Chain of Infection Detection

A Hands-On Workshop on Forensic Timeline Analysis

$whoami

- Sr. Security Research Engineer @Qualys
- Talk to me about
  - Security
  - Forensics
- Full-time Foodie
  - Please share recommendations
- Anime Enthusiast

Agenda

Timeline Analysis for Forensics

Intro to Go

Today’s Problem Statement

Caveats

QA

Timeline Analysis for Forensics

- What is Timeline Analysis?
- What are some Use Cases?
- Artefact of Need
- What is a Timestamp?
- Types of Timestamps
- How to create a Timeline?

What is Timeline Analysis?

Process of re-constructing chain of events to pinpoint initial attack vector, movement of adversary/malware, end goal or point of detection.

Stages of a Timeline Analysis

- Collection
- Normalization
- Correlation
- Interpretation

Collection – Sources Gathered

Event logs from the suspect’s workstation

File system metadata (timestamps)

Email server logs

VPN access logs

USB device connection logs

Collection – Potential Key Finding

USB device connected at 10:55 PM

Normalization – Action Taken

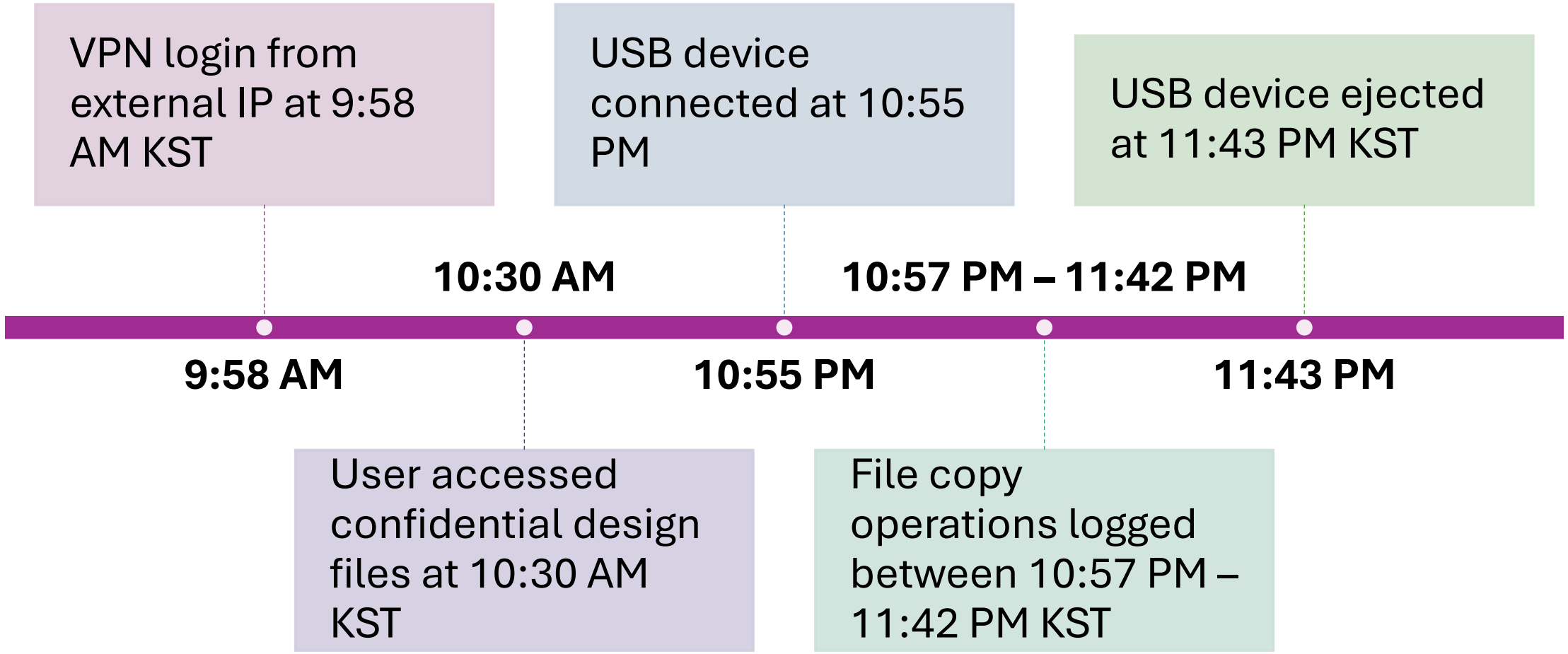
Timestamps converted into UTC

Logs from Windows and Linux parsed into a unified schema

Normalization – Result

A normalized dataset where every event is timestamped and categorized (login, file access, etc.) in a common format.

Correlation – Cross Referenced Events



Correlation – Insight

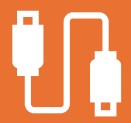
The sequence of events strongly indicates targeted data exfiltration by the person of interest:

During normal working hours: The individual accessed and flagged high-value files, identifying assets of interest.

Late at night: The same user connected a USB device and copied the previously identified files; executing the exfiltration outside regular monitoring windows.

This deliberate separation of reconnaissance and extraction is a classic indicator of insider threat behavior.

Interpretation – Conclusion Drawn



The person of interest remotely accessed the network, logged in, accessed sensitive files, and transferred them to a USB drive.

The timeline supports the hypothesis of intentional data exfiltration.

Some More Examples

- Unexpected Login → Financial File Access → Large DNS Requests
- File Execution → Prefetch Creation
- USB Insertion → Process Creation
- Browser → Download File → Filesystem Write

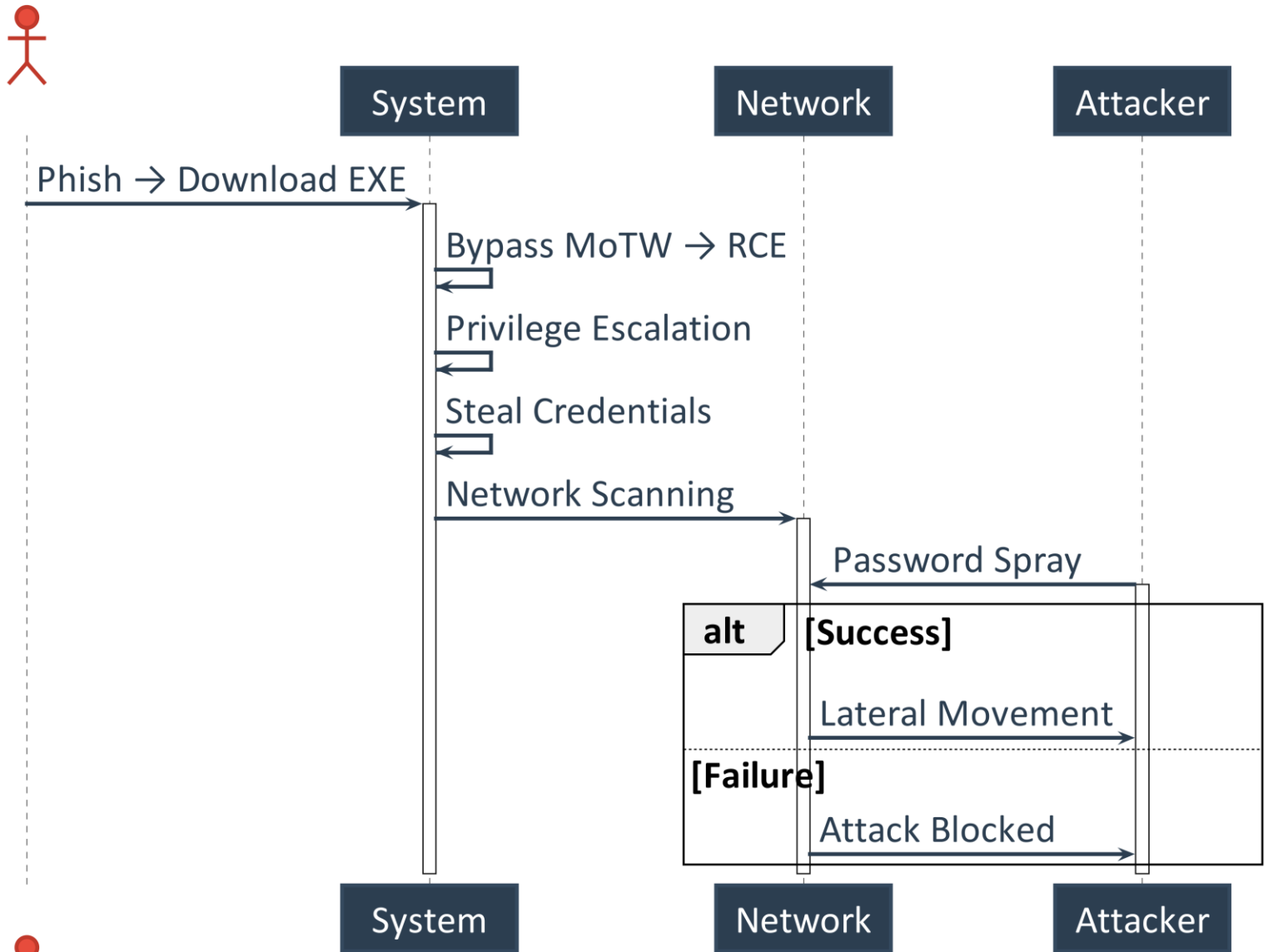
What are some Use Cases?

- Entry-point Discovery
- Persistence Detection
- Lateral Movement Mapping
- IoC Creation & Movement Tracking

Sample Malware Attack Sequence

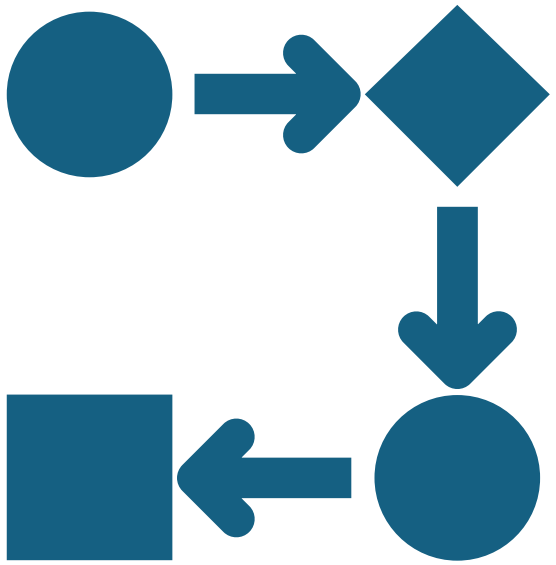
1. Phishing Email Received
2. Email Downloads EXE
3. EXE Bypasses MoTW (Mark of The Web)
4. Remote Code Execution Achieved
5. Exploits CLFS to achieve Local Privilege Escalation
6. Leaks NTLM Hashes
7. Monitors Local IPs for RDP/SSH Access
8. Sprays Passwords Until Successful / Complete Failure

Malware Attack Chain



Artefact of Need

Timestamps



What is a Timestamp?

A sequence of characters that precisely records the date and time an event occurred, typically down to the second or even millisecond.

Great. But... can you give us an example?

Sure, here's one (unix epoch)

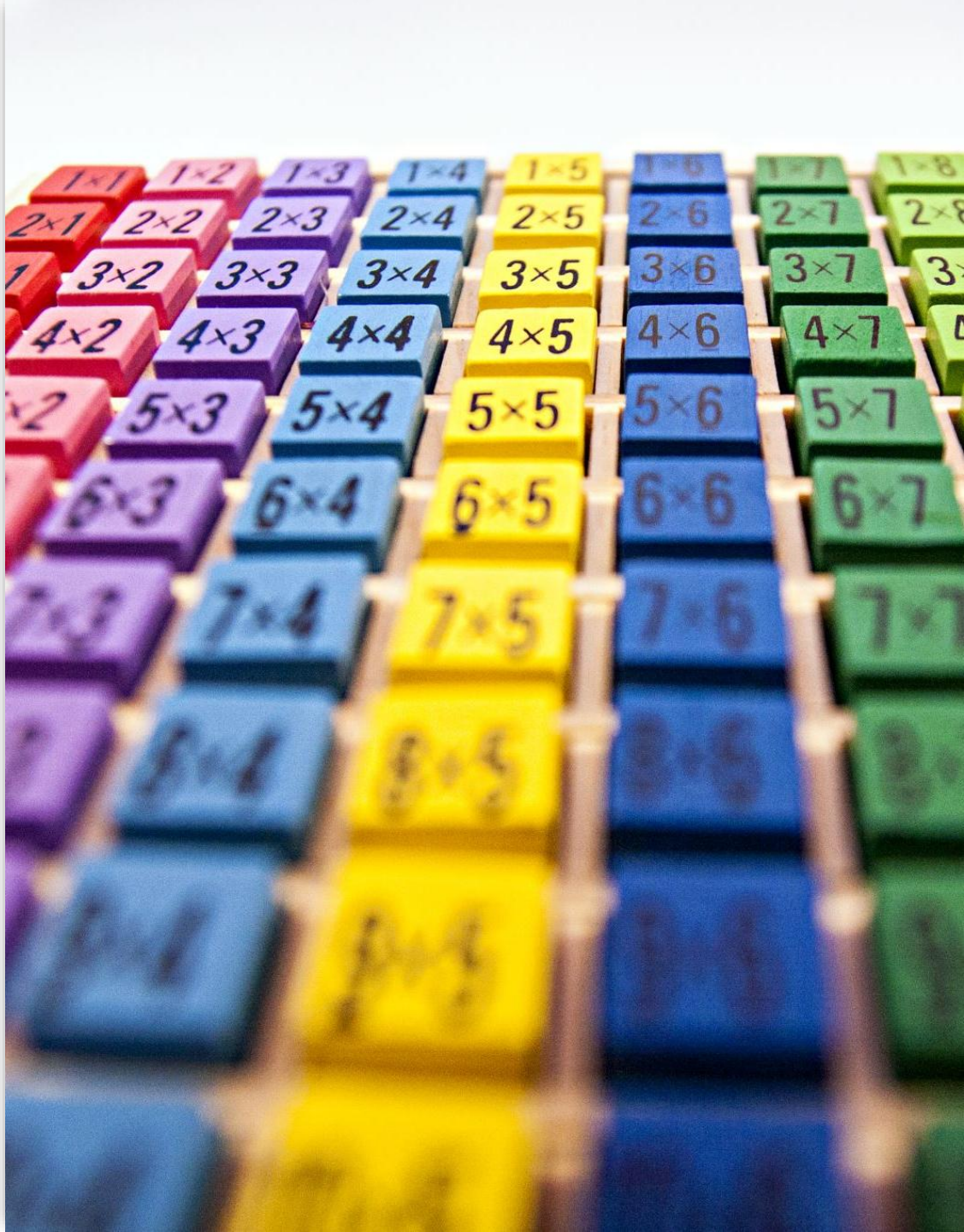
Component	Value	Explanation
Raw Timestamp	1633072800	Total seconds since Unix epoch (1 Jan 1970, 00:00:00 UTC)
Time Zone	UTC	Unix epoch timestamps are always in Coordinated Universal Time
Converted Date	2021-10-01	The calendar date in YYYY-MM-DD format
Milliseconds	1633072800000	To convert to millisecond precision, multiply by 1000

But wait… there’s more

We’ll discuss different types of timestamps based on various classifications, next.

Types of Timestamps

- Classification by Format
- Classification by Semantics
- Classification by Source



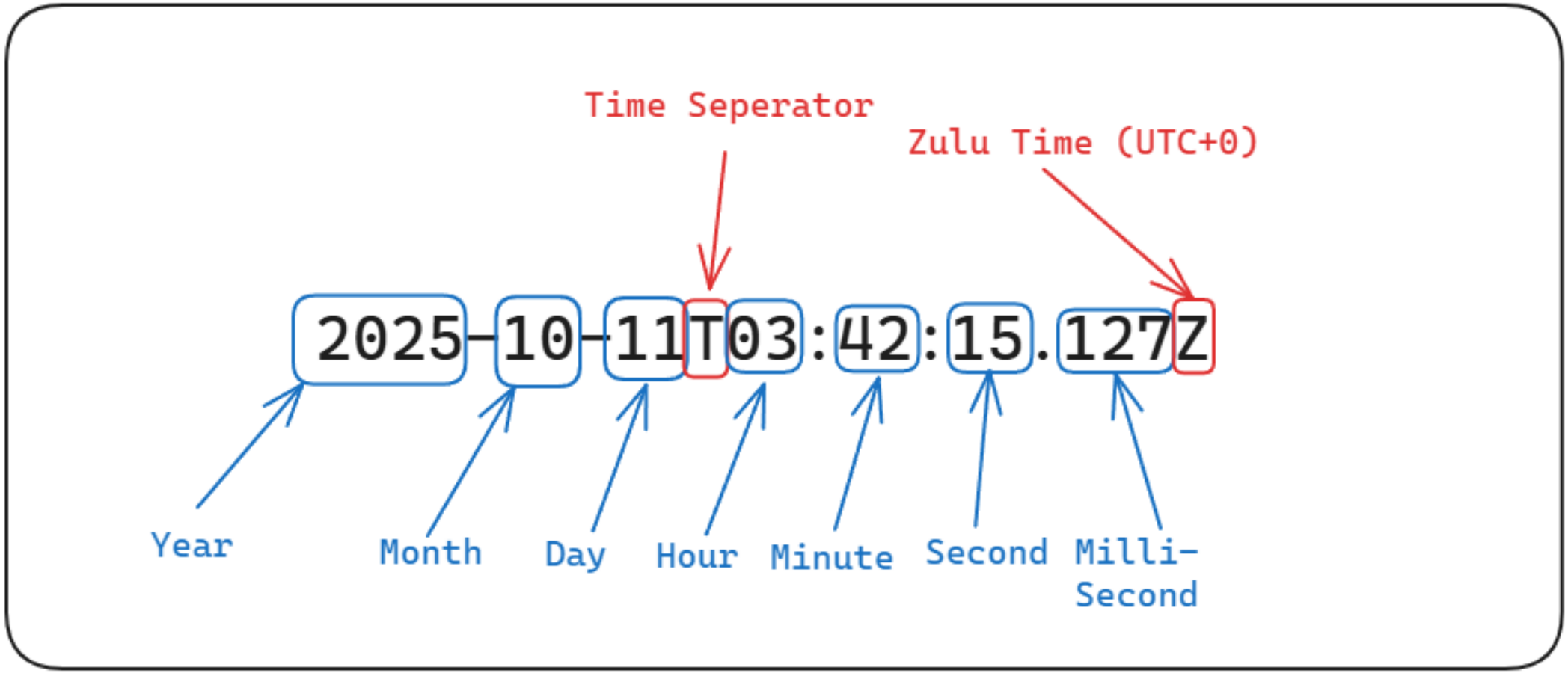
Classification by Format

- ISO 8601
- Unix Epoch
- Windows Filetime
- RFC 3339

ISO 8601

- An international standard for representing dates and times in a consistent and unambiguous way, using the order of year, month, day, hour, minute, and second.
- Often found in system / app logs
- Lower-level mechanisms don’t use this

ISO 8601


Time Seperator
Zulu Time (UTC+0)
2025-10-11T03:42:15.127Z
Year
Month
Day
Hour
Minute
Second
Milli-
Second

Unix Epoch

This timestamp represents the number of seconds since 00:00:00 UTC on 1 January 1970.

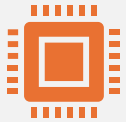
Always in UTC.

Used by system drivers / file systems and low-level entities.

Unix Epoch

Component	Value	Explanation
Raw Timestamp	1633072800	Total seconds since Unix epoch (1 Jan 1970, 00:00:00 UTC)
Time Zone	UTC	Unix epoch timestamps are always in Coordinated Universal Time
Converted Date	2021-10-01	The calendar date in YYYY-MM-DD format
Milliseconds	1633072800000	To convert to millisecond precision, multiply by 1000

Windows Filetime



A 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 A.M. January 1, 1601.

Always in UTC.

Minimum supported OS: Windows 2000 / Windows Server 2000

Windows Filetime

- 0x01D7B6C0D53E8000
- A 64-bit value made of 2 32-bit parts

Component	Hex	Decimal
dwHighDateTime	0x01D7B6C0	30914240
dwLowDateTime	0xD53E8000	3577643008

RFC 3339

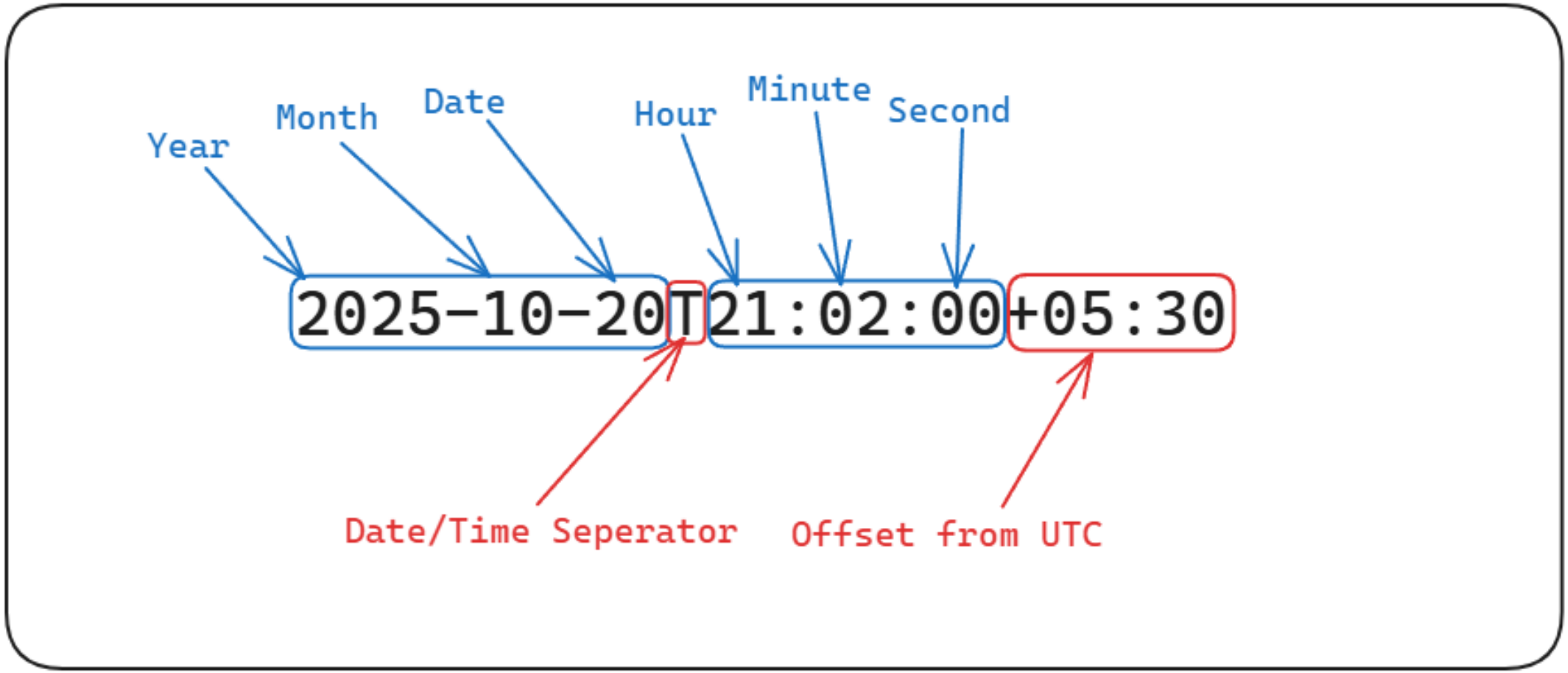
Formalises ISO 8601 with a more restricted definition.

All dates and times are assumed to be in the "current era", somewhere between 0000AD and 9999AD.

All times expressed have a stated relationship (offset) to Coordinated Universal Time (UTC).

Timestamps can express times that occurred before the introduction of UTC.

RFC 3339


Year
Month
Date
Hour
Minute
Second
2025-10-20T21:02:00+05:30
Date/Time Seperator
Offset from UTC

Classification by Semantics

Creation Time	Modification Time	Access Time	Metadata Change
Deletion	Installation	Download	Print Time

Classification by Source

FILESYSTEM

WINDOWS EVENT LOGS

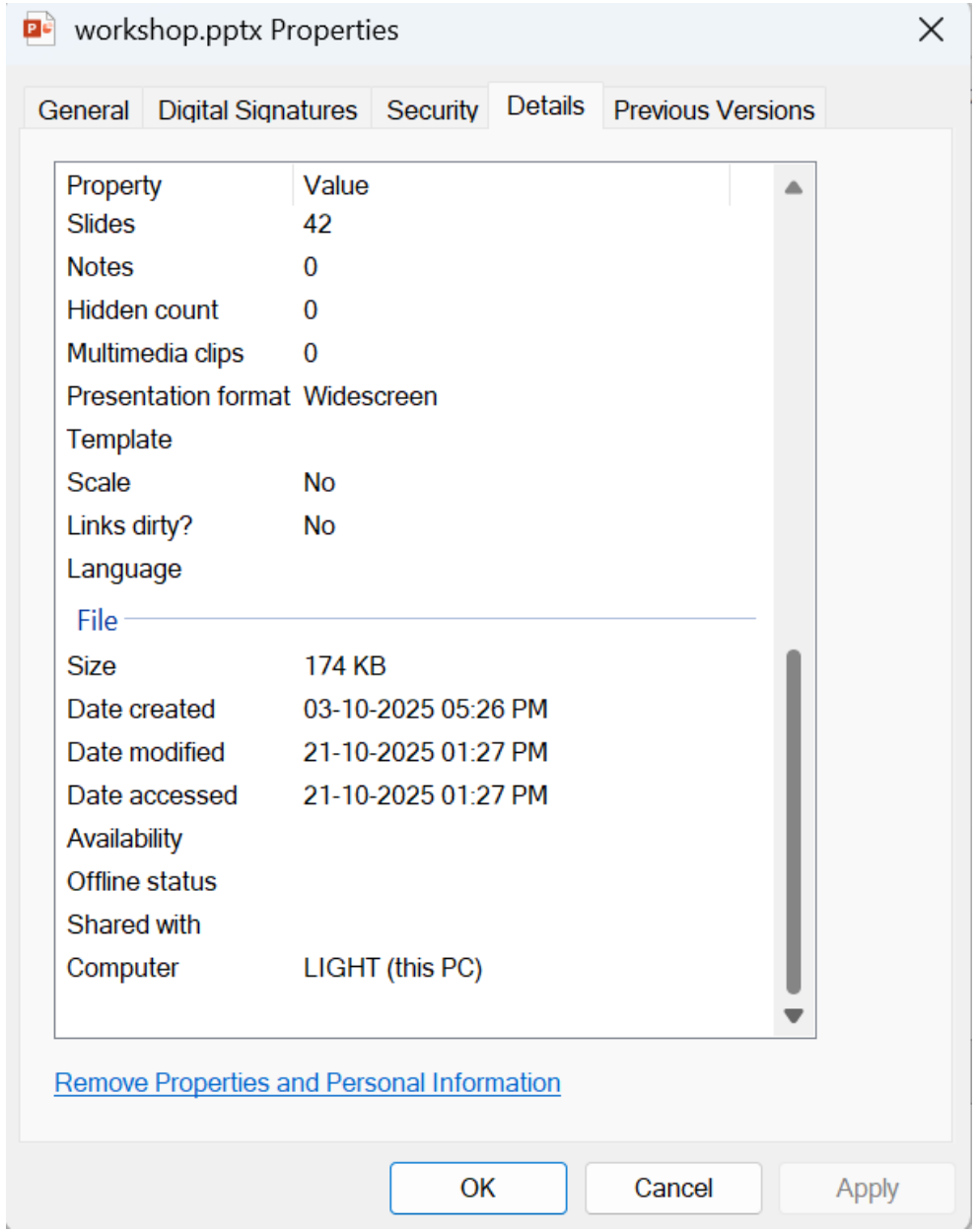
PREFETCH FILES

FileSystem

As the name suggests, these timestamps are classified for files found on the file system

Different file systems store timestamps with varying levels of accuracy

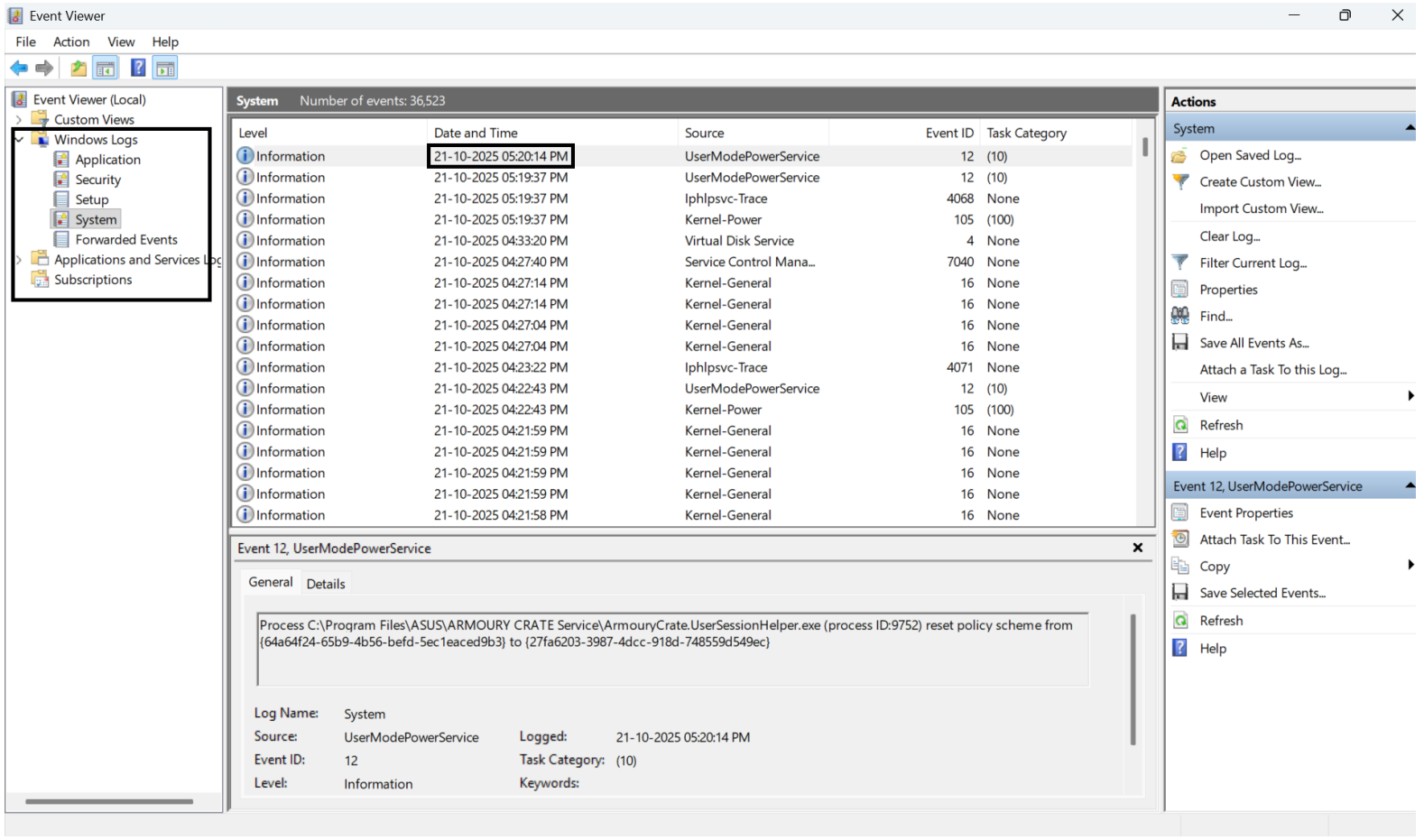
FileSystem


workshop.pptx Properties
General
Digital Signatures
Security
Details
Previous Versions
Property
Value
Slides
42
Notes
0
Hidden count
0
Multimedia clips
0
Presentation format
Widescreen
Template
Scale
No
Links dirty?
No
Language
File
Size
174 KB
Date created
03-10-2025 05:26 PM
Date modified
21-10-2025 01:27 PM
Date accessed
21-10-2025 01:27 PM
Availability
Offline status
Shared with
Computer
LIGHT (this PC)
Remove Properties and Personal Information
OK
Cancel
Apply

Event Logs

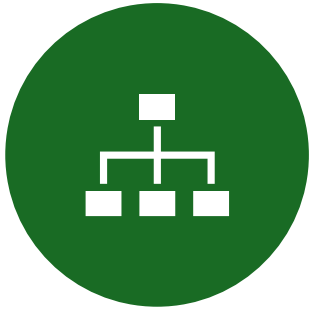
Event logs are detailed records generated by the operating system that track system activities, security events, and application behavior, providing valuable insights for troubleshooting and monitoring.

Event Logs


Event Viewer
File Action View Help
Event Viewer (Local)
Custom Views
Windows Logs
Application
Security
Setup
System
Forwarded Events
Applications and Services Lo
Subscriptions
System Number of events: 36,523
Level Date and Time Source Event ID Task Category
Information 21-10-2025 05:20:14 PM UserModePowerService 12 (10)
Information 21-10-2025 05:19:37 PM UserModePowerService 12 (10)
Information 21-10-2025 05:19:37 PM IphlpsvcTrace 4068 None
Information 21-10-2025 05:19:37 PM Kernel-Power 105 (100)
Information 21-10-2025 04:33:20 PM Virtual Disk Service 4 None
Information 21-10-2025 04:27:40 PM Service Control Mana... 7040 None
Information 21-10-2025 04:27:14 PM Kernel-General 16 None
Information 21-10-2025 04:27:14 PM Kernel-General 16 None
Information 21-10-2025 04:27:04 PM Kernel-General 16 None
Information 21-10-2025 04:27:04 PM Kernel-General 16 None
Information 21-10-2025 04:23:22 PM IphlpsvcTrace 4071 None
Information 21-10-2025 04:22:43 PM UserModePowerService 12 (10)
Information 21-10-2025 04:22:43 PM Kernel-Power 105 (100)
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:58 PM Kernel-General 16 None
Event 12, UserModePowerService
General Details
Process C:\Program Files\ASUS\ARMOURY CRATE Service\ArmouryCrate.UserSessionHelper.exe (process ID:9752) reset policy scheme from {64a64f24-65b9-4b56-befd-5ec1eaced9b3} to {27fa6203-3987-4dcc-918d-748559d549ec}
Log Name: System
Source: UserModePowerService
Logged: 21-10-2025 05:20:14 PM
Event ID: 12
Task Category: (10)
Level: Information
Keywords:
Actions
System
Open Saved Log...
Create Custom View...
Import Custom View...
Clear Log...
Filter Current Log...
Properties
Find...
Save All Events As...
Attach a Task To this Log...
View
Refresh
Help
Event 12, UserModePowerService
Event Properties
Attach Task To This Event...
Copy
Save Selected Events...
Refresh
Help

Some Interesting Windows Events

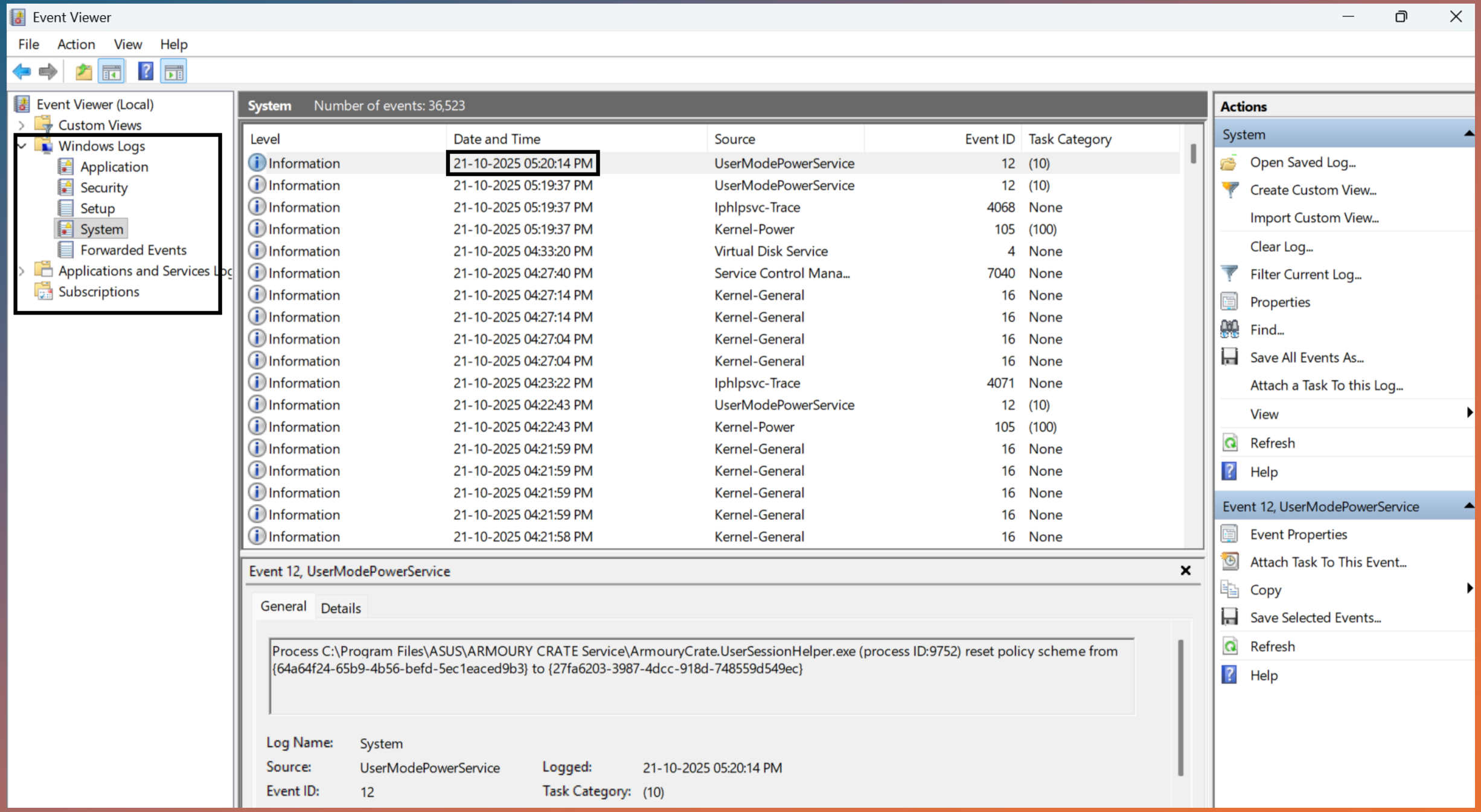
4625 – LOGON



4688 – PROCESS CREATION

4104 – POWERSHELL

7036 – SERVICE START/STOP


Event Viewer
File Action View Help
Event Viewer (Local)
Custom Views
Windows Logs
Application
Security
Setup
System
Forwarded Events
Applications and Services Log
Subscriptions
System Number of events: 36,523
Level Date and Time Source Event ID Task Category
Information 21-10-2025 05:20:14 PM UserModePowerService 12 (10)
Information 21-10-2025 05:19:37 PM UserModePowerService 12 (10)
Information 21-10-2025 05:19:37 PM Iphlpsvc-Trace 4068 None
Information 21-10-2025 05:19:37 PM Kernel-Power 105 (100)
Information 21-10-2025 04:33:20 PM Virtual Disk Service 4 None
Information 21-10-2025 04:27:40 PM Service Control Mana... 7040 None
Information 21-10-2025 04:27:14 PM Kernel-General 16 None
Information 21-10-2025 04:27:14 PM Kernel-General 16 None
Information 21-10-2025 04:27:04 PM Kernel-General 16 None
Information 21-10-2025 04:27:04 PM Kernel-General 16 None
Information 21-10-2025 04:23:22 PM Iphlpsvc-Trace 4071 None
Information 21-10-2025 04:22:43 PM UserModePowerService 12 (10)
Information 21-10-2025 04:22:43 PM Kernel-Power 105 (100)
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:59 PM Kernel-General 16 None
Information 21-10-2025 04:21:58 PM Kernel-General 16 None
Event 12, UserModePowerService
General Details
Process C:\Program Files\ASUS\ARMOURY CRATE Service\ArmouryCrate.UserSessionHelper.exe (process ID:9752) reset policy scheme from {64a64f24-65b9-4b56-befd-5ec1eaced9b3} to {27fa6203-3987-4dcc-918d-748559d549ec}
Log Name: System
Source: UserModePowerService
Logged: 21-10-2025 05:20:14 PM
Event ID: 12
Task Category: (10)
Actions
System
Open Saved Log...
Create Custom View...
Import Custom View...
Clear Log...
Filter Current Log...
Properties
Find...
Save All Events As...
Attach a Task To this Log...
View
Refresh
Help
Event 12, UserModePowerService
Event Properties
Attach Task To This Event...
Copy
Save Selected Events...
Refresh
Help

Prefetch Files

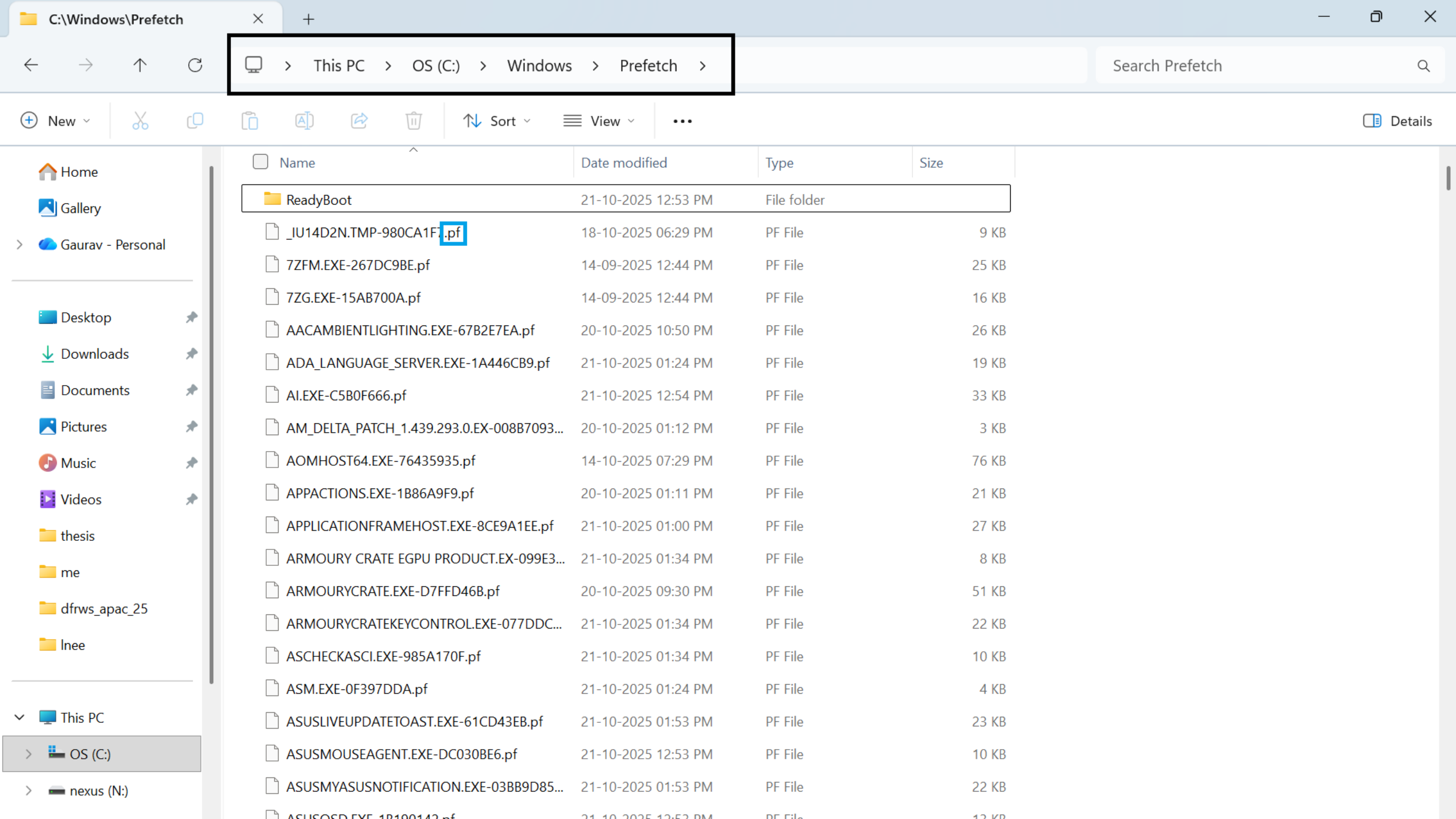
Introduced in Windows XP

Prefetch files are created to speed up the loading time of applications by caching the necessary data for frequently used programs.

Max Prefetch Files

Windows XP to 7 = 128	Windows 8 to 10 = 1024	Windows 11 = 8192

Stored in the C:/Windows/Prefetch.


C:\Windows\Prefetch
This PC
OS (C:)
Windows
Prefetch
Search Prefetch
New
Sort
View
Details
Home
Gallery
Gaurav - Personal
Desktop
Downloads
Documents
Pictures
Music
Videos
thesis
me
dfrws_apac_25
Inee
This PC
OS (C:)
nexus (N:)
Name
Date modified
Type
Size
ReadyBoot 21-10-2025 12:53 PM File folder
_IU14D2N.TMP-980CA1F7.pf 18-10-2025 06:29 PM PF File 9 KB
7ZFM.EXE-267DC9BE.pf 14-09-2025 12:44 PM PF File 25 KB
7ZG.EXE-15AB700A.pf 14-09-2025 12:44 PM PF File 16 KB
AACAMBIENTLIGHTING.EXE-67B2E7EA.pf 20-10-2025 10:50 PM PF File 26 KB
ADA_LANGUAGE_SERVER.EXE-1A446CB9.pf 21-10-2025 01:24 PM PF File 19 KB
AI.EXE-C5B0F666.pf 21-10-2025 12:54 PM PF File 33 KB
AM_DELTA_PATCH_1.439.293.0.EX-008B7093... 20-10-2025 01:12 PM PF File 3 KB
AOMHOST64.EXE-76435935.pf 14-10-2025 07:29 PM PF File 76 KB
APPACTIONS.EXE-1B86A9F9.pf 20-10-2025 01:11 PM PF File 21 KB
APPLICATIONFRAMEHOST.EXE-8CE9A1EE.pf 21-10-2025 01:00 PM PF File 27 KB
ARMOURY CRATE EGPU PRODUCT.EX-099E3... 21-10-2025 01:34 PM PF File 8 KB
ARMOURYCRATE.EXE-D7FFD46B.pf 20-10-2025 09:30 PM PF File 51 KB
ARMOURYCRATEKEYCONTROL.EXE-077DDC... 21-10-2025 01:34 PM PF File 22 KB
ASCHECKASCI.EXE-985A170F.pf 21-10-2025 01:34 PM PF File 10 KB
ASM.EXE-0F397DDA.pf 21-10-2025 01:24 PM PF File 4 KB
ASUSLIVEUPDATETOAST.EXE-61CD43EB.pf 21-10-2025 01:53 PM PF File 23 KB
ASUSMOUSEAGENT.EXE-DC030BE6.pf 21-10-2025 12:53 PM PF File 10 KB
ASUSMYASUSNOTIFICATION.EXE-03BB9D85... 21-10-2025 01:53 PM PF File 22 KB

Prefetch files can be parsed using PECmd tool

```
net6.0 .\PECmd.exe -f C:\Windows\Prefetch\EXPLORER.EXE-D5E97654.pf
PECmd version 1.5.1.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/PECmd

Command line: -f C:\Windows\Prefetch\EXPLORER.EXE-D5E97654.pf

Warning: Administrator privileges not found!

Keywords: temp, tmp

Processing C:\Windows\Prefetch\EXPLORER.EXE-D5E97654.pf

Created on: 2024-11-25 18:52:25
Modified on: 2025-10-18 13:22:20
Last accessed on: 2025-10-21 08:22:08

Executable name: EXPLORER.EXE
Hash: D5E97654
File size (bytes): 4,64,822
Version: null

Run count: 100
Last run: 2025-10-18 13:22:18
Other run times: 2025-10-18 12:58:53, 2025-10-18 08:03:31, 2025-10-13 20:11:54, 2025-10-13 20:11:54, 2025-09-23 16:22:06, 20
25-09-23 16:22:06, 2025-09-22 18:27:06

Volume information:

#0: Name: \VOLUME{01d8164791681b58-fc917b88} Serial: FC917B88 Created: 2022-01-31 02:09:24 Directories: 91 File references:
305
#1: Name: \VOLUME{01da522087d5e8e9-08880acf} Serial: 8880ACF Created: 2024-01-28 19:31:03 Directories: 0 File references: 0

Directories referenced: 91
```

MORE Sources for Timestamps!

Registry Hives

- LastWrite Time
- UserAssists
- MRU Lists
- Run MRU
- RecentDocs
- ShellBags
- USBSTOR

How to Create a Timeline?

- Gather the Artefacts
  - Logs
  - EVTX Files
  - Prefetch Files
  - Suspected Files
- Parse Timestamps
- Map Timestamps to Artefacts to Origin
- Generate Report / Visualise

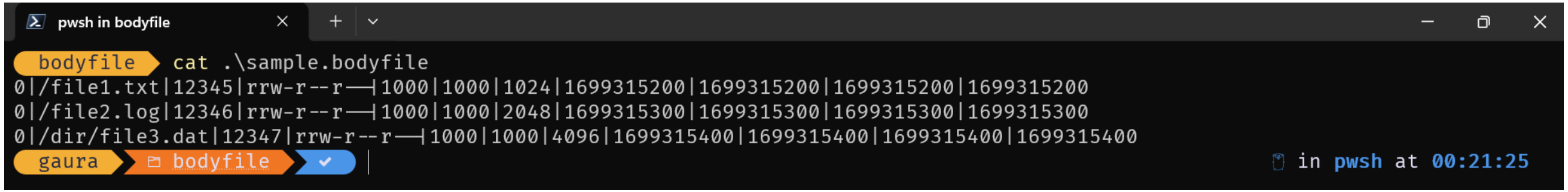
BONUS

BODYFILE

BODYFILE

- The body file format is a delimiter-separated output timeline format (as far as known) introduced by the The Sleuth Kit.
- Body files are pipe (|) delimited and are referred to as an "intermediate file", as they are not sorted chronologically and are often staged for post-processing.
- Subsequent timeline sorting is done via the mactime tool.
- All times within a body file are reported in UNIX time format.

Sample BODYFILE



Tools for Forensic Timeline Analysis

Plaso

Timesketch

MFTECmd

PECmd

EvtxECmd

KAPE

Eric Zimmerman's Tools

Introduction to Go

WHAT IS GO?

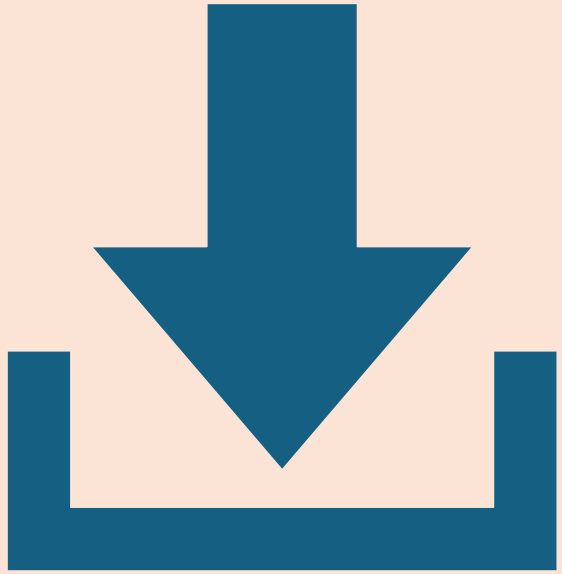
HOW TO GO?

LET'S GO

WHY GO FOR FORENSICS?

OVA Download Link –
Ubuntu24 and Win10

https://tinyurl.com/v4ak7k6



What is Go?

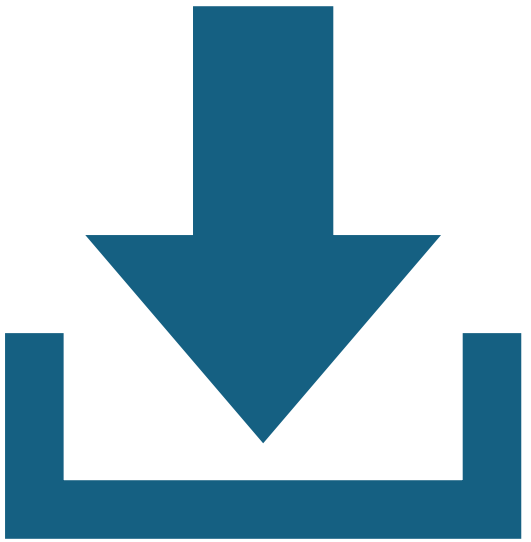
- Open source, garbage collected programming language
- Developed by Robert Griesemer, Rob Pike, and Ken Thompson at Google
- Designed for building systems tooling
- Currently used by Google, K8s, Docker, etc.

How to Go

Download & Install Go from `go.dev`

OR

Visit `play.go.dev` in your browser



For offline coding

Create a folder by name “dfrwsDemo”

Optional: run `go mod init dfrwsDemo`

Open in any text/code editor

Let’s Go

Hello World

Conditions

Loops

User Defined Functions

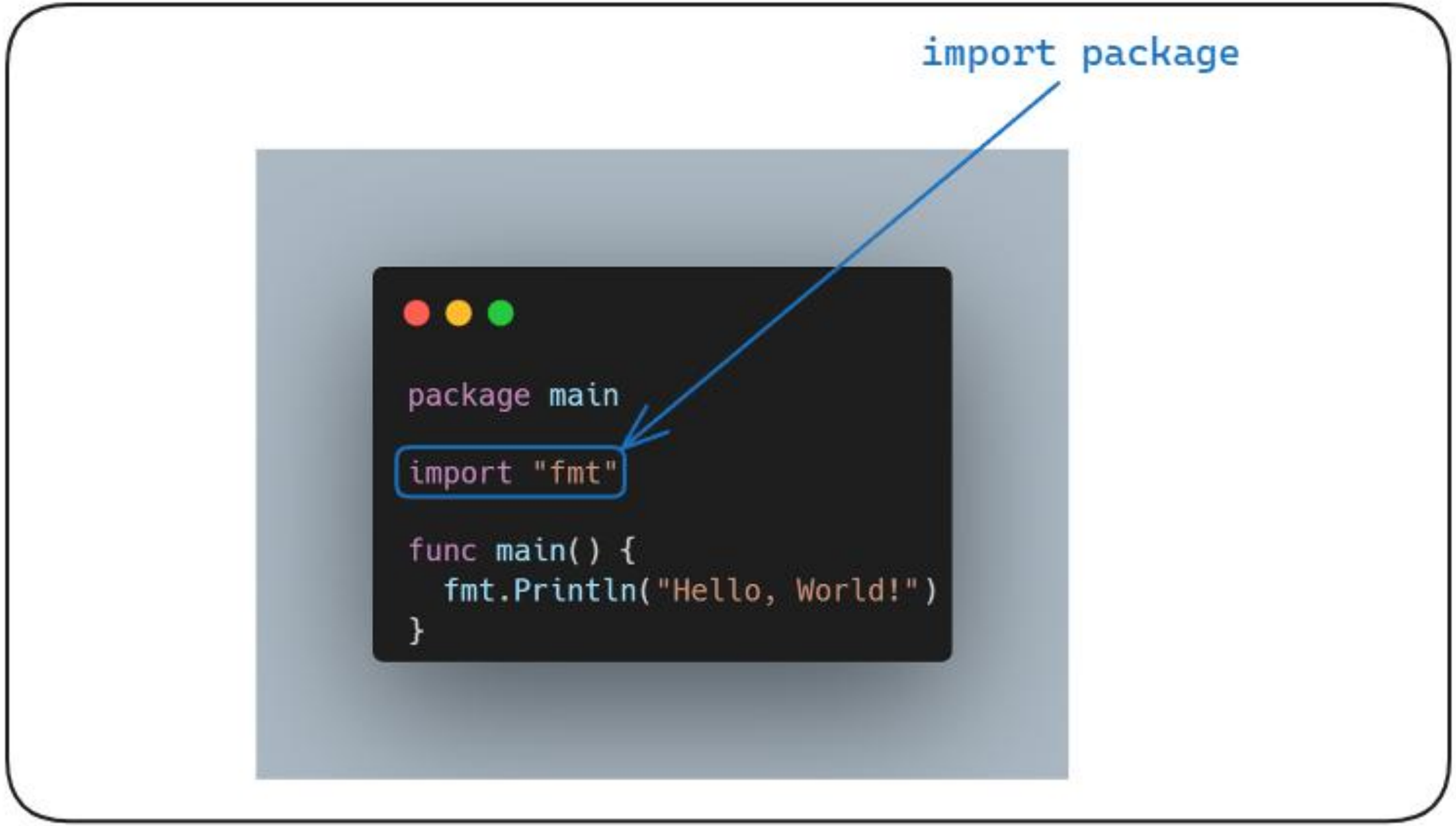
Reading Files

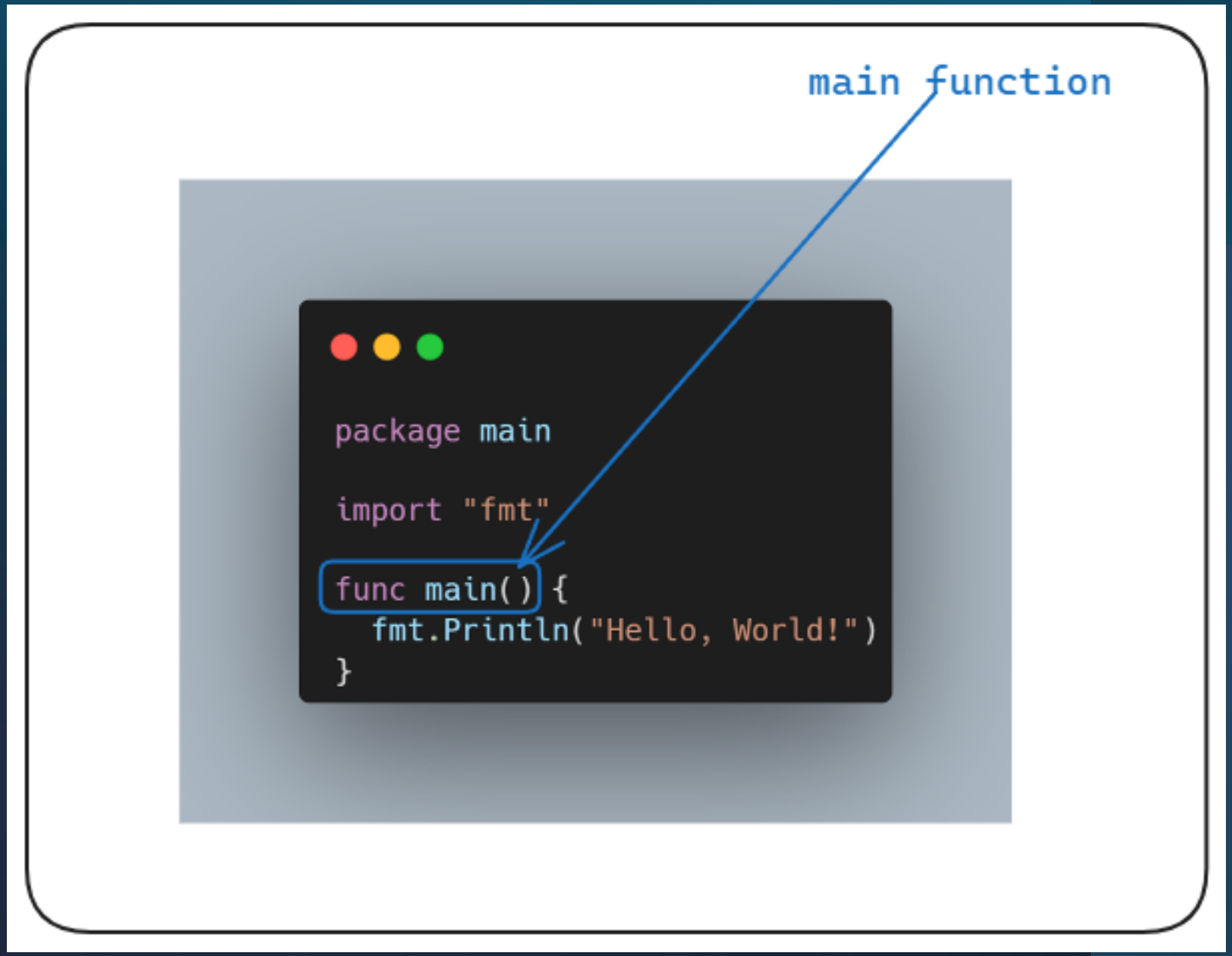
Hello World

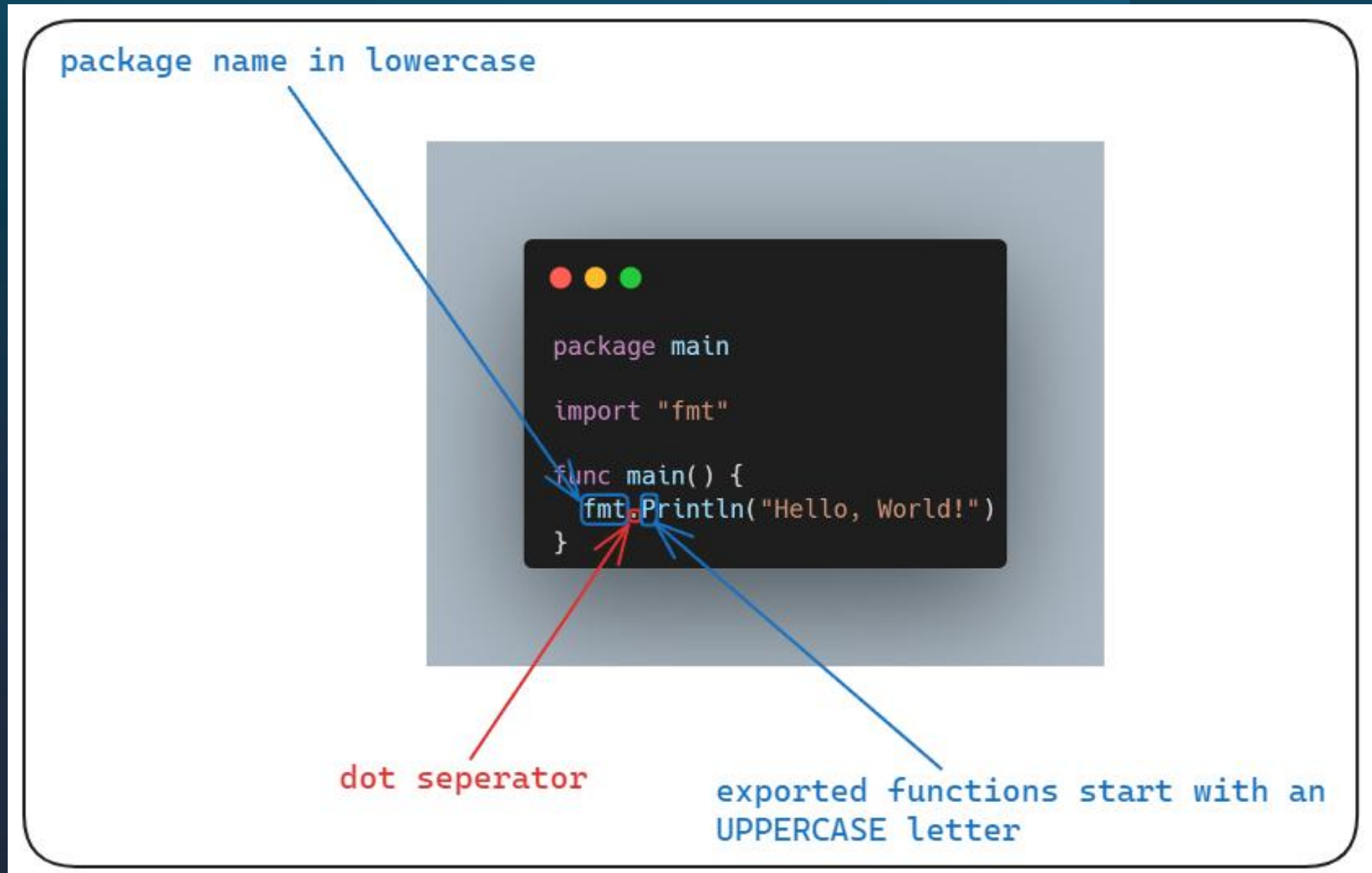
```go
package main

import "fmt"

func main() {
  fmt.Println("Hello, World!")
}
```

But before we go run, let’s unpack each line


package declaration
package main
import "fmt"
func main() {
fmt.Println("Hello, World!")
}


import package
package main
import "fmt"
func main() {
fmt.Println("Hello, World!")
}


main function
package main
import "fmt"
func main() {
fmt.Println("Hello, World!")
}

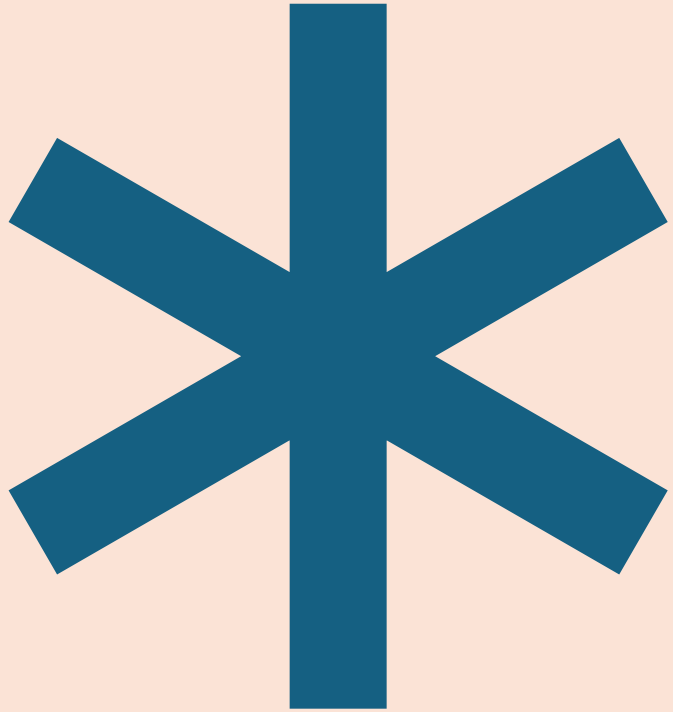

package name in lowercase
package main
import "fmt"
func main() {
fmt.Println("Hello, World!")
}
dot seperator
exported functions start with an UPPERCASE letter

no semi-colon here
we will see their use in specific cases

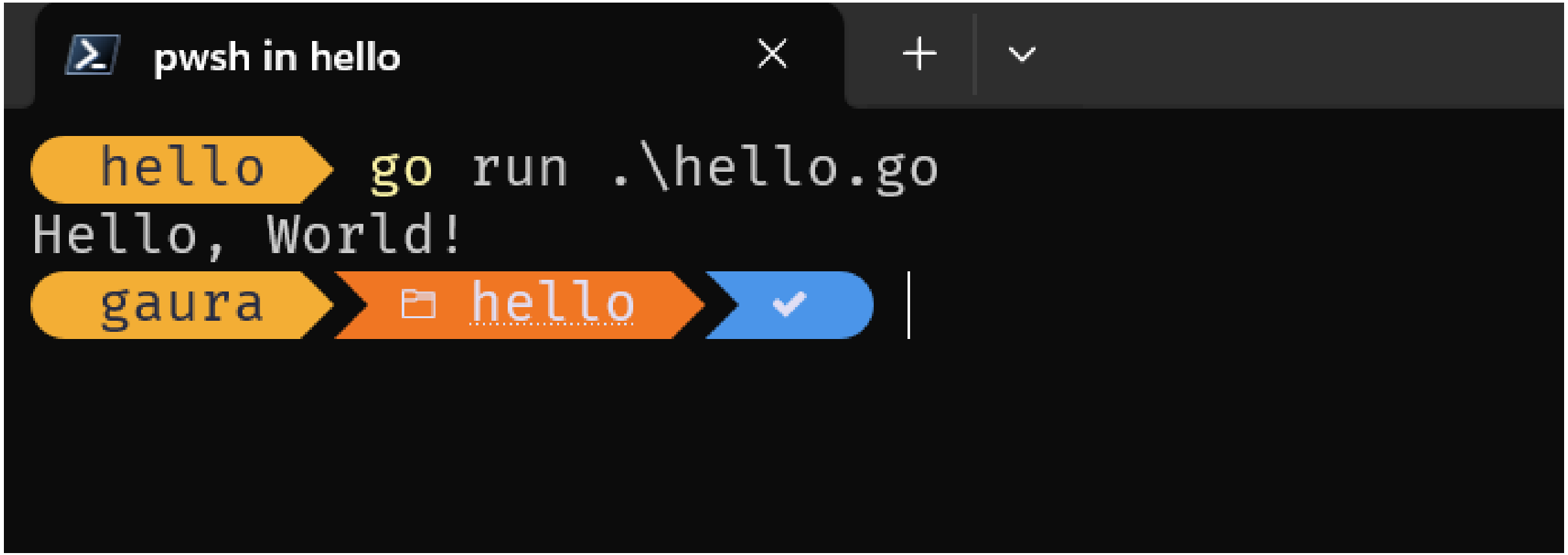
```go
package main

import "fmt"

func main() {
  fmt.Println("Hello, World!")
}
```

Run Code

go run <filename.go>



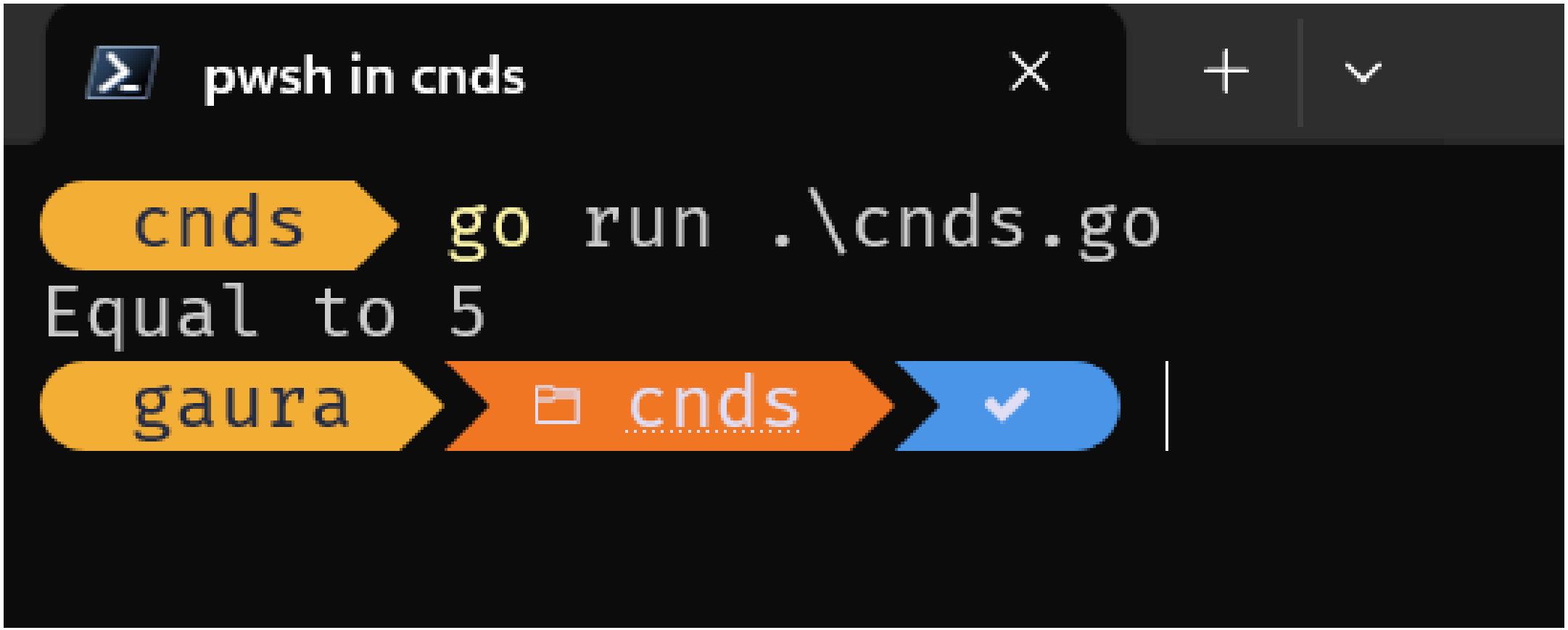
Output



Conditions

```go
package main

import "fmt"

func main() {
  var num int
  num = 5

  if num > 5 {
    fmt.Println("Greater than 5")
  } else if num < 5 {
    fmt.Println("Less than 5")
  } else {
    fmt.Println("Equal to 5")
  }
}
```

Output



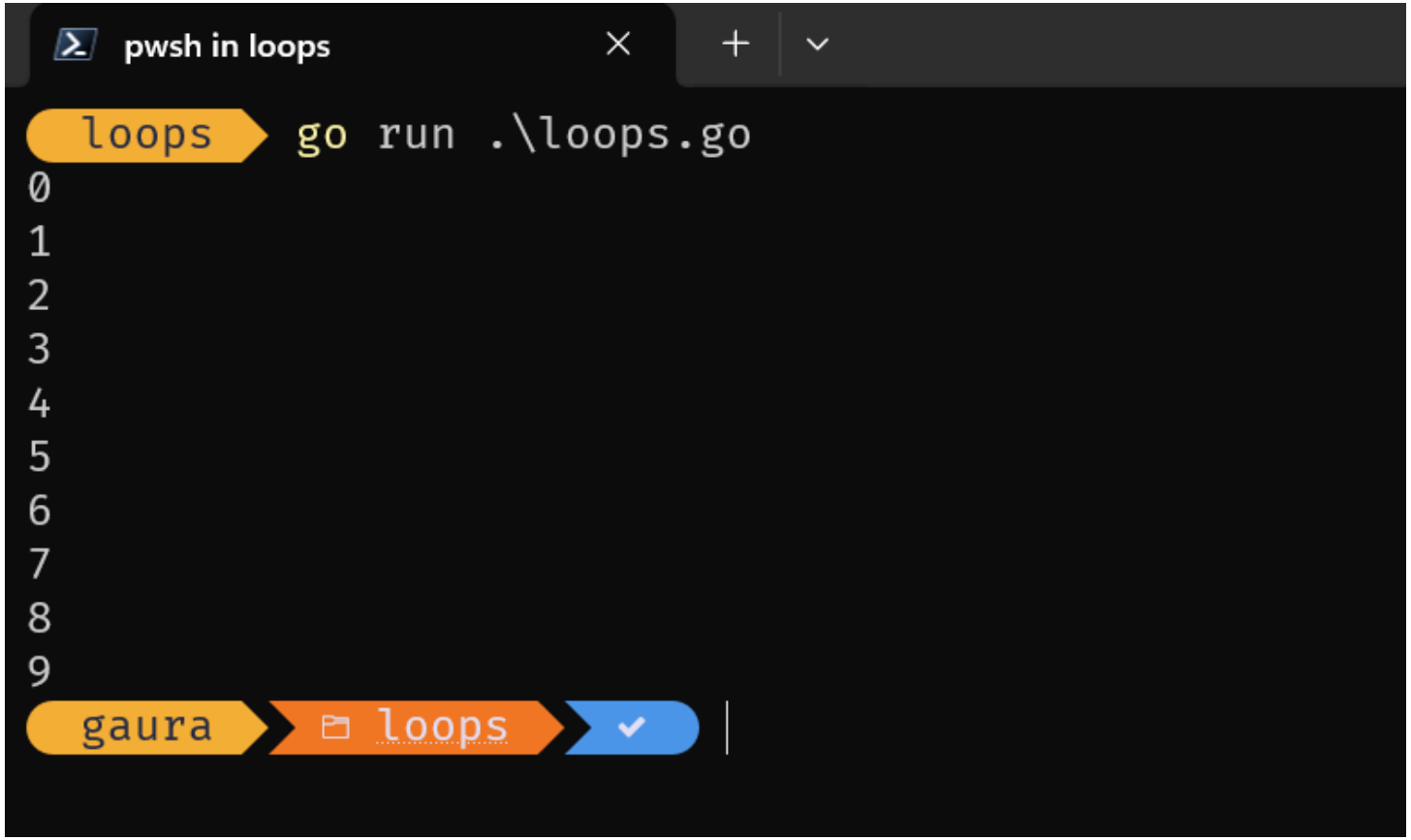
Ways to declare a variable

```go
package main

import "fmt"

func main() {
    fpath := "/some/path/to/file"

    var anotherPath string
    anotherPath  = "/some/path/foo/bar"

    const fixedPath = "/another/path/fixed"

    fmt.Println(fpath)
    fmt.Println(anotherPath)
    fmt.Println(fixedPath)
}
```

Loops

```go
package main

import "fmt"

func main() {
  for i := 0; i < 10; i++ {
    fmt.Println(i)
  }
}
```

Output



User Defined Functions

```go
package main

import "fmt"

func main() {
    var num int
    num = 5
    res := isEven(num)
    fmt.Println(res)
}

func isEven(num int) bool {
    if num%2 == 0 {
        return true
    }
    return false
}
```

Output

Reading Files - Data

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    fpath := "./go.mod"
    data, err := os.ReadFile(fpath)
    if err ≠ nil {
        panic(err)
    }
    dataStr := string(data)
    fmt.Println(dataStr)
}
```

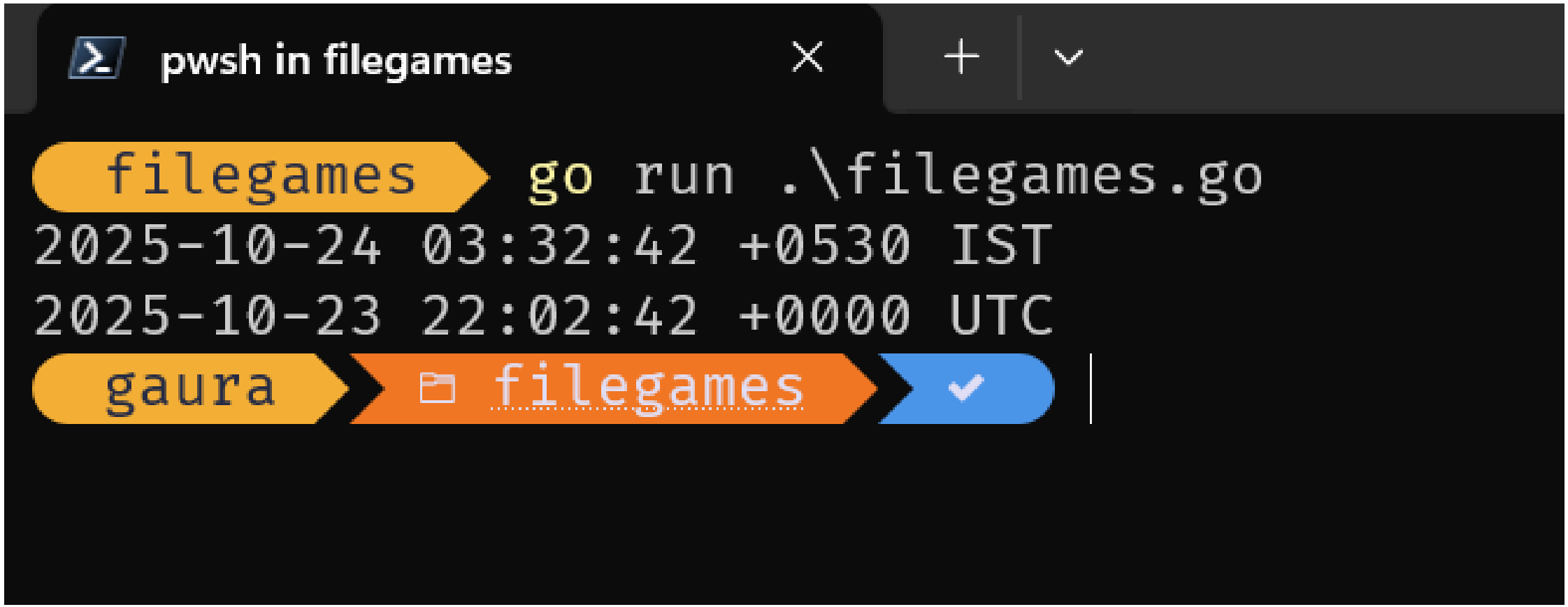
Output


pwsh in filegames
filegames
go run .\filegames.go
module filegames
go 1.25.2
gaura
filegames

Reading Files - Metadata

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    fpath := "./go.mod"
    finfo, err := os.Stat(fpath)

    if err ≠ nil {
        if os.IsNotExist(err) {
            fmt.Println("File does NOT exist")
            return
        }

        panic(err)
    }

    fmt.Println(finfo.ModTime())
    fmt.Println(finfo.ModTime().UTC())
}
```

Output

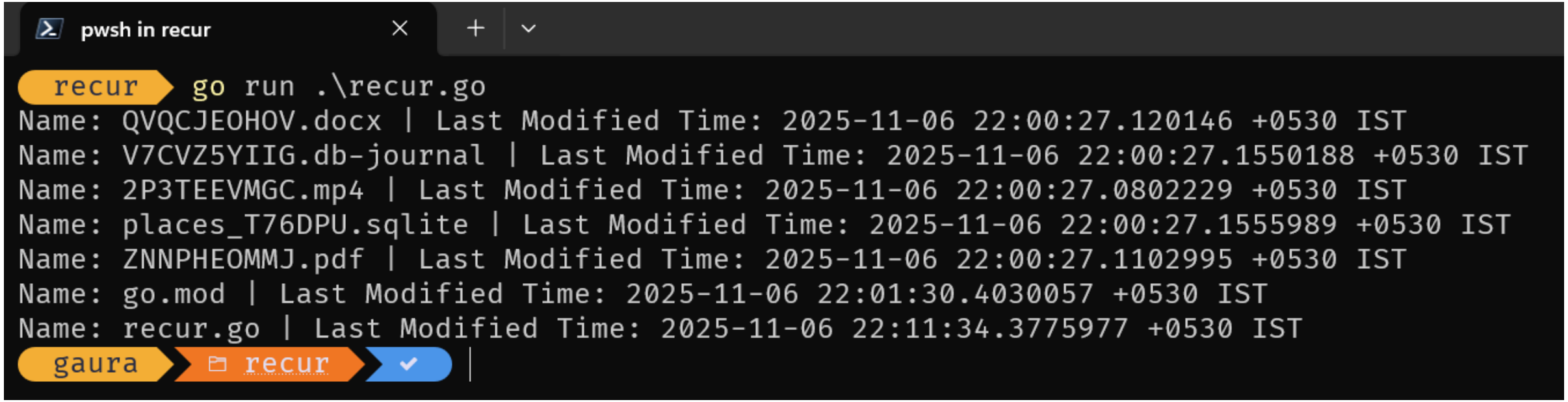

pwsh in filegames
filegames go run .\filegames.go
2025-10-24 03:32:42 +0530 IST
2025-10-23 22:02:42 +0000 UTC
gaura filegames

Recursively Finding Files

pwsh in recur

```go
package main

import (
    "fmt"
    "io/fs"
    "path/filepath"
)

func main() {
    fpath := "."

    filepath.Walk(fpath, func(path string, info fs.FileInfo, err error) error {
        if info.IsDir() {
            return nil
        }

        fmt.Printf("Name: %s | Last Modified Time: %v\n", info.Name(), info.ModTime())

        return nil
    })
}
```

Output

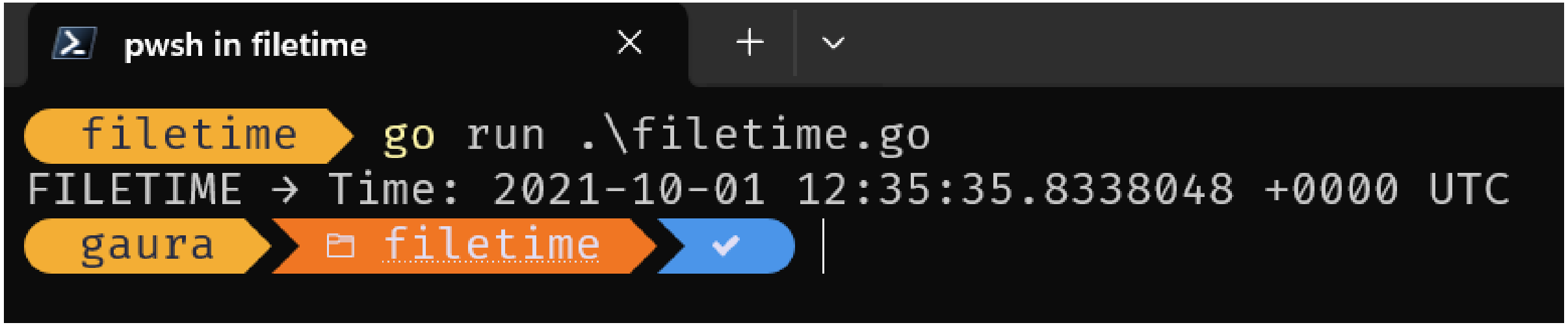

pwsh in recur
recur go run .\recur.go
Name: QVQCJEOHOV.docx | Last Modified Time: 2025-11-06 22:00:27.120146 +0530 IST
Name: V7CVZ5YIIG.db-journal | Last Modified Time: 2025-11-06 22:00:27.1550188 +0530 IST
Name: 2P3TEEVMGC.mp4 | Last Modified Time: 2025-11-06 22:00:27.0802229 +0530 IST
Name: places_T76DPU.sqlite | Last Modified Time: 2025-11-06 22:00:27.1555989 +0530 IST
Name: ZNNPHEOMMJ.pdf | Last Modified Time: 2025-11-06 22:00:27.1102995 +0530 IST
Name: go.mod | Last Modified Time: 2025-11-06 22:01:30.4030057 +0530 IST
Name: recur.go | Last Modified Time: 2025-11-06 22:11:34.3775977 +0530 IST
gaura recur

Parsing Windows FILETIME

pwsh in filetime

```go
package main

import (
    "fmt"
    "time"
)

func filetimeToTime(ft uint64) time.Time {
    const filetimeOffset = 116444736000000000 // in 100-ns ticks
    ns := int64((ft - filetimeOffset) * 100)
    return time.Unix(0, ns).UTC()
}

func main() {
    ft := uint64(0x01D7B6C0D53E8000)

    parsed := filetimeToTime(ft)
    fmt.Println("FILETIME → Time:", parsed)
}
```

Output

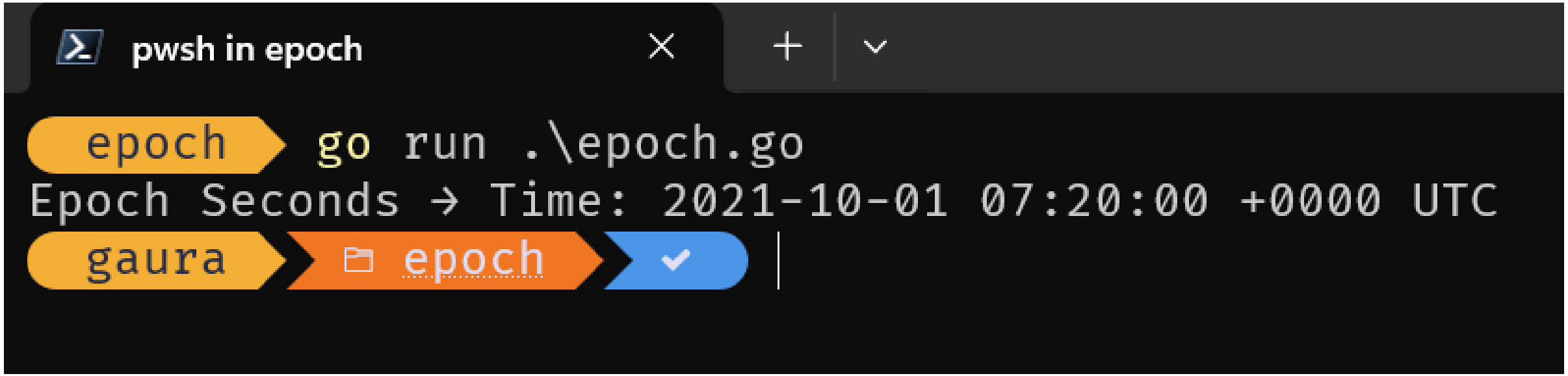

pwsh in filetime
filetime go run .\filetime.go
FILETIME → Time: 2021-10-01 12:35:35.8338048 +0000 UTC
gaura filetime

Parsing Linux EPOCH Time

pwsh in epoch

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    epoch := int64(1633072800) // Example Unix timestamp in seconds

    t := time.Unix(epoch, 0).UTC()
    fmt.Println("Epoch Seconds → Time:", t)
}
~
```

Output

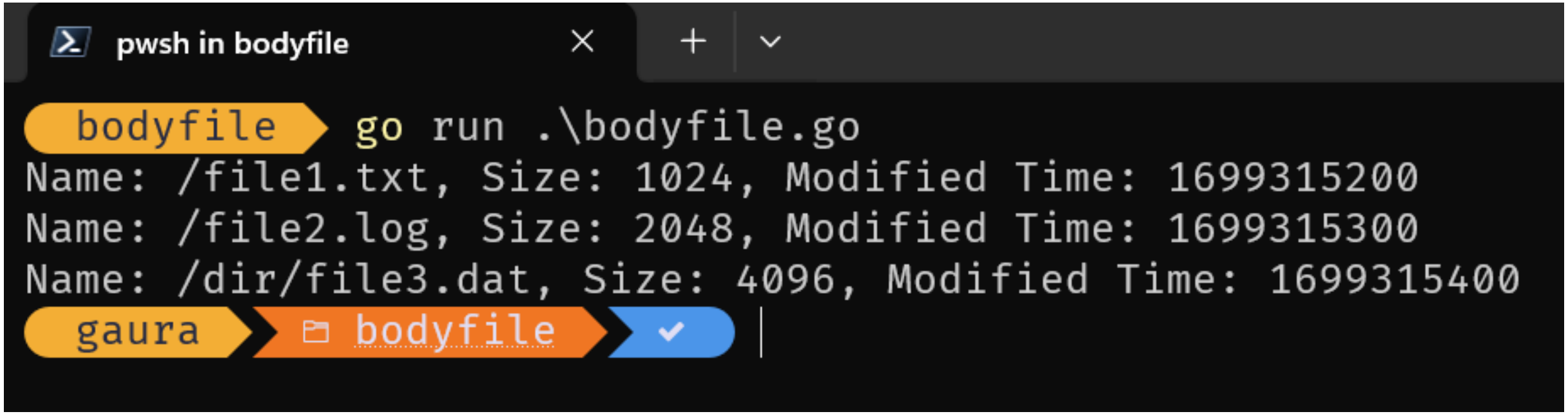

pwsh in epoch
epoch
go run .\epoch.go
Epoch Seconds → Time: 2021-10-01 07:20:00 +0000 UTC
gaura
epoch

Parsing BODYFILE

pwsh in bodyfile

```go
package main

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

func main() {
    file, err := os.Open("sample.bodyfile")
    if err ≠ nil {
        panic(err)
    }
    defer file.Close()

    scanner := bufio.NewScanner(file)
    for scanner.Scan() {
        fields := strings.Split(scanner.Text(), "|")
        fmt.Printf("Name: %s, Size: %s, Modified Time: %s\n",
            fields[1], fields[6], fields[8])
    }
}
~
```

Output

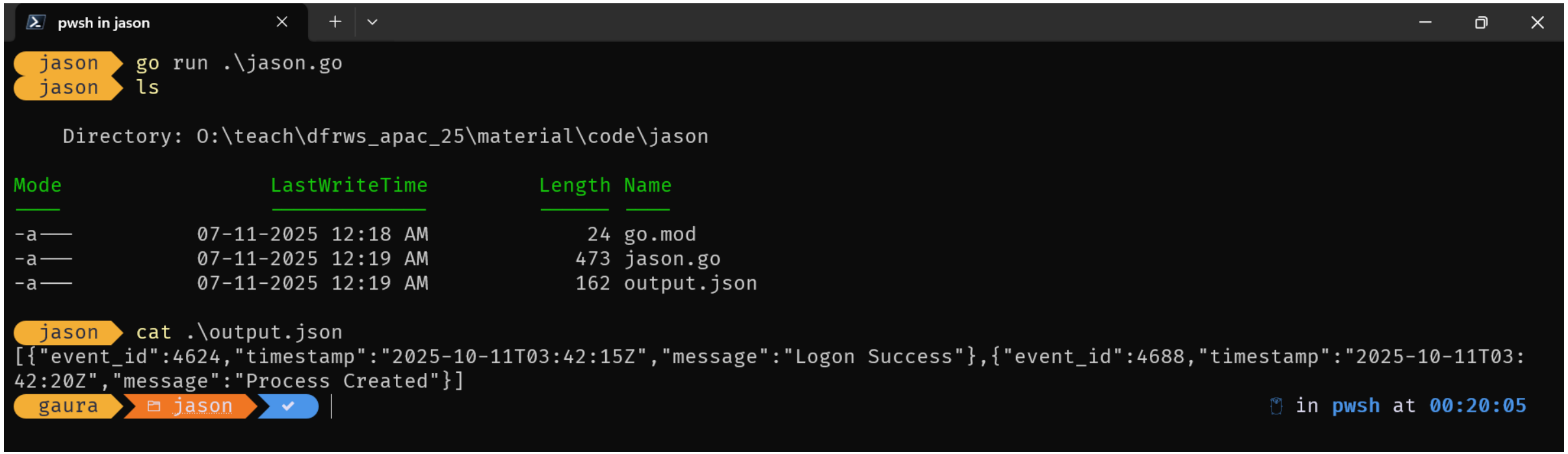

pwsh in bodyfile
bodyfile
go run .\bodyfile.go
Name: /file1.txt, Size: 1024, Modified Time: 1699315200
Name: /file2.log, Size: 2048, Modified Time: 1699315300
Name: /dir/file3.dat, Size: 4096, Modified Time: 1699315400
gaura
bodyfile

Write JSON

pwsh in jason

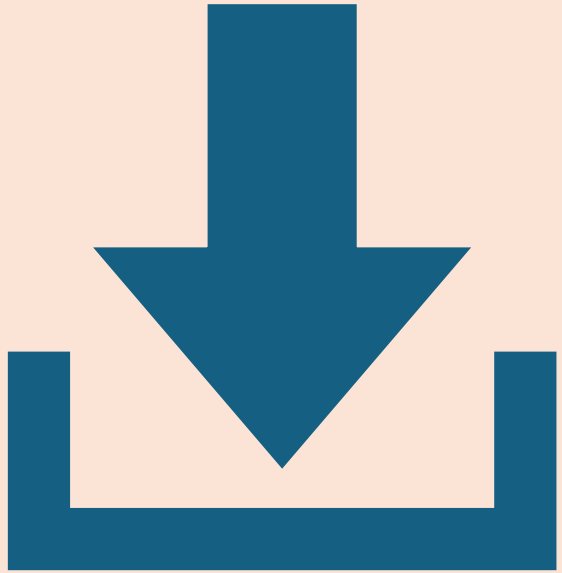
```go
package main

import (
    "encoding/json"
    "os"
)

type Event struct {
    EventID   int    `json:"event_id"`
    Timestamp string `json:"timestamp"`
    Message   string `json:"message"`
}

func main() {
    events := []Event{
        {4624, "2025-10-11T03:42:15Z", "Logon Success"},
        {4688, "2025-10-11T03:42:20Z", "Process Created"},
    }

    f, err := os.Create("output.json")
    if err ≠ nil {
        panic(err)
    }
    defer f.Close()

    json.NewEncoder(f).Encode(events)
}
```

Output


pwsh in jason
jason go run .\jason.go
jason ls
Directory: O:\teach\dfrws_apac_25\material\code\jason
Mode LastWriteTime Length Name
-a—— 07-11-2025 12:18 AM 24 go.mod
-a—— 07-11-2025 12:19 AM 473 jason.go
-a—— 07-11-2025 12:19 AM 162 output.json
jason cat .\output.json
[{"event_id":4624,"timestamp":"2025-10-11T03:42:15Z","message":"Logon Success"},{"event_id":4688,"timestamp":"2025-10-11T03:42:20Z","message":"Process Created"}]
gaura jason
in pwsh at 00:20:05

Code Download Link –
code.zip

https://tinyurl.com/v4ak7k6



Why Go for Forensics?

Go compiles to native code and runs fast, making it ideal for processing large datasets like disk images or memory dumps.

Concurrency through multi-threading is built into go through green threads called goroutines.

Cross-compilation works across all major operating systems.

Exposes C style ABI allowing easy foreign function interfaces between other programming languages.

Today’s Problem Statement

Problem Statement Discussion

Potential Solutions

Time for Experiment

Solution Discussion

Caveats

Problem Statement Discussion

Background (completely fictional story)

MegaCorp operates a semi-automated internal "Update Distribution System" used for pushing utilities to employee systems.

On the morning of 15 October 2024, MegaCorp's internal monitoring systems detected unusual file distribution activity across five employee workstations.

Wave of EXEs

Two executables: update-tool.exe and system-monitor.exe

Unexpected installer droppers inside temp/ with sequence numbers

At first, this looked like a coordinated software update.

But the timestamps didn’t match any scheduled deployment window

First Signs of Trouble

SOC analysts noticed multiple employees receiving unusual emails

- notification-001.eml
- alert-system.eml
- update-###.eml

SoC in Action

All machines received identical files, often with identical mtimes,

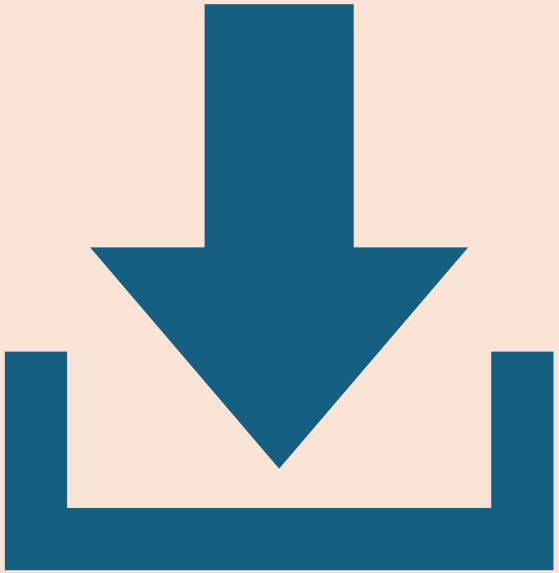
Files appeared in waves, a few seconds apart per machine, All machines referenced a network-share/deployment folder that no team publicly owned.

Was this a legitimate update, or a coordinated lateral spread?

Your job: **Reconstruct the true chain of events using timestamps alone.**

Dataset Download Link –
scenario.zip

https://tinyurl.com/v4ak7k6



Dataset Generated using **FSAGen**

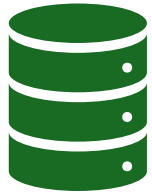
Find it here:

https://github.com/aoiflux/generator

Solution Discussion – High Level View

Loop through

Loop through all the artefacts (files)



Get

Get the data modified timestamps

Generate

Generate report

End Goal

Construction of a timeline of events that represents a chain of infection to show which computer was infected first and how malware moved through to the last one.

Time for Experiment

Solution Discussion

Caveats

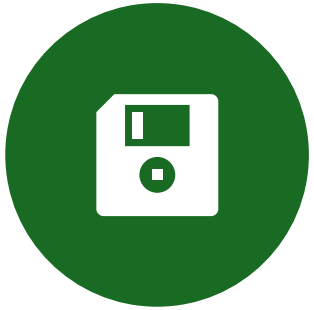
Different Timezone?

Clock Skew?

System Clock not in Sync with NTP Server?

Additional Content – Anti-Forensics

TIME-STOMPING ATTACKS



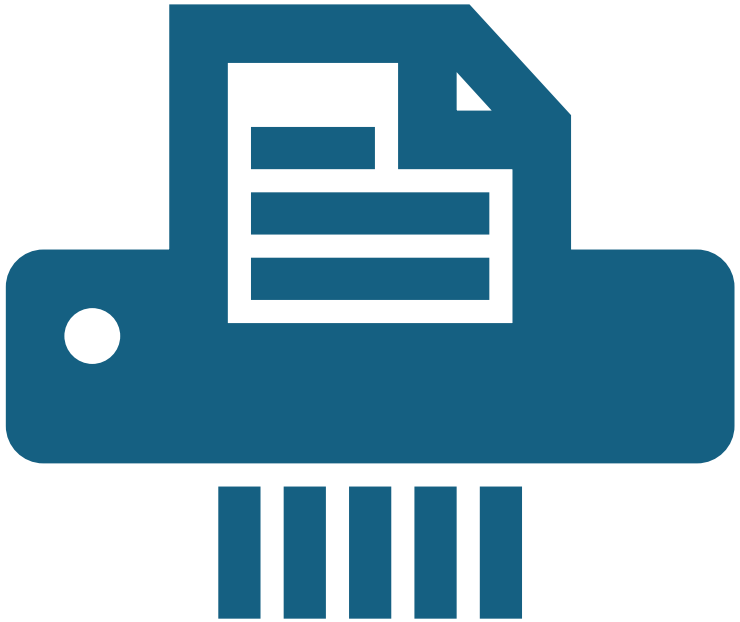
RAW DISK READS

SYSTEM CLOCK MANIPULATION

USN / $LOGFILE MANIPULATION

Additional Content – Anti-Forensics

- Forced SSD Trim
- Deleted Prefetch Files
- Fileless Malware
- Raw Disk Writes (Bypassing Filesystem)



Additional Content – More Investigation Scenarios!

FSAGen can be used to generate any number of scenarios

Provided OVA file contains FSAGen tool and playbooks

Please feel free to generate as many scenarios as you like

Q/A

Let’s Connect

Gaurav Gogia

Security R&D @Qualys | Purple Teaming