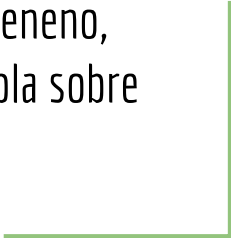


# Análisis de código 100% in house

Una historia de árboles, veneno,  
impuestos y código que habla sobre  
código.



¿Qué es análisis estático?

¿Por qué análisis estático?

¿Qué haría un equipo de  
seguridad sano y normal?

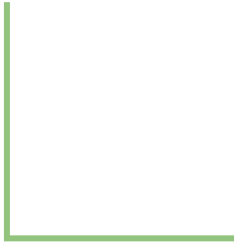


Contratar una solución





Contract solutions



¿Se puede hacer algo mejor?

# ASTs Fantastic Infosec Pilgrim

AFIP, para los amigos: persigue  
arbolitos de código como la AFIP  
persigue arbolitos de dólares





# Regexes



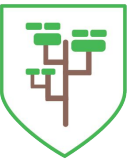
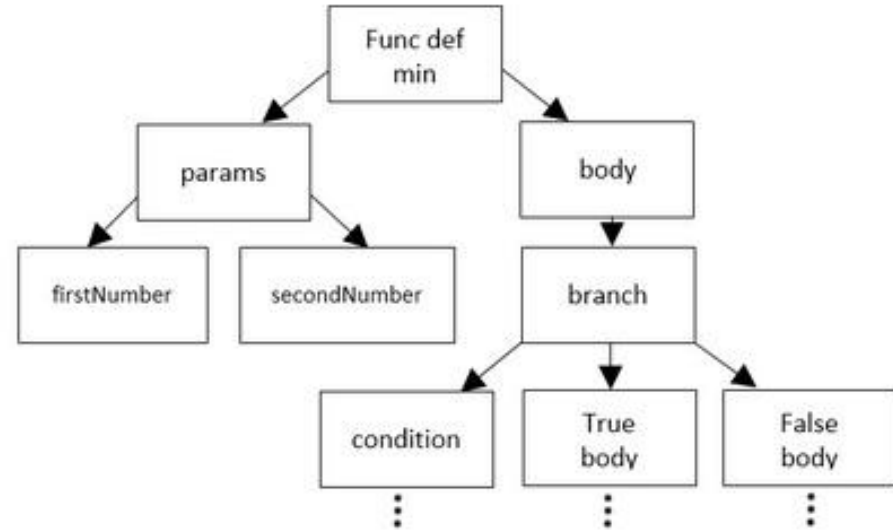
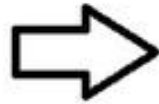
# Regexes

```
if\s*\(((?: (?: (?: " (?: (?: \\ " ) | [^"] ) * " ) | (?: ' (?: (?: \\ ' ) | [^'] ) * ' ) ) | [^\(\)] | \ ( ( ? 1 ) \ ) * + ) \) \s* { ( (?: (?: (?: " (?: (?: \\ " ) | [^"] ) * " ) | (?: ' (?: (?: \\ ' ) | [^'] ) * ' ) ) | [^{} ] | { ( ? 2 ) } } * + ) } \s* (?: (?: else\s* { ( (?: (?: (?: " (?: (?: \\ " ) | [^"] ) * " ) | (?: ' (?: (?: \\ ' ) | [^'] ) * ' ) ) | [^{} ] | { ( ? 3 ) } } * + ) } \s* ) | (?: else\s* if\s* \ ( ( (?: (?: (?: " (?: (?: \\ " ) | [^"] ) * " ) | (?: ' (?: (?: \\ ' ) | [^'] ) * ' ) ) | [^\(\)] | \ ( ( ? 4 ) \ ) * + ) \) \s* { ( (?: (?: (?: " (?: (?: \\ " ) | [^"] ) * " ) | (?: ' (?: (?: \\ ' ) | [^'] ) * ' ) ) | [^{} ] | { ( ? 5 ) } } * + ) } \s* ) ) * ;
```



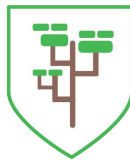
# AST: Abstract Syntax Trees

```
int min(int firstNumber, int secondNumber)
{
    if (firstNumber > secondNumber) {
        return secondNumber;
    }
    else {
        return firstNumber;
    }
}
```



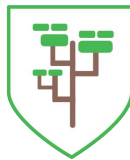
# Taint analysis

```
def controller() {  
  def parser = new JsonSlurper()  
  def body = request.JSON.toString()  
  def json = parser.parseText(body)  
  def echo = json.get("echo")  
  render(status: 200, text: echo)  
}
```



# Taint analysis

```
def controller() {  
  def parser = new JsonSlurper()  
  def body = request.JSON.toString()  
  def json = parser.parseText(body)  
  def echo = json.get("echo")  
  echo = onlyAlphanumeric(echo)  
  render(status: 200, text: echo)  
}
```



Linda teoría

¿Y la práctica?

# Podemos encontrar taint.

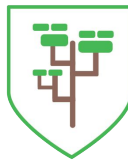
```
1. // http://foo.com/bar?example=HelloWorld
2. // params.example == HelloWorld
3. def OpenRedirect() {
4.     def url = params.redirect
5.     redirect(url: url)
6. }
```





# También cuando algo se limpió

```
1. def isValidURL(url) {  
2.     return url.startswith("https://mercadolibre.com")  
3. }  
4.  
5. def makeValidURL(url) {  
6.     if (isValidURL(url)) {  
7.         return url  
8.     }  
9.     return "https://mercadolibre.com"  
10. }  
11.  
12. def OpenRedirect() {  
13.     def url = makeValidURL(params.redirect)  
14.     redirect(url: url)  
15. }
```



# Hacer análisis inter método

```
1. def capitalize(url) {  
2.     return url.capitalize()  
3. }  
4.  
5. def OpenRedirect() {  
6.     def url = capitalize(params.redirect)  
7.     redirect(url: url)  
8. }
```



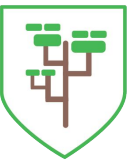
# ¡Y funciona!

```
1. def MPOrML(site) {
2.     if (site == "ML") {
3.         return "https://mercadolibre.com"
4.     }
5.     return "https://mercadopago.com"
6. }
7.
8. def OpenRedirect() {
9.     def url = MPOrML(params.site)
10.    redirect(url: url)
11. }
```



# ¡Funciona-ish!

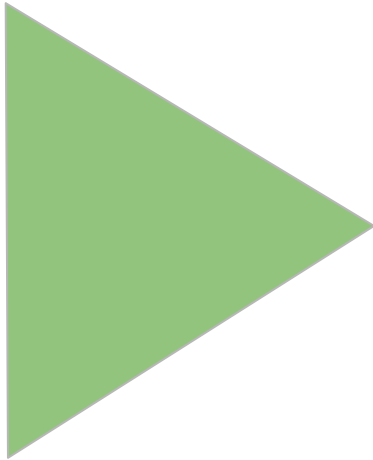
```
1. def OpenRedirect() {  
2.     def url = params.redirect  
3.     if (url) {  
4.         url = "https://mercadolibre.com"  
5.     } else {  
6.         url = url  
7.     }  
8.     redirect(url: url)  
9. }
```



Ok, ¿cómo funciona?

# 1. Exploración del proyecto

```
grails-app
├── conf
│   ├── BootStrap.groovy
│   ├── BuildConfig.groovy
│   ├── Config.groovy
│   ├── spring
│   │   └── resources.groovy
│   └── UrlMappings.groovy
├── controllers
│   ├── env
│   │   └── app
│   │       └── EnvController.groovy
│   ├── mlsec
│   │   └── app
│   │       └── MlsecController.groovy
│   ├── ping
│   │   └── app
│   │       └── PingController.groovy
│   ├── stress
│   │   └── app
│   │       └── StressController.groovy
│   └── worst
│       └── app
│           └── WorstController.groovy
└── views
```



```
project
.getService("aService")
.getScope("myMethod")
```

## 2. Canonización

```
def getFoo() {  
    return ["foo": "bar"]  
}  
  
def controller() {  
    def f = foo  
    return f  
}
```



```
def getFoo() {  
    return ["foo": "bar"]  
}  
  
def controller() {  
    def f = getFoo()  
    return render(f)  
}
```

# 3. AST -> Variables

```
[BinaryExpression]
→ LeftHand
  → VariableExpression
    → Text: "foo"
→ RightHand
  → LiteralString
    → "bar"
```



```
def v= getVar("foo")
v.amountOfDefinitions()
v.getValue()
v.isTainted()
```



## 4. Clasificación de métodos

$$\lambda(p_1, p_2, p_n) \rightarrow R$$

Source

## 4. Clasificación de métodos

$$\lambda(p_1, p_2, p_n) \rightarrow R$$

Cleaner

## 4. Clasificación de métodos

$$\lambda(p_1, p_2, p_n) \rightarrow R$$

$$\lambda(p_1, p_2, p_n) \rightarrow R$$

Menace

## 4. Clasificación de métodos

$$\lambda(p_1, p_2, p_n) \left\{ \lambda_{\text{SINK}}(P) \right\}$$

## 5. Finding de taint

```
def x = λ(p)
def y = x + γ(g)
return y
```

## 6. Finding de vulns

$\lambda_{\text{SINK}}(P)$





¿Qué hicimos bien?





Tenemos más del 50% del  
código cubierto

Porcentaje relativamente bajo  
de falsos positivos

Customizable y sin riesgo de  
vendor locking

¿Qué salió mal?

No hicimos prioritaria la  
interfaz de usuario

Too deep, little breadth

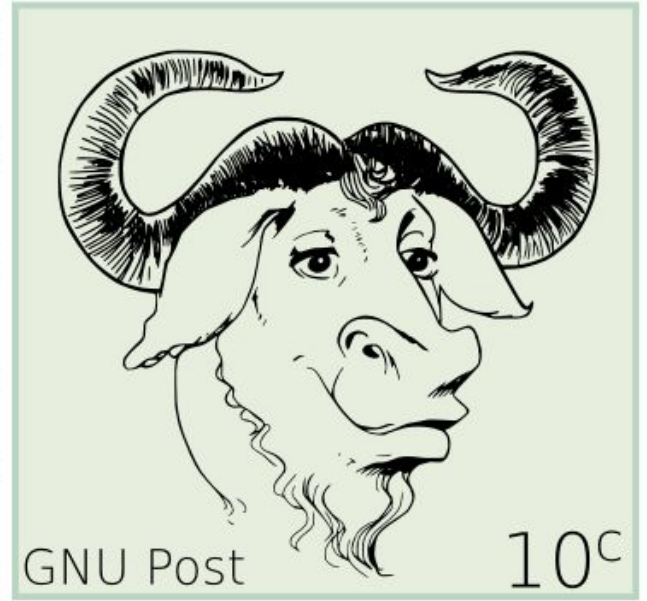
Subestimar métricas y  
seguimiento necesario

# El ecosistema AFIP

- **Scanner** (Groovy)
- **API** (Golang)
- **Generación de código** (Python)
- **Playground** (Node + React)
- **Tests integrales** (Node)
- **Tools de CLI** (Python)
- **Backoffice** (Node + React)



Una cosa más



# github.com/mercadolibre/afip-grails

- **Scanner** (Groovy)
- **Generación de código** (Python)
- **Playground** (Node + React)
- **API** (Golang)
- **Tests integrales** (Node)
- **Tools de CLI** (Python)
- **Backoffice** (Node + React)



¡Gracias! <3

