	Event Triggered Execution (0/15)	Domain Policy Modification (0/2)	Modify Authentication Process (0/4)	Container and Resource Discovery	Taint Shared Content	Data from Configuration Repository (0/2)	Ingress Tool Transfer	Scheduled Transfer	Firmwa Corrupt
Search Open Websites/Domains (0/2)		Valid Accounts (0/4)	Software Deployment Tools	Event Triggered Execution (0/15)	Exploitation for Privilege Escalation	Execution Guardrails (0/1)	Network Sniffing	Domain Trust Discovery	Use Alternate Authentication Material (0/4)	Data from Information Repositories (0/3)	Multi-Stage Channels	Transfer Data to Cloud Account	Inhibit S Recove
Search Victim-Owned Websites			System Services (0/2)	External Remote Services	Hijack Execution Flow (0/11)	Exploitation for Defense Evasion	OS Credential Dumping (0/8)	File and Directory Discovery		Data from Local System	Non-Application Layer Protocol		Networ Service
			User Execution (0/3)	Hijack Execution Flow (0/11)	Process Injection (0/11)	File and Directory Permissions Modification (0/2)	Steal Application Access Token	Group Policy Discovery		Data from Network Shared Drive	Non-Standard Port		Resour Hijackin
			Windows Management Instrumentation	Implant Internal Image	Scheduled Task/Job (0/6)	Hide Artifacts (0/9)	Steal or Forge Kerberos Tickets (0/4)	Network Service Scanning		Data from Removable Media	Protocol Tunneling		Service
				Modify Authentication Process (0/4)	Valid Accounts (0/4)	Hijack Execution Flow (0/11)	Steal Web Session Cookie	Network Share Discovery		Data Staged (0/2)	Proxy (0/4)		System Shutdo
				Office Application Startup (0/6)		Impair Defenses (0/9)	Two-Factor Authentication Interception	Network Sniffing		Email Collection (0/3)	Remote Access Software		
				Pre-OS Boot (0/5)		Indicator Removal on Host (0/6)	Unsecured Credentials (0/7)	Password Policy Discovery		Input Capture (0/4)	Traffic Signaling (0/1)		
				Scheduled Task/Job (0/6)		Indirect Command Execution		Peripheral Device Discovery		Screen Capture	Web Service (0/3)		
						Masquerading (0/7)		Permission Groups Discovery (0/3)		Video Capture			
						Modify Authentication Process (0/4)		Process Discovery					
						Modify Cloud Compute Infrastructure (0/4)		Query Registry					

ATT&CK v10 has been released! Check out the blog post or release notes for more information. | MITRE ATT&

layer ×

selection controls | layer controls | technique control

Reconnaissance 10 techniques	Resource Development 7 techniques	Initial Access 9 techniques	Execution 12 techniques	Persistence 19 techniques	Privilege Escalation 13 techniques	Defense Evasion 40 techniques	Credential Access 15 techniques	Discovery 29 techniques	Lateral Movement 9 techniques	Collection 17 techniques	Command and Control 16 techniques	Exfiltration 9 techniques	I 13 t
Active Scanning (0/2)	Acquire Infrastructure (0/6)	Drive-by Compromise	Command and Scripting Interpreter (0/8)	Account Manipulation (0/4)	Abuse Elevation Control Mechanism (0/4)	Abuse Elevation Control Mechanism (0/4)	Adversary-in-the-Middle (0/2)	Account Discovery (0/4)	Exploitation of Remote Services	Adversary-in-the-Middle (0/2)	Application Layer Protocol (0/4)	Automated Exfiltration (0/1)	Accoun Remova
Gather Victim Host Information (0/4)	Compromise Accounts (0/2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (0/5)	Access Token Manipulation (0/5)	Brute Force (0/4)	Application Window Discovery	Internal Spearphishing	Archive Collected Data (0/3)	Communication Through Removable Media	Data Transfer Size Limits	Data De
Gather Victim Identity Information (0/3)	Compromise Infrastructure (0/6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (0/15)	Boot or Logon Autostart Execution (0/15)	BITS Jobs	Credentials from Password Stores (0/5)	Browser Bookmark Discovery	Lateral Tool Transfer	Audio Capture	Data Encoding (0/2)	Exfiltration Over Alternative Protocol (0/3)	Data Er for Imp
Gather Victim Network Information (0/6)	Develop Capabilities (0/4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (0/5)	Boot or Logon Initialization Scripts (0/5)	Build Image on Host	Exploitation for Credential Access	Cloud Infrastructure Discovery	Remote Service Session Hijacking (0/2)	Automated Collection	Data Obfuscation (0/3)	Exfiltration Over C2 Channel	Data Manipu
Gather Victim Org Information (0/4)	Establish Accounts (0/2)	Phishing (0/3)	Inter-Process Communication (0/2)	Browser Extensions	Create or Modify System Process (0/4)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Remote Services (0/6)	Browser Session Hijacking	Dynamic Resolution (0/3)	Exfiltration Over Other Network Medium (0/1)	Deface
Phishing for Information (0/3)	Obtain Capabilities (0/6)	Replication Through Removable Media	Native API	Compromise Client Software Binary	Domain Policy Modification (0/2)	Deploy Container	Forge Web Credentials (0/2)	Cloud Service Discovery	Replication Through Removable Media	Clipboard Data	Encrypted Channel (0/2)	Exfiltration Over Physical Medium (0/1)	Disk Wi
Search Closed Sources (0/2)	Stage Capabilities (0/5)	Supply Chain Compromise (0/3)	Scheduled Task/Job (0/6)	Create Account (0/3)	Escape to Host	Direct Volume Access	Input Capture (0/4)	Cloud Storage Object Discovery	Software Deployment Tools	Data from Cloud Storage Object	Fallback Channels	Exfiltration Over Web Service (0/2)	Endpoi Service
Search Open Technical Databases (0/5)		Trusted Relationship	Shared Modules	Create or Modify System Process (0/4)	Event Triggered Execution (0/15)	Domain Policy Modification (0/2)	Modify Authentication Process (0/4)	Container and Resource Discovery	Taint Shared Content	Data from Configuration Repository (0/2)	Ingress Tool Transfer	Scheduled Transfer	Firmwa Corrupt
Search Open Websites/Domains (0/2)		Valid Accounts (0/4)	Software Deployment Tools	Event Triggered Execution (0/15)	Exploitation for Privilege Escalation	Execution Guardrails (0/1)	Network Sniffing	Domain Trust Discovery	Use Alternate Authentication Material (0/4)	Data from Information Repositories (0/3)	Multi-Stage Channels	Transfer Data to Cloud Account	Inhibit S Recove
Search Victim-Owned Websites			System Services (0/2)	External Remote Services	Hijack Execution Flow (0/11)	Exploitation for Defense Evasion	OS Credential Dumping (0/8)	File and Directory Discovery		Data from Local System	Non-Application Layer Protocol		Networ Service
			User Execution (0/3)	Hijack Execution Flow (0/11)	Process Injection (0/11)	File and Directory Permissions Modification (0/2)	Steal Application Access Token	Group Policy Discovery		Data from Network Shared Drive	Non-Standard Port		Resour Hijackin
			Windows Management Instrumentation	Implant Internal Image	Scheduled Task/Job (0/6)	Hide Artifacts (0/9)	Steal or Forge Kerberos Tickets (0/4)	Network Service Scanning		Data from Removable Media	Protocol Tunneling		Service
				Modify Authentication Process (0/4)	Valid Accounts (0/4)	Hijack Execution Flow (0/11)	Steal Web Session Cookie	Network Share Discovery		Data Staged (0/2)	Proxy (0/4)		System Shutdo
				Office Application Startup (0/6)		Impair Defenses (0/9)	Two-Factor Authentication Interception	Network Sniffing		Email Collection (0/3)	Remote Access Software		
				Pre-OS Boot (0/5)		Indicator Removal on Host (0/6)	Unsecured Credentials (0/7)	Password Policy Discovery		Input Capture (0/4)	Traffic Signaling (0/1)		
				Scheduled Task/Job (0/6)		Indirect Command Execution		Peripheral Device Discovery		Screen Capture	Web Service (0/3)		
						Masquerading (0/7)		Permission Groups Discovery (0/3)		Video Capture			
						Modify Authentication Process (0/4)		Process Discovery					
						Modify Cloud Compute Infrastructure (0/4)		Query Registry					

Scheduled Task/Job (T1053)

ATT&CK v10 has been released! Check out the blog post or release notes for more information.

TECHNIQUES
- Scheduled Task/Job
  - At (Linux)
  - At (Windows)
  - Cron
  - Launchd
  - Scheduled Task
  - Systemd Timers
  - Container Orchestration Job
- Shared Modules
- Software Deployment Tools
- System Services
- User Execution
- Windows Management Instrumentation
- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Collection
- Command and Control
- Exfiltration
- Impact

Scheduled Task/Job

Sub-techniques (7)

Adversaries may abuse task scheduling functionality to facilitate initial or recurring execution of malicious code. Utilities exist within all major operating systems to schedule programs or scripts to be executed at a specified date and time. A task can also be scheduled on a remote system, provided the proper authentication is met (ex: RPC and file and printer sharing in Windows environments). Scheduling a task on a remote system typically requires being a member of an admin or otherwise privileged group on the remote system.[1]

Adversaries may use task scheduling to execute programs at system startup or on a scheduled basis for persistence. These mechanisms can also be abused to run a process under the context of a specified account (such as one with elevated permissions/privileges).

ID: T1053

Sub-techniques: T1053.001, T1053.002, T1053.003, T1053.004, T1053.005, T1053.00 T1053.007

Tactics: Execution, Persistence, Privilege Escalation

Platforms: Containers, Linux, Windows, mac

Permissions Required: Administrator, SYST User

Effective Permissions: Administrator, SYST User

Supports Remote: Yes

CAPEC ID: CAPEC-557

Contributors: Alain Homewood, Insomnia Security; Leo Loobeek, @leoloobeek; Prashan Verma, Paladion; Travis Smith, Tripwire

Version: 2.1

Created: 31 May 2017

Last Modified: 15 October 2021

Version Permalink

Procedure Examples

ID	Name	Description

TECHNIQUES
Scheduled Task/Job
At (Linux)
At (Windows)
Cron
Launchd
Scheduled Task
Systemd Timers
Container Orchestration Job
Shared Modules
Software Deployment Tools
System Services
User Execution
Windows Management Instrumentation
Persistence
Privilege Escalation
Defense Evasion
Credential Access
Discovery
Lateral Movement
Collection
Command and Control
Exfiltration
Impact
Mobile

Procedure Examples

ID	Name	Description
S0447	Lokibot	Lokibot's second stage DLL has set a timer using "timeSetEvent" to schedule its next execution.[2]
S0125	Remsec	Remsec schedules the execution one of its modules by creating a new scheduler task.[3]

Mitigations

ID	Mitigation	Description
M1047	Audit	Toolkits like the PowerSploit framework contain PowerUp modules that can be used to explore systems for permission weaknesses in schedul tasks that could be used to escalate privileges. [4]
M1028	Operating System Configuration	Configure settings for scheduled tasks to force tasks to run under the context of the authenticated account instead of allowing them to run as SYSTEM. The associated Registry key is located at `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\SubmitControl`. The setting can be conf through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > Security Options: Domain Controlle Allow server operators to schedule tasks, set to disabled. [5]
M1026	Privileged Account Management	Configure the Increase Scheduling Priority option to only allow the Administrators group the rights to schedule a priority process. This can be c configured through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignmen Increase scheduling priority. [6]
M1018	User Account Management	Limit privileges of user accounts and remediate Privilege Escalation vectors so only authorized administrators can create scheduled tasks on r systems.

Detection

ID	Data Source	Data Component
DS0017	Command	Command Execution
DS0032	Container	Container Creation
DS0022	File	File Creation

At (Linux)
At (Windows)
Cron
Launchd
Scheduled Task
Systemd Timers
Container Orchestration Job
Shared Modules
Software Deployment Tools
System Services
User Execution
Windows Management Instrumentation
Persistence
Privilege Escalation
Defense Evasion
Credential Access
Discovery
Lateral Movement
Collection
Command and Control
Exfiltration
Impact
Mobile

Detection

ID	Data Source	Data Component
DS0017	Command	Command Execution
DS0032	Container	Container Creation
DS0022	File	File Creation
		File Modification
DS0009	Process	Process Creation
DS0003	Scheduled Job	Scheduled Job Creation

Monitor scheduled task creation from common utilities using command-line invocation. Legitimate scheduled tasks may be created during installation of new software or throug administration functions. Look for changes to tasks that do not correlate with known software, patch cycles, etc.

Suspicious program execution through scheduled tasks may show up as outlier processes that have not been seen before when compared against historical data. Data and event should not be viewed in isolation, but as part of a chain of behavior that could lead to other activities, such as network connections made for Command and Control, learning deta about the environment through Discovery, and Lateral Movement.

References

1. Microsoft. (2005, January 21). Task Scheduler and security. Retrieved June 8, 2016.
2. Muhammad, I., Unterbrink, H.. (2021, January 6). A Deep Dive into Lokibot Infection Chain. Retrieved August 31, 2021.
3. Kaspersky Lab's Global Research & Analysis Team. (2016, August 9). The ProjectSauron APT. Technical Analysis. Retrieved August 17, 2016.
4. PowerSploit. (n.d.). Retrieved December 4, 2014.
5. Microsoft. (2012, November 15). Domain controller: Allow server operators t schedule tasks. Retrieved December 18, 2017.
6. Microsoft. (2013, May 8). Increase scheduling priority. Retrieved December 1

MITRE
© 2015-2021, The MITRE Corporation. MITRE ATT&CK and ATT&CK are registered trademarks of The MITRE Corporation.
Privacy Policy Terms of Use ATT&CK v10.0
@MIT
Con

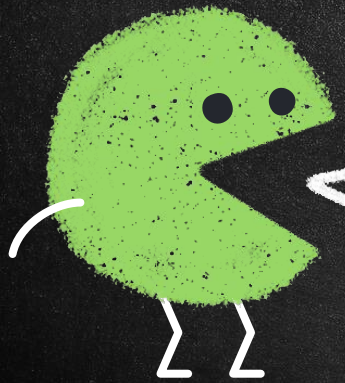
DATA SOURCES
Command
Command Execution
Container
Domain Name
Drive
Driver
File
Firewall
Firmware
Group
Image
Instance
Internet Scan
Kernel
Logon Session
Malware Repository
Module
Named Pipe
Network Share
Network Traffic
Persona
Pod
Process
Scheduled Job

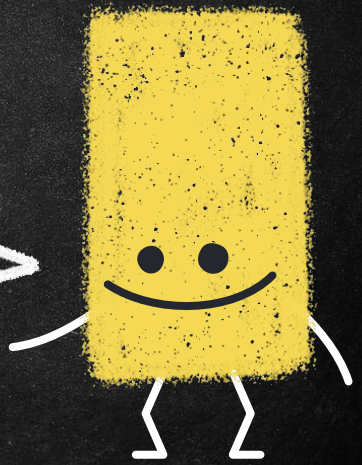
Command: Command Execution

Invoking a computer program directive to perform a specific task (ex: Windows EID 4688 of cmd.exe showing command-line parameters, ~/.bash_history, or ~/.zsh_history)

Enterprise	T1496		Resource Hijacking
Enterprise	T1053		Scheduled Task/Job
		.001	At (Linux)
		.002	At (Windows)
		.003	Cron
		.005	Scheduled Task
		.006	Systemd Timers
Enterprise	T1113		Screen Capture
Enterprise	T1505	.004	Server Software Component: IIS Components
Enterprise	T1489		Service Stop
Enterprise	T1218		Signed Binary Proxy Execution
		.001	Compiled HTML File
		.002	Control Panel
		.003	CMSTP
		.004	InstallUtil
		.005	Mshta
		.007	Msiexec
		.008	Odbcconf

Atomic Red Team!





Atomic Red Team

- The Atomic Red Team framework is a library of security test vectors that can be used to prove whether or not a particular behaviour is detectable by your security infrastructure (e.g. via EDR platforms such as FireEye HX or Microsoft Defender for Endpoint)
- Test vectors are available for Windows, Linux, and macOS systems

Atomic Red Team Training - Basics and Beyond - https://www.youtube.com/watch?v=d_E-hfKQ5Hw

redcanaryco / **atomic-red-team** Public

CircleCI Atomic Red Team doc generator Generate docs from job=generate_and_commit_guids_and_docs branch=mast… 717757d 3 days ago History

..

Indexes	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	3 days ago
T1003.001	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1003.002	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	3 months ago
T1003.003	rebuilt T1027.yml (#1649)	22 days ago
T1003.004	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1003.006	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1003.007	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1003.008	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1003	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1006	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1007	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1010	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1012	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1014	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1016	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	5 months ago
T1018	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1020	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago

Why GitHub? Team Enterprise Explore Marketplace Pricing Search / Sign in Sign up

redcanaryco / atomic-red-team Public

Notifications Star 5.2k Fork

Code Issues 10 Pull requests 2 Wiki Security Insights

master atomic-red-team / atomics / Go to file

CircleCI Atomic Red Team doc generator Generate docs from job=generate_and_commit_guids_and_docs branch=mast... ··· 717757d 3 days ago History

..		
Indexes	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	3 days ago
T1049	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
T1053.001	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	3 months ago
T1053.002	rebuilt T1027.yml (#1649)	22 days ago
T1053.003	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
T1053.004	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
T1053.005	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
T1053.006	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
T1053.007	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	5 months ago
	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago
T1020	Generate docs from job=generate_and_commit_guids_and_docs branch=mast...	2 months ago

redcanaryco / **atomic-red-team** Public

master **atomic-red-team** / atomics / **T1053.005** /

Go to file

CircleCI Atomic Red Team doc generator Generate docs from job=generate_and_commit_guids_and_docs branch=mast… bc21f59 on Sep 3 **History**

..		
src	T1053.005 create SchduledTask using WMI class (#1434)	6 months ago
T1053.005.md	Generate docs from job=generate_and_commit_guids_and_docs branch=mast…	2 months ago
T1053.005.yaml	Updating atomics to conform to standard (#1619)	2 months ago

redcanaryco / atomic-red-team Public

264 lines (138 sloc) | 7.49 KB

T1053.005 - Scheduled Task

Description from ATT&CK

> Adversaries may abuse the Windows Task Scheduler to perform task scheduling for initial or recurring execution of malicious code. There are multiple ways to access the Task Scheduler in Windows. The `schtasks` can be run directly on the command line, or the Task Scheduler can be opened through the GUI within the Administrator Tools section of the Control Panel. In some cases, adversaries have used a .NET wrapper for the Windows Task Scheduler, and alternatively, adversaries have used the Windows netapi32 library to create a scheduled task.
> The deprecated at utility could also be abused by adversaries (ex: At (Windows)), though `at.exe` can not access tasks created with `schtasks` or the Control Panel.
>
> An adversary may use Windows Task Scheduler to execute programs at system startup or on a scheduled basis for persistence. The Windows Task Scheduler can also be abused to conduct remote Execution as part of Lateral Movement and or to run a process under the context of a specified account (such as SYSTEM).

Atomic Tests

- Atomic Test #1 - Scheduled Task Startup Script
- Atomic Test #2 - Scheduled task Local
- Atomic Test #3 - Scheduled task Remote
- Atomic Test #4 - Powershell Cmdlet Scheduled Task
- Atomic Test #5 - Task Scheduler via VBA
- Atomic Test #6 - WMI Invoke-CimMethod Scheduled Task

Atomic Test #1 - Scheduled Task Startup Script

Run an exe on user logon or system startup. Upon execution, success messages will be displayed for the two scheduled tasks. To view the tasks, open the Task Scheduler and look in the Active Tasks pane.

Supported Platforms: Windows

auto_generated_guid: fec27f65-db86-4c2d-b66c-61945aee87c2

master atomic-red-team / atomics / T1053.005 / **T1053.005.yaml** Go to file ···

MSAdministrator Updating atomics to conform to standard (#1619) ··· ✓ Latest commit 1513717 on Sep 3 History

10 contributors

143 lines (141 sloc) | 5.74 KB Raw Blame

```yaml
---
attack_technique: T1053.005
display_name: 'Scheduled Task/Job: Scheduled Task'
atomic_tests:
- name: Scheduled Task Startup Script
  auto_generated_guid: fec27f65-db86-4c2d-b66c-61945aee87c2
  description: |
    Run an exe on user logon or system startup.  Upon execution, success messages will be displayed for the two scheduled tasks. To view
    the tasks, open the Task Scheduler and look in the Active Tasks pane.
  supported_platforms:
  - windows
  executor:
    command: |
      schtasks /create /tn "T1053_005_OnLogon" /sc onlogon /tr "cmd.exe /c calc.exe"
      schtasks /create /tn "T1053_005_OnStartup" /sc onstart /ru system /tr "cmd.exe /c calc.exe"
    cleanup_command: |
      schtasks /delete /tn "T1053_005_OnLogon" /f >nul 2>&1
      schtasks /delete /tn "T1053_005_OnStartup" /f >nul 2>&1
    name: command_prompt
    elevation_required: true
- name: Scheduled task Local
  auto_generated_guid: 42f53695-ad4a-4546-abb6-7d837f644a71
  description: |
    Upon successful execution, cmd.exe will create a scheduled task to spawn cmd.exe at 20:10.
  supported_platforms:
    - windows
```

Go to file

9 on Sep 3 History

6 months ago

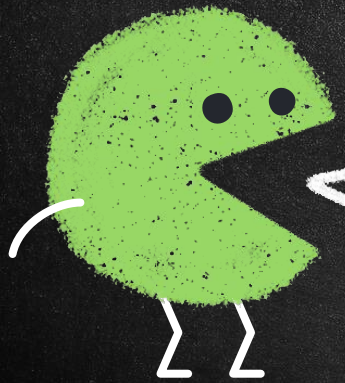
2 months ago

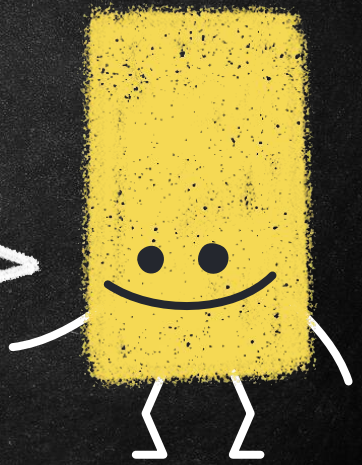
2 months ago

What to learn from here?

After class on your own?

Helpful References to learn more





Additional resources

Black Hat Go is a great book to continue on from here. WIth the foundational knowledge from this course you can more easily learn the tools you create in the book.

- ➜ Remote APIs
- ➜ DNS server and Proxy
- ➜ Fuzzer

https://nostarch.com/blackhatgo

Additional resources

Interfaces are important to understand:

- https://medium.com/rungo/interfaces-in-go-ab1601159b3a

Channels:

- https://medium.com/rungo/anatomy-of-channels-in-go-concurrency-in-go-1ec336086adb

Digital Ocean - group of tutorials

https://www.digitalocean.com/community/tutorial_series/how-to-code-in-go

The Go playground - run Go in a browser

https://play.golang.org/

Atomic Red Team Training - Basics and Beyond

https://www.youtube.com/watch?v=d_E-hfKQ5Hw

Atomic Red Team: Testing Your Defenses on MacOS and Linux
https://www.youtube.com/watch?v=yQkoYdEEXv0