

# Desenvolvimento Seguro em Go

O que é? É importante? Como fazer?

# Quem sou eu

- Me chamo Anderson
- Meu primeiro programa foi para o Tibia 7.3
- Desde 2019 oficialmente na na área de TI.
- Ex-Consultor de Segurança
- Hoje estou no MELI (Mercado Livre) ❤️
- Também: Roteirista, Produtor & Editor de vídeo,  
Vendedor, Artista e Dançarino

Por que um dançarino está aqui?

# Agenda

1. O que é segurança no contexto do desenvolvimento
2. Por que isso é importante
3. Como isso aparece no código de verdade

ATO 1.0 que é segurança no  
contexto do desenvolvimento

# O que significa “desenvolver c/ segurança”

Desenvolver com segurança é...

- Previne abusos, ataques e usos indevidos
- Escrever código que resiste a falhas, não só que "funciona"
- Escrever defensivamente

# Infraestrutura vs Código

## Infraestrutura

Firewalls, autenticação externa

Configuração da cloud

Monitoramento e alertas

## Código

Validação de input, controle de acesso

Sanitização de dados, tratamento de erros

Logging seguro, autenticação por sessão

# Desenvolvimento Seguro $\neq$ AppSec

- AppSec é a estratégia: políticas, processos, ferramentas
- Desenv. Seguro é a prática: código que resiste a ataques
- Um orienta, o outro executa

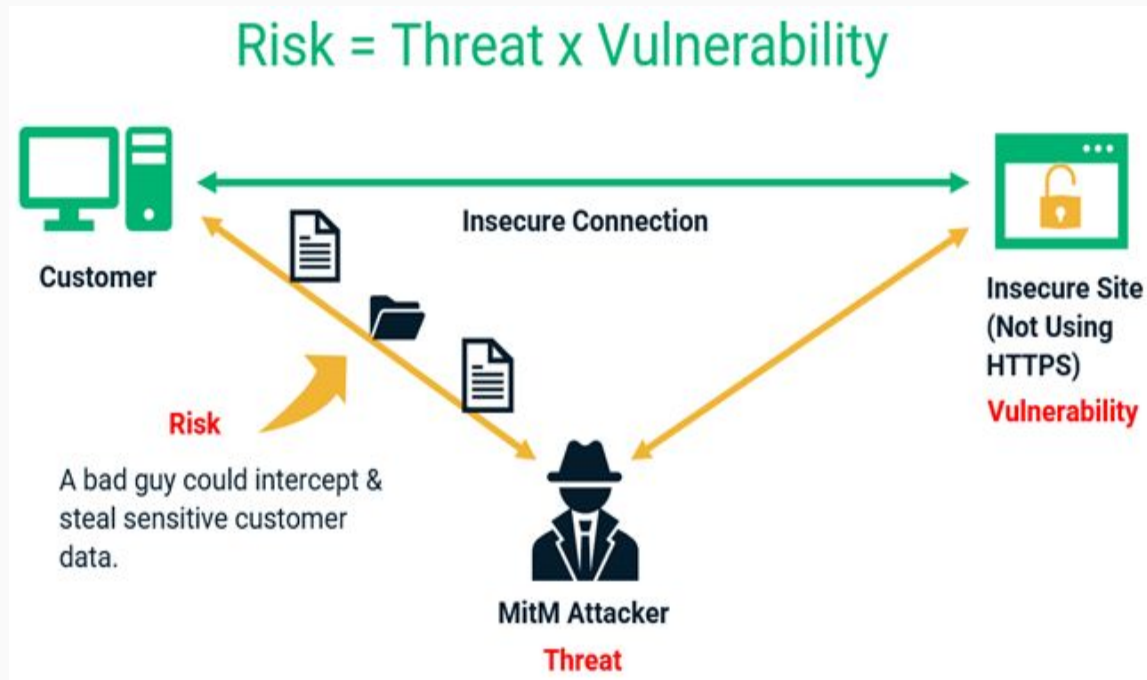


# Princípios: Vulnerabilidade, Ameaça e Risco

**Vulnerabilidade** → é o ponto fraco

**Ameaça** → agente ou evento que explora o ponto fraco

**Risco** → impacto se essa exploração acontecer



# Dev's são a primeira linha de defesa

- AppSec é a estratégia: políticas, processos, ferramentas
- Desenv. Seguro é a prática: código que resiste a ataques
- Um orienta, o outro executa

ATO 2. Por que isso é  
importante

## Hackers causaram prejuízos a cerca de 25% das empresas brasileiras em 2022, diz pesquisa

A varejista Americanas perdeu R\$ 1 bilhão em vendas após sofrer um ataque hacker em 2022; estudo foi divulgado pela empresa de segurança Proofpoint.



Por Reuters

08/03/2023 14h00 · Atualizado há 2 anos

Hacke  
das en  
pesqu

A varejista A  
2022; estud



Por  
08/

**exame.****Entrar**

# Americanas (AME3) sofre prejuízo de quase R\$ 1 bilhão com ataque hacker

No dia 19 de fevereiro, a Americanas sofreu um ataque hacker que impediu o funcionamento do site por cinco dias.

● Ao vivo Política WW Economia Esportes Pop Viagem

## Hackers: empresas do Brasil perderão R\$ 2,2 trilhões em 3 anos, diz estudo

Análise da VULTUS Cybersecurity Ecosystem observou 117 médias e grandes empresas brasileiras de 10 setores diferentes

Flávio Ismerim, da CNN

22/03/25 às 09:23:16 | Atualizado 24/03/25 às 12:15:32

Directed by  
ROBERT B. WEIDE

# Segurança não é um problema hipotético — já é realidade

- Americanas: **R\$ 923 milhões** em perdas por ataque em 2023
- **25%** das empresas brasileiras sofreram ataques em 2022
- Custo médio global de violação em 2024: **US\$ 4,88 milhões**
- Clientes perdem confiança → **churn**
- Marca abalada, multas, perda de **valor de mercado**



# Go tem menos pegadinhas?

## Sim. Mas ainda é software


- Menos mágica: mais previsível
- Mas ainda depende de validações, verificações e padrões
- Segurança não é só evitar bugs – é evitar comportamento inesperado

ATO 3. Como isso aparece no código?

# Como isso aparece no código?

1. Vamos ver códigos reais com falhas comuns
  - a. SQL injection
  - b. Server Side Request Forgery
  - c. Path Traversal
  - d. Information Exposure
2. E suas versões corrigidas e explicações práticas

# SQL Injection



```
1 func getUserByEmail(db *sql.DB, email string) (*User, error) {
2     query := "SELECT id, name FROM users WHERE email = '" + email + "'"
3     row := db.QueryRow(query)
4
5     var user User
6     row.Scan(&user.ID, &user.Name)
7     return &user, nil
8 }
```

- ✗ Nenhuma sanitização.
- ✗ Query construída via concatenação de string.

## SQL INJECTION



[Links & Slides](#)

# SQL Injection



```
1 func getUserByEmail(db *sql.DB, email string) (*User, error) {
2     if email == "" || !isValidEmail(email) {
3         return nil, errors.New("email inválido")
4     }
5     row := db.QueryRow("SELECT id, name FROM users WHERE email = ?", email)
6     var user User
7     if err := row.Scan(&user.ID, &user.Name); err != nil {
8         return nil, err
9     }
10    return &user, nil
11 }
```

- ✓ Sempre use **PreparedStatements** ao montar queries com dados e deixa o Driver cuidar do resto.
- ✓ Receba os valores separadamente e aplique validações de negócio.

# Server Side Request Forgery (SSRF)

```
1 func fetchURL(w http.ResponseWriter, r *http.Request) {  
2     target := r.URL.Query().Get("url")  
3  
4     resp, err := http.Get(target)  
5     if err != nil {  
6         http.Error(w, "Failed to fetch", http.StatusInternalServerError)  
7         return  
8     }  
9     defer resp.Body.Close()  
10    io.Copy(w, resp.Body)  
11 }
```

✗ Permite que o usuário defina qualquer URL (externa ou interna).

✗ Sem validação de domínio ou IP.

# Server Side Request Forgery (SSRF) DEMO

SSRF



[Links & Slides](#)



# Server Side Request Forgery (SSRF)



```
12     ips, err := net.LookupIP(u.Hostname())
1 func func 13     if
14         15
16     } 1 func
17     r 2     1 func areIPsSafe(ips []net.IP) bool {
18     i 3     2     for _, ip := range ips {
19         4     3         if ip.IsLoopback() || ip.IsPrivate()
20         5     4         || ip.IsLinkLocalUnicast() || ip.IsLinkLocalMulticast() {
21     } 6     5         return false
22     d 7     6     }
23     i 8     7     }
24 } 9     8     return true
10     9 }
```

✓ Valide domínios permitidos (whitelist) e evite acessar IPs internos.

✓ Faça resolução DNS antes da requisição.

# Path Traversal



```
1 func fileHandler(w http.ResponseWriter, r *http.Request) {  
2     file := r.URL.Query().Get("file")  
3     http.ServeFile(w, r, "./uploads/" + file)  
4 }
```

- ❌ Concatenação direta do caminho sem validação
- ❌ Nenhuma verificação se o caminho final está dentro da pasta permitida

# Path Traversal DEMO

PATH  
TRAVERSAL



[Links & Slides](#)

# Path Traversal



```
1 func fileHandler(w http.ResponseWriter, r *http.Request) {
2     file := filepath.Clean(r.URL.Query().Get("file"))
3     path := filepath.Join("./uploads", file)
4
5     if !strings.HasPrefix(path, filepath.Clean("./uploads")) {
6         http.Error(w, "Invalid path", http.StatusBadRequest)
7         return
8     }
9     http.ServeFile(w, r, path)
10 }
```

- ✓ Faça verificação do path e use caminhos absolutos
- ✓ Faça normalização dos paths (filepath.Clean)

# Exposição de Informações



```
1 log.Printf("Login attempt: email=%s, password=%s", email, password)
2 log.Printf("Auth header: %s", r.Header.Get("Authorization"))
3 log.Printf("Token: %s", token)
4 log.Printf("User data: %+v", user) // cuidado com structs que têm senha!
```

- ❌ Não logar senhas, tokens ou headers de autenticação
- ❌ Não expor dados PII (email, CPF, telefone, etc)
- ❌ Não enviar log para sistemas de observabilidade

# INFORMATION DISCLOSURE



[Links & Slides](#)

# Exposição de Informações



```
1 logger = zap.NewProduction()
2 logger.debug("Login attempt: user_id=%d", userId)
3 logger.debug("Has token: %v", token != "")
4 logger.debug("Auth header: %s",
5     utils.ObfuscateString(r.Header.Get("Authorization"), 5)
6 ) // OUTPUT: *****abcde
```

- ✓ Log apenas dados não sensíveis.
- ✓ Usar logger estruturado com campos explícitos
- ✓ Use obfuscadores automáticos de log para esconder ou omitir campos sensíveis.

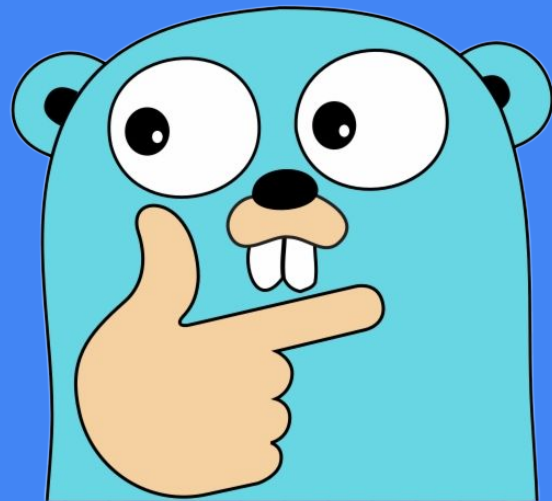
# Lições aprendidas

- Nunca confie em dados externos
- Valide input sempre
- Valide autenticação e autorização
- Logue com cuidado
- Segurança começa no dev, não no deploy



# Você é a primeira linha de defesa

Comece pequeno, mas comece



# Obrigado!



[Links & Slides](#)