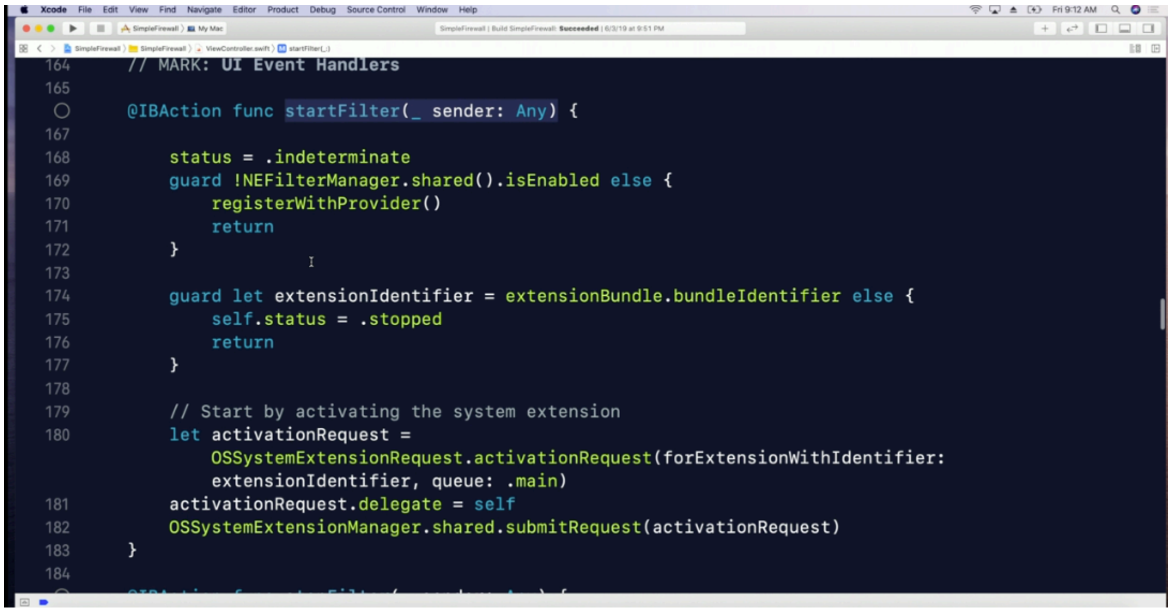
Simple Firewall Tutorial

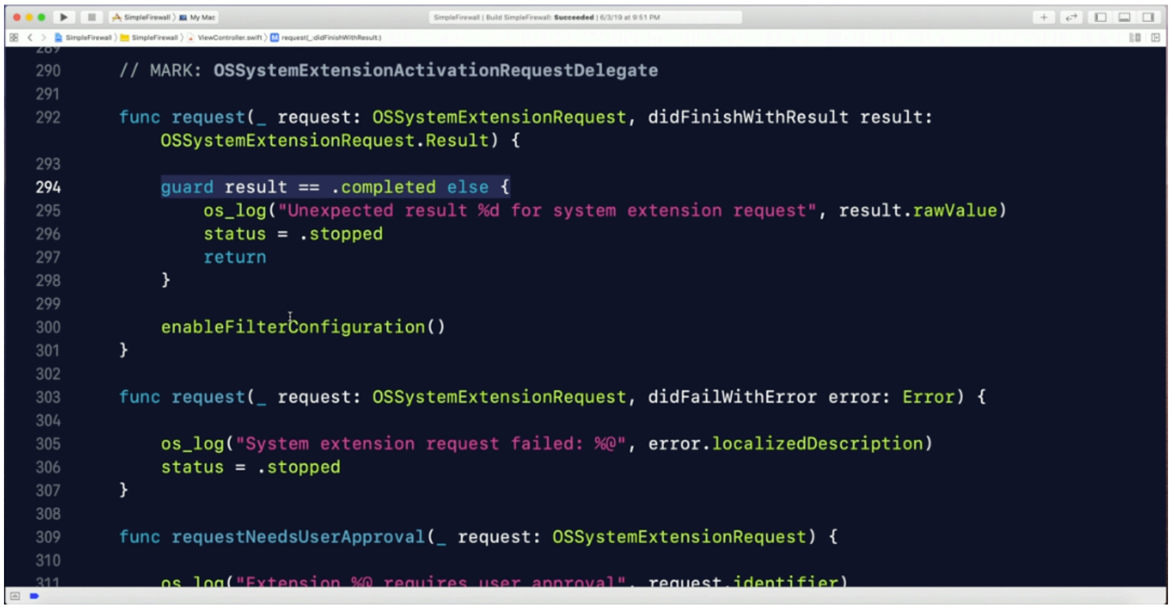
Start filter function;



```
    // MARK: UI Event Handlers

    @IBAction func startFilter(_ sender: Any) {

        status = .indeterminate
        guard !NEFilterManager.shared().isEnabled else {
            registerWithProvider()
            return
        }

        guard let extensionIdentifier = extensionBundle.bundleIdentifier else {
            self.status = .stopped
            return
        }

        // Start by activating the system extension
        let activationRequest =
            OSSystemExtensionRequest.activationRequest(forExtensionWithIdentifier:
            extensionIdentifier, queue: .main)
        activationRequest.delegate = self
        OSSystemExtensionManager.shared.submitRequest(activationRequest)
    }
```

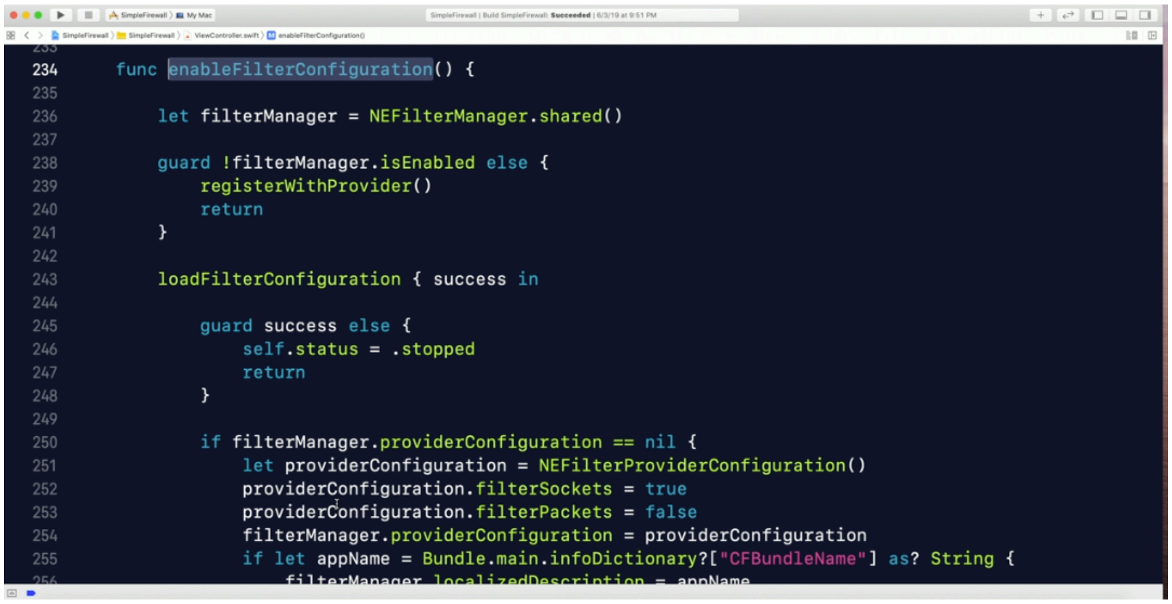
- requests System Extension

On completion of system extension request;



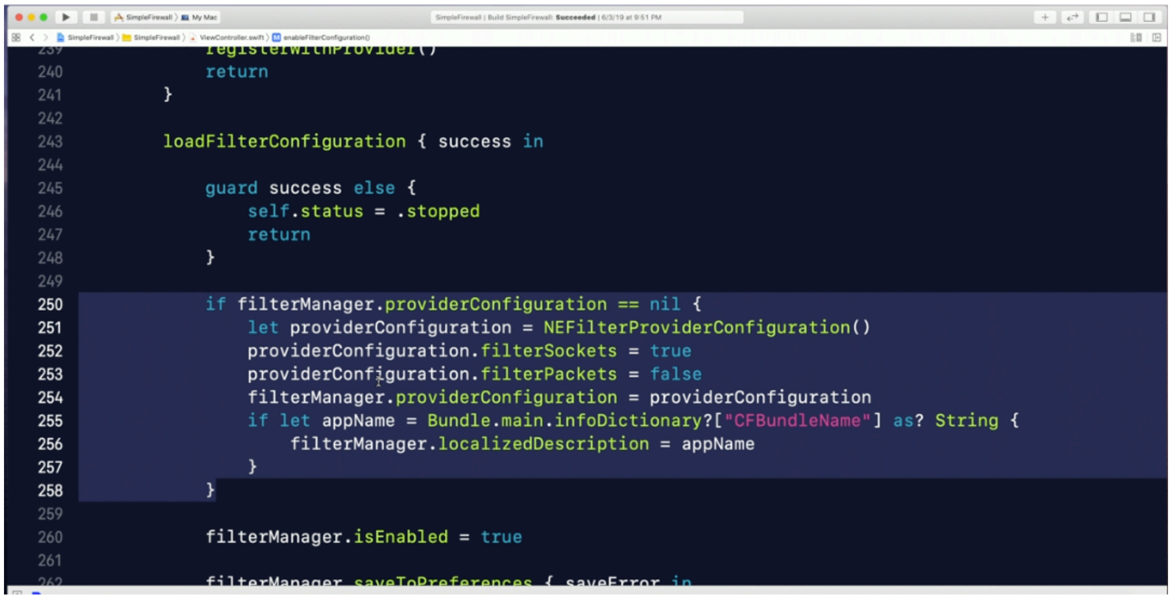
```
    // MARK: OSSystemExtensionActivationRequestDelegate

    func request(_ request: OSSystemExtensionRequest, didFinishWithResult result:
        OSSystemExtensionRequest.Result) {

        guard result == .completed else {
            os_log("Unexpected result %d for system extension request", result.rawValue)
            status = .stopped
            return
        }

        enableFilterConfiguration()
    }

    func request(_ request: OSSystemExtensionRequest, didFailWithError error: Error) {

        os_log("System extension request failed: %@", error.localizedDescription)
        status = .stopped
    }

    func requestNeedsUserApproval(_ request: OSSystemExtensionRequest) {

        os_log("Extension %@ requires user approval", request.identifier)
```

Create content filter config w/NEFilterManager after req succ;



```
    func enableFilterConfiguration() {

        let filterManager = NEFilterManager.shared()

        guard !filterManager.isEnabled else {
            registerWithProvider()
            return
        }

        loadFilterConfiguration { success in

            guard success else {
                self.status = .stopped
                return
            }

            if filterManager.providerConfiguration == nil {
                let providerConfiguration = NEFilterProviderConfiguration()
                providerConfiguration.filterSockets = true
                providerConfiguration.filterPackets = false
                filterManager.providerConfiguration = providerConfiguration
                if let appName = Bundle.main.infoDictionary?["CFBundleName"] as? String {
                    filterManager.localizedDescription = appName
```

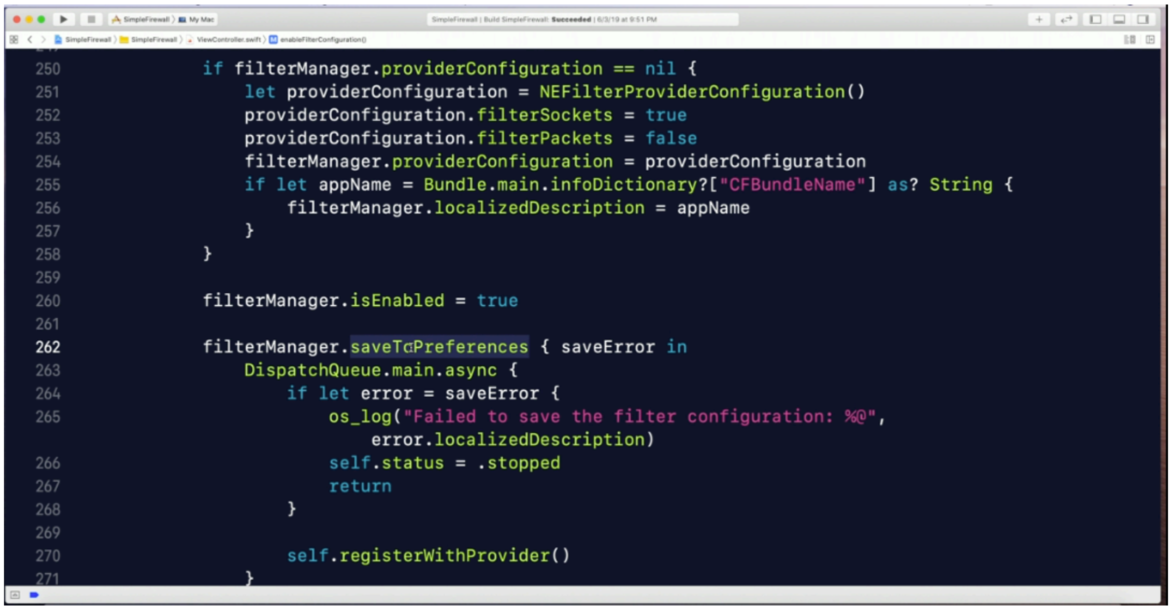
Set up details on config;



```
                registerWithProvider()
            return
        }

        loadFilterConfiguration { success in

            guard success else {
                self.status = .stopped
                return
            }

            if filterManager.providerConfiguration == nil {
                let providerConfiguration = NEFilterProviderConfiguration()
                providerConfiguration.filterSockets = true
                providerConfiguration.filterPackets = false
                filterManager.providerConfiguration = providerConfiguration
                if let appName = Bundle.main.infoDictionary?["CFBundleName"] as? String {
                    filterManager.localizedDescription = appName
                }
            }

            filterManager.isEnabled = true

            filterManager.saveToPreferences { saveError in
```

- filterSockets true; specifies to filter network traffic at the flow layer
- filterPackets false; not filtering network traffic at packet layer.
- enable the config

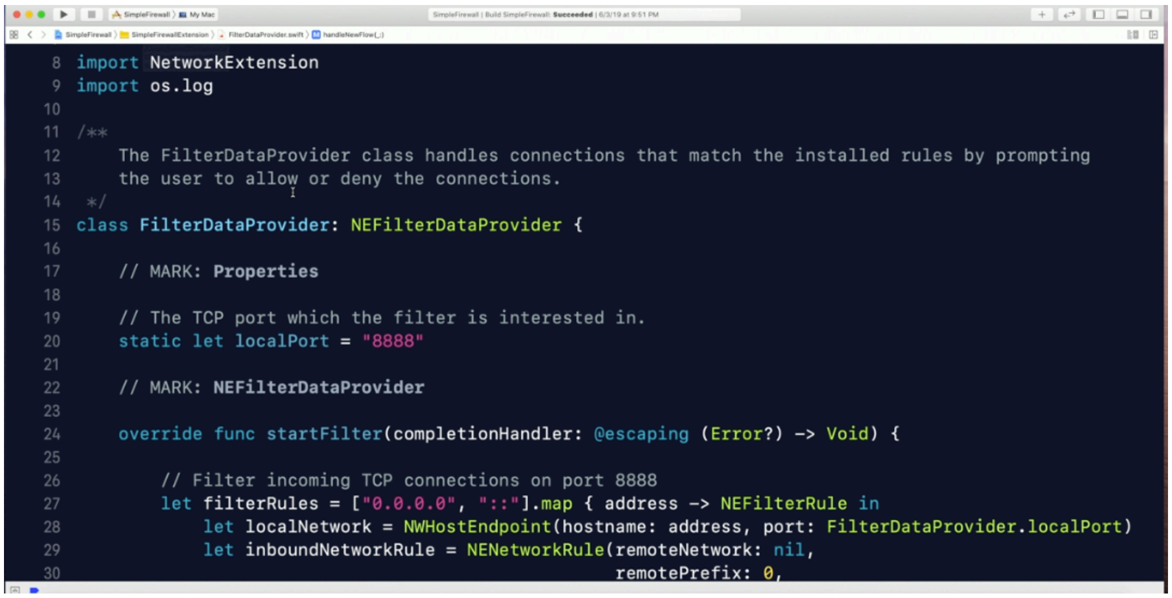
Register config w/system by calling saveToPreferences;



```
        if filterManager.providerConfiguration == nil {
            let providerConfiguration = NEFilterProviderConfiguration()
            providerConfiguration.filterSockets = true
            providerConfiguration.filterPackets = false
            filterManager.providerConfiguration = providerConfiguration
            if let appName = Bundle.main.infoDictionary?["CFBundleName"] as? String {
                filterManager.localizedDescription = appName
            }
        }

        filterManager.isEnabled = true

        filterManager.saveToPreferences { saveError in
            DispatchQueue.main.async {
                if let error = saveError {
                    os_log("Failed to save the filter configuration: %@",
                        error.localizedDescription)
                    self.status = .stopped
                    return
                }

                self.registerWithProvider()
            }
```

- Because it's enabled
- causes system to start the extension and filtering.

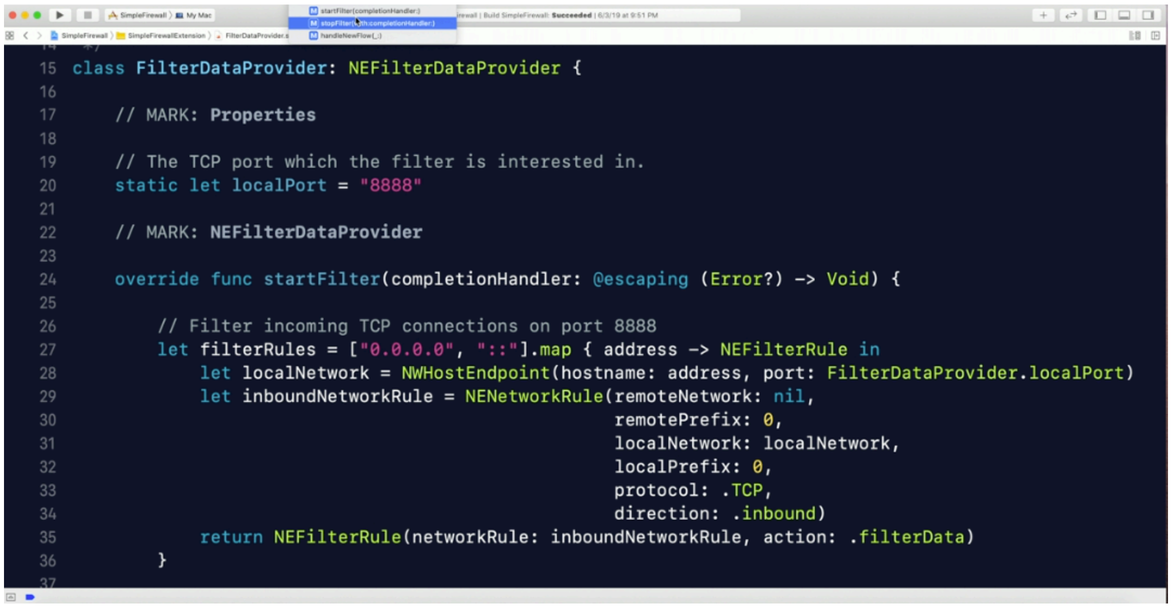
NEFilterDataProviderSubclass



```
import NetworkExtension
import os.log

/**
    The FilterDataProvider class handles connections that match the installed rules by prompting
    the user to allow or deny the connections.
 */
class FilterDataProvider: NEFilterDataProvider {

    // MARK: Properties

    // The TCP port which the filter is interested in.
    static let localPort = "8888"

    // MARK: NEFilterDataProvider

    override func startFilter(completionHandler: @escaping (Error?) -> Void) {

        // Filter incoming TCP connections on port 8888
        let filterRules = ["0.0.0.0", "::"].map { address -> NEFilterRule in
            let localNetwork = NWHostEndpoint(hostname: address, port: FilterDataProvider.localPort)
            let inboundNetworkRule = NENetworkRule(remoteNetwork: nil,
                                                   remotePrefix: 0,
```

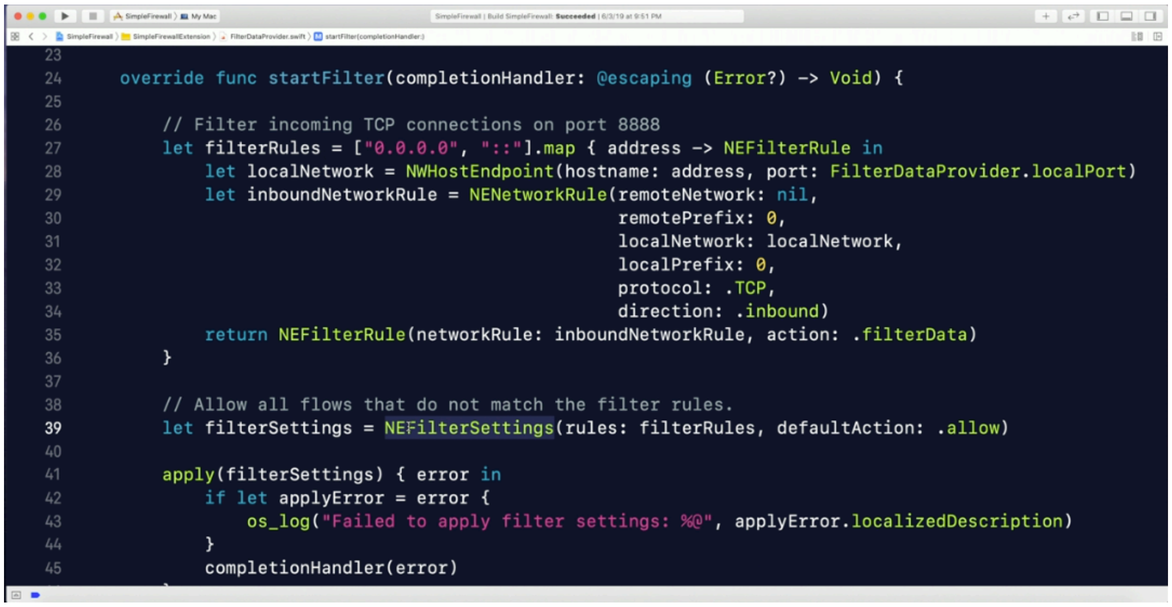
- Runs inside the system extension

Class overrides three different methods (1. startFilter, 2. stopFilter 3. handleNewFlow)


SimpleFirewall > My Mac
startFilter(completionHandler:)
stopFilter(with:completionHandler:)
handleNewFlow(_:)
SimpleFirewall > SimpleFirewallExtension > FilterDataProvider.swift
Build SimpleFirewall: Succeeded | 6/3/19 at 9:51 PM

```swift
class FilterDataProvider: NEFilterDataProvider {

    // MARK: Properties

    // The TCP port which the filter is interested in.
    static let localPort = "8888"

    // MARK: NEFilterDataProvider

    override func startFilter(completionHandler: @escaping (Error?) -> Void) {

        // Filter incoming TCP connections on port 8888
        let filterRules = ["0.0.0.0", "::"].map { address -> NEFilterRule in
            let localNetwork = NWHostEndpoint(hostname: address, port: FilterDataProvider.localPort)
            let inboundNetworkRule = NENetworkRule(remoteNetwork: nil,
                                                   remotePrefix: 0,
                                                   localNetwork: localNetwork,
                                                   localPrefix: 0,
                                                   protocol: .TCP,
                                                   direction: .inbound)
            return NEFilterRule(networkRule: inboundNetworkRule, action: .filterData)
        }
```

startFilter (by default filters all tcp udp traffic)

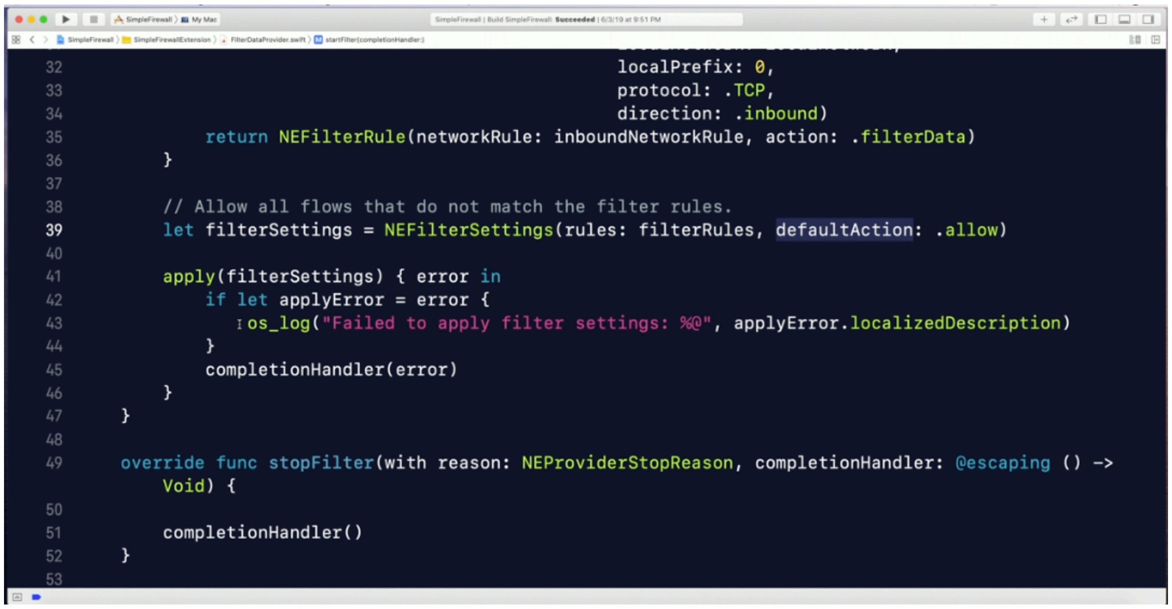

SimpleFirewall > My Mac
SimpleFirewall | Build SimpleFirewall: Succeeded | 6/3/19 at 9:51 PM
SimpleFirewall > SimpleFirewallExtension > FilterDataProvider.swift > startFilter(completionHandler:)

```swift
    override func startFilter(completionHandler: @escaping (Error?) -> Void) {

        // Filter incoming TCP connections on port 8888
        let filterRules = ["0.0.0.0", "::"].map { address -> NEFilterRule in
            let localNetwork = NWHostEndpoint(hostname: address, port: FilterDataProvider.localPort)
            let inboundNetworkRule = NENetworkRule(remoteNetwork: nil,
                                                   remotePrefix: 0,
                                                   localNetwork: localNetwork,
                                                   localPrefix: 0,
                                                   protocol: .TCP,
                                                   direction: .inbound)
            return NEFilterRule(networkRule: inboundNetworkRule, action: .filterData)
        }

        // Allow all flows that do not match the filter rules.
        let filterSettings = NEFilterSettings(rules: filterRules, defaultAction: .allow)

        apply(filterSettings) { error in
            if let applyError = error {
                os_log("Failed to apply filter settings: %@", applyError.localizedDescription)
            }
            completionHandler(error)
```

- called when system starts the filter
- **filterSettings:** Creates the NEFilterSettings object passing in the filter rules.
- creates NEFiltersSetting object to inform the system what it wants to see and thus filter.
- filterRules; creates wildcard ipv4 & ipv6 addresses.
- NENetworkRule;
    - **remoteNetwork: nil;** (*filter rule will match traffic coming from anywhere*)
    - **remotePrefix: 0;** (*filter rule will match traffic coming from anywhere*)
    - **localNetwork: localNetwork;** *(Uses NWHostEntpoint to accept from local port 8.8.8.8)*
    - **localPrefix: 0;**
    - **protocol: .TCP;**

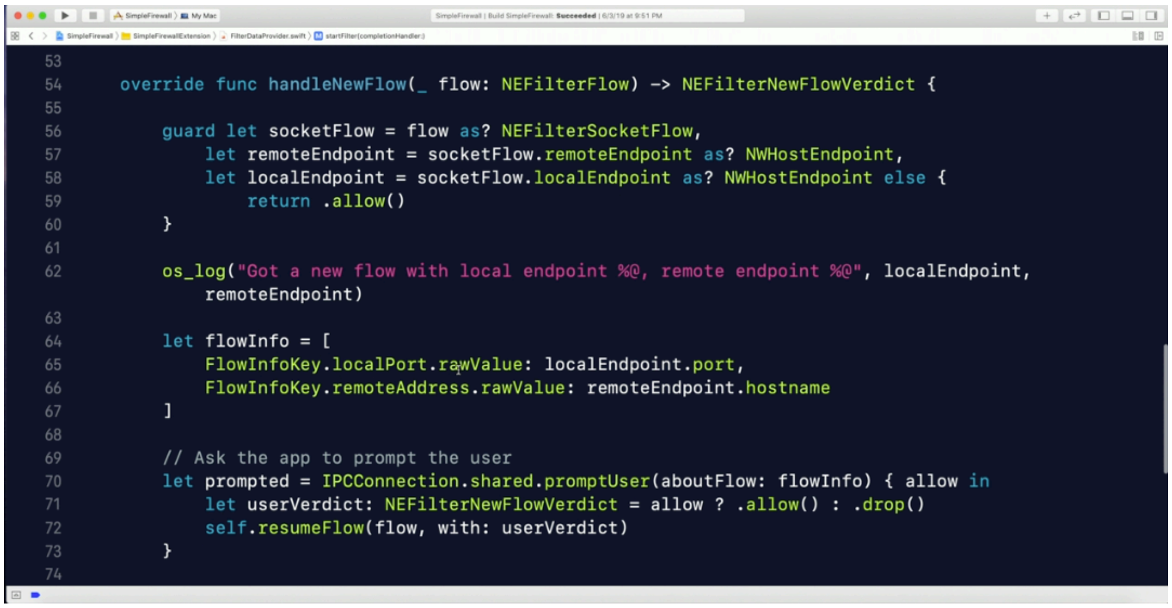
• - **direction: .inbound;**

Call apply to apply filter settings to the system;



```
                                                  localPrefix: 0,
                                                  protocol: .TCP,
                                                  direction: .inbound)
            return NEFilterRule(networkRule: inboundNetworkRule, action: .filterData)
        }

        // Allow all flows that do not match the filter rules.
        let filterSettings = NEFilterSettings(rules: filterRules, defaultAction: .allow)

        apply(filterSettings) { error in
            if let applyError = error {
                os_log("Failed to apply filter settings: %@", applyError.localizedDescription)
            }
            completionHandler(error)
        }
    }

    override func stopFilter(with reason: NEProviderStopReason, completionHandler: @escaping () ->
        Void) {

        completionHandler()
    }

```

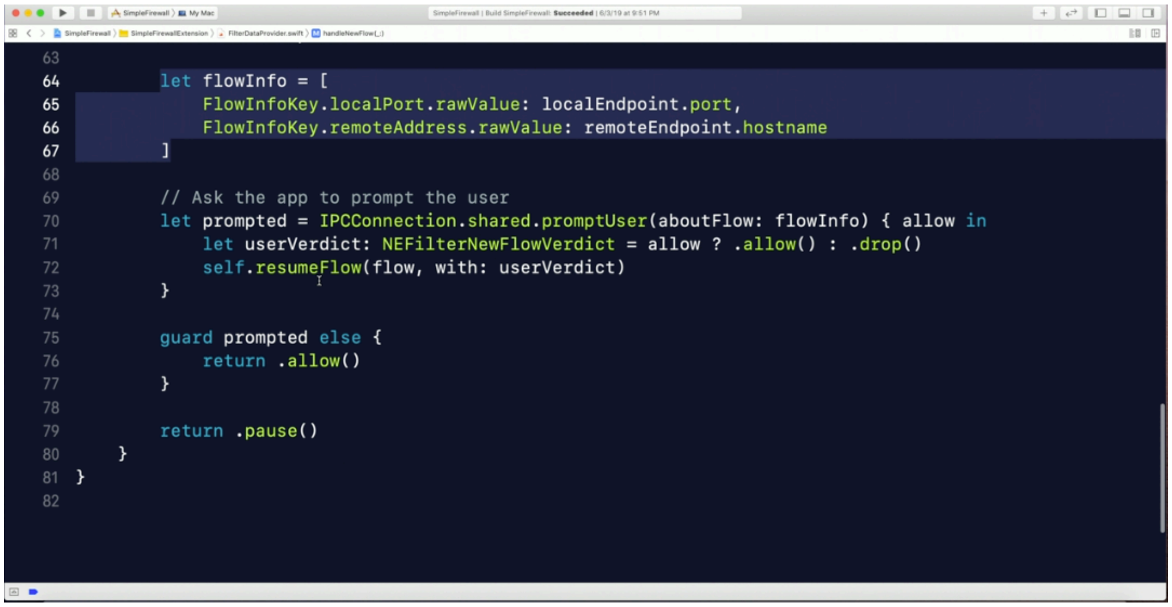
handleNewFlow: function called when new flow is created matching filter rules.



```

    override func handleNewFlow(_ flow: NEFilterFlow) -> NEFilterNewFlowVerdict {

        guard let socketFlow = flow as? NEFilterSocketFlow,
            let remoteEndpoint = socketFlow.remoteEndpoint as? NWHostEndpoint,
            let localEndpoint = socketFlow.localEndpoint as? NWHostEndpoint else {
                return .allow()
        }

        os_log("Got a new flow with local endpoint %@, remote endpoint %@", localEndpoint,
            remoteEndpoint)

        let flowInfo = [
            FlowInfoKey.localPort.rawValue: localEndpoint.port,
            FlowInfoKey.remoteAddress.rawValue: remoteEndpoint.hostname
        ]

        // Ask the app to prompt the user
        let prompted = IPCConnection.shared.promptUser(aboutFlow: flowInfo) { allow in
            let userVerdict: NEFilterNewFlowVerdict = allow ? .allow() : .drop()
            self.resumeFlow(flow, with: userVerdict)
        }

```

- Accepts **NEFilterFlow** object as argument
- Returns **NEFilterNewFlowVerdict**
- **flowInfo:** Packags up some flow details in a dictionary.
- **prompted:** Sends flowInfo details to ui to prompt user for permission.
- **return .pause():** pauses app while waiting for user to grant permission.

handleNewFlow: (second half)



```swift

        let flowInfo = [
            FlowInfoKey.localPort.rawValue: localEndpoint.port,
            FlowInfoKey.remoteAddress.rawValue: remoteEndpoint.hostname
        ]

        // Ask the app to prompt the user
        let prompted = IPCConnection.shared.promptUser(aboutFlow: flowInfo) { allow in
            let userVerdict: NEFilterNewFlowVerdict = allow ? .allow() : .drop()
            self.resumeFlow(flow, with: userVerdict)
        }

        guard prompted else {
            return .allow()
        }

        return .pause()
    }
}

```