If You're Happy and You Know It
Inside the mind of a developer

Dom Davis
@idomdavis
THE WHEELS ON THE BUS
As I was going to St. Ives,
As I was going to St. Ives,
I met a man with seven wives,
As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
As I was going to St. Ives,
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
I met a man with seven wives,
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
Each wife had seven sacks,
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
1 + 7 = 8
7 x 7 = 49
Each sack had seven cats,
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
1 + 7 = 8
7 x 7 = 49
49 x 7 = 343
Each cat had seven kits:
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
1 + 7 = 8
7 x 7 = 49
49 x 7 = 343
343 x 7 = 2,401
Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
7
7 x 7 = 49
49 x 7 = 343
343 x 7 = 2,401

Kits, cats, sacks, and wives,
How many were there going to St. Ives?
$7^0 = 1$
$7^1 = 7$
$7^2 = 49$
$7^3 = 343$
$7^4 = 2,401$

Kits, cats, sacks, and wives,
How many were there going to St. Ives?
1
7
7 \times 7 = 49
49 \times 7 = 343
343 \times 7 = 2,401
7 + 49 + 353 + 2,401 = 2,800
How many were there going to St. Ives?
var howMany
undefined
>
As I was going to St. Ives, 
I met a man with seven wives, 
Each wife had seven sacks, 
Each sack had seven cats, 
Each cat had seven kits: 
Kits, cats, sacks, and wives, 
How many were there going to St. Ives?
$7^0 + 7^1 + 7^2 + 7^3 + 7^4 = 2,801$
Heads
Heads
Shoulders
Heads
Shoulders
Neezantos
Define: Web Services
Define: Neezanto
Oh, the cow in the meadow goes "moo!"
3/14 = \pi \text{ day}
3/14 = 3rd of when??
TGIF
TGIW
Mary had a little lamb,
its fleece was white as snow.
package main

type size string

type colour struct {
    r int
    g int
    b int
}

type lamb struct {
    size
    colour
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return lamb{size: s, colour: c}
}

func main() {
    marysLamb := New(little, snow)
}
Mary had a little lamb,
its fleece was white as snow.
And everywhere that Mary went,
The lamb was sure to go.
package main

type size string

type colour struct {
    r int
    g int
    b int
}

type location struct {
    x int
    y int
}

type lamb struct {
    size
    colour
    location
    mary location
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return return lamb{size: s, colour: c}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New(little, snow)
}
package main

type size string

type bags int

type colour struct {
    r int
    g int
    b int
}

type location struct {
    x int
    y int
}

type lamb struct {
    size
    colour
    location
    mary location
    wool bool
    yield bags
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return return lamb{size: s, colour: c}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New(little, snow)
}
package main

type size string

type bags int

type colour struct {
    r int
    g int
    b int
}

type location struct {
    x int
    y int
}

type lamb struct {
    size
    colour
    location
    mary location
    wool bool
    yield bags
    sound string
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return lamb{size: s, colour: c, sound: "Baa, baa!"}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New(little, snow)
}
package main

import (
    "fmt"
    "net/http"
    "github.com/gorilla/mux"
)

type size string

type bags int

type colour struct {
    r int
    g int
    b int
}

type location struct {
    x int
    y int
}

type lamb struct {
    size
    colour
    location
    mary location
    wool bool
    yield bags
    sound string
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return lamb{size: s, colour: c, sound: "Baa, baa!"}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New(little, snow)
    router := mux.NewRouter()
    
    router.HandleFunc("/size", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "%s", marysLamb.size)
    }).Methods("GET")
    
    router.HandleFunc("/colour", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "{r: %d, g: %d, b: %d}"
            + "\n",
            marysLamb.colour.r,
            marysLamb.colour.g,
            marysLamb.colour.b)
    }).Methods("GET")

    http.HandleFunc("/", router)
    fmt.Println("Listening on port 8001...")
    if err := http.ListenAndServe(":8001", nil); err != nil {
        panic(err)
    }
}
package main

import (
    "fmt"
    "net/http"
    "github.com/gorilla/mux"
)

type size string

type bags int

type colour struct {
    r int
    g int
    b int
}

type location struct {
    x int
    y int
}

type lamb struct {
    size
    colour
    location
    mary location
    wool bool
    yield bags
    sound string
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
    return lamb{size: s, colour: c, sound: "Baa, baa!", yield bags}
}

func marysLamb() {
    // route from l.location to l.mary
    // path() {
}

func main() {
    marysLamb = New(little, snow)
    router = mux.NewRouter()
        fmt.Printf("%s", marysLamb.size)
    }))).Methods("GET")
        fmt.Printf("{r: %d, g: %d, b: %d}", marysLamb.colour.r, marysLamb.colour.g, marysLamb.colour.b)
    }))).Methods("GET")
    http.Handle("/", router)
    fmt.Println("Listening on port 8001...")
    if err := http.ListenAndServe(":8001", nil); err != nil {
        panic(err)
    }
}
package main

type size string

type colour struct {
  r int
  g int
  b int
}

type location struct {
  x int
  y int
}

type lamb struct {
  size
  colour
  location
  mary location
}

var snow = colour{255, 255, 255}
const little = size("little")

func New(s size, c colour) lamb {
  return lamb{size: s, colour: c}
}

func (l lamb) path() {
  // route from l.location to l.mary
}

func main() {
  marysLamb := New(little, snow)
}
package main

type location struct {
    x int
    y int
}

type lamb struct {
    location
    mary location
}

func New() lamb {
    return return lamb{}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New()
}
package main

type location struct {
    x int
    y int
}

type lamb struct {
    location
    mary location
}

func New() lamb {
    return lamb{
}
}

func (l lamb) path() {
    // route from l.location to l.mary
}

func main() {
    marysLamb := New()
}
package main

type location struct {
    x int
    y int
}

func (l location) path(to location) {
    // route from current location to new location
}

func main() {
    lamb := location{0, 0}
    mary := location{1, 0}
    lamb.path(mary)
}
package main

type location struct {
    x int
    y int
}

type area struct {
    tl location
    tr location
    bl location
    br location
}

func (l location) path(to location, avoid []area) {
    // route from current location to new location
    // avoiding the given areas
}

func main() {
    lamb := location{0, 0}
    mary := location{1, 0}
    lamb.path(mary, []area{})
}
type name struct {
    title string
    givenName string
    middleNames []string
    surname string
    suffixes []string
}

type name struct {
    title string
    givenName string
    middleNames []string
    surname string
    suffixes []string
}

var re = regexp.MustCompile(`\s+`) 

func (n name) String() string {
    parts := []string{n.title, n.givenName}
    parts = append(parts, n.middleNames...)
    parts = append(parts, n.surname)
    parts = append(parts, n.suffixes...)

    fullName := strings.Join(parts, " ")
    fullName = strings.TrimSpace(fullName)
    fullName = re.ReplaceAllString(fullName, " ")

    return fullName
}
func (n name) String() string {
    parts := []string{n.title}
    if n.eastern {
        parts = append(parts, n.surname)
    } else {
        parts = append(parts, n.givenName)
    }
    parts = append(parts, n.middleNames...)
    if n.eastern {
        parts = append(parts, n.givenName)
    } else {
        parts = append(parts, n.surname)
    }
    parts = append(parts, n.suffixes...)
    fullName := strings.Join(parts, " ")
    fullName = strings.TrimSpace(fullName)
    fullName = re.ReplaceAllString(fullName, " ")
    return fullName
}
var name string
If you're happy and you know it
Clap your hands
package main

import "fmt"

type person struct {
    areHappy bool
    knowIt bool
    reallyWantToShowIt bool
}

func (p person) clapHands() {
    fmt.Println("Clap! Clap!")
}

func main() {
    you := person{true, true, true}

    if you.areHappy && you.knowIt {
        you.clapHands()
    }

    if you.areHappy && you.knowIt {
        you.clapHands()
    }

    if you.areHappy && you.knowIt && you.reallyWantToShowIt {
        if you.areHappy && you.knowIt {
            you.clapHands()
        }
    }
}
Exploit found: "If you're happy and you know it" allows for execution of unsigned, arbitrary instructions on toddler.
package main

import "fmt"

type person struct {
    areHappy     bool
    knowIt       bool
    reallyWantToShowIt bool
}

func (p person) clapHands() {
    fmt.Println("Clap! Clap!")
}

func main() {
    you := person{true, true, true}

    if you.areHappy && you.knowIt {
        you.clapHands()
    }

    if you.areHappy && you.knowIt {
        you.clapHands()
    }

    if you.areHappy && you.knowIt && you.reallyWantToShowIt {
        if you.areHappy && you.knowIt {
            you.clapHands()
        }
    }
}
package main

import "fmt"

type person struct {
    areHappy    bool
    knowIt      bool
    reallyWantToShowIt bool
}

func (p person) clapHands() {
    fmt.Println("Clap! Clap!")
}

func main() {
    you := person{true, true, true}

    if you.areHappy && you.knowIt {
        you.clapHands()
        you.clapHands()
        if you.reallyWantToShowIt {
            you.clapHands()
        }
    }
}
package main

import "fmt"

type Person struct {
    AreHappy bool
    KnowIt bool
    ReallyWantToShowIt bool
}

func (p Person) clapHands() {
    fmt.Println("Clap! Clap!")
}

func Clapper(you Person) {
    if you.AreHappy && you.KnowIt {
        you.clapHands()
    }

    if you.AreHappy && you.KnowIt {
        you.clapHands()
    }

    if you.AreHappy && you.KnowIt && you.ReallyWantToShowIt {
        if you.AreHappy && you.KnowIt {
            you.clapHands()
        }
    }
}