OVERSECURED

oversecured.com

A mobile application vulnerability scanner, designed for DevOps process integration, that is built to protect your customers' privacy and defend their devices against modern threats.

ovia-debug.zip

App ID	oversecured.OVIA
Version	1.0

Statistics

17 vulnerabilities found

High severity vulnerabilities	6	Can lead to user data being leaked and to a serious violation of the user's privacy	36%
Medium severity vulnerabilities	6	May affect the app's level of protection and reduce the level of security	34%
Low severity vulnerabilities	5	Violate best practices	30%

List of vulnerabilities

	Category	Level	Amount
1	Memory corruption	● High	1
2	Overwriting arbitrary files	● High	1
3	Cross-site scripting	● High	1
4	Injection of arbitrary HTML code	● High	1
5	Storing sensitive information in a public directory	● High	1
6	Hardcoded cryptographic key	● High	1
7	Insecure App Transport Security configuration	● Medium	1
8	Fake HTTP request	● Medium	1

9	Fake file path	● Medium	1
10	Usage of deeplinks	● Medium	1
11	Enabled iTunes file sharing	● Medium	1
12	Remote debugging of WebView is enabled	● Medium	1
13	Enabled JavaScript	● Low	1
14	Weak cryptographic algorithms	● Low	4

Vulnerabilities in the code

Memory corruption

Found in the file **OVIA/AppDelegate.swift**

```
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",
                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)
                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Vulnerability description

The attacker controls a native address that the application dereferences or otherwise uses, which may lead to memory corruption in the application. In some cases this error may be raised prior to the execution of arbitrary code.

Remediation

It's recommended to take measures to prevent the attacker from controlling native addresses in any way possible.

Links

https://cwe.mitre.org/data/definitions/822.html

Overwriting arbitrary files

Found in the file **OVIA/AppDelegate.swift**

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Extensions/Foundation+Extensions.swift**

```swift
import Foundation

extension FileManager {
    class func documentsUrlForFile(withName name: String) -> URL {
        let documents = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)[0]
        return documents.appendingPathComponent(name)
    }
}

```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```swift
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",
                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)
                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Vulnerability description

An attacker has the ability to write arbitrary content into an arbitrary file. In most cases this can lead to forging various settings, user sessions, and history.

Remediation

Prevent the attacker from being able to control the path to the file, for example by encoding the file name or verifying that the file obtained is in the expected directory.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/23.html

Cross-site scripting

Found in the file **OVIA/AppDelegate.swift**

```
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",
                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)

                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Found in the file **OVIA/Controller/WebView/WebViewController.swift**

```
    }

    private var request: URLRequest {
        var request = URLRequest(url: url)
        self.extraHeaders.forEach{ name, value in
            request.setValue(value, forHTTPHeaderField: name)
        }
        return request
    }

    func set(url: URL) {
        self.url = url
    }

    override func loadView() {
        let config = WKWebViewConfiguration()
        config.defaultWebpagePreferences.allowsContentJavaScript = true

        webView = WKWebView(frame: .zero, configuration: config)
        webView.allowsBackForwardNavigationGestures = true
        webView.configuration.preferences.setValue(true, forKey: "developerExtrasEnabled")
        view = webView

        webView.load(self.request)
    }
}

```

Vulnerability description

XSS or Cross-site scripting is a kind of attack where malicious scripts are inserted into a WebView page. An attacker can influence a URL that is loaded, JavaScript code that is executed, or files that are stored in public directories. In this event, harmful code may be processed and executed by the built-in browser. Execution of malicious scripts might cause unintended information leakage, modification of settings on the server side via bypassed CSRF protection, and more. On iOS there is also a risk of access to built-in WebView handlers such as script messages, URL handlers or content intended for internal communications by the web page and the mobile app, thus exposing internal app logic and functionality.

Remediation

Before insertion, client data should be correctly validated and sanitized. In this case, all metacharacters will be escaped. In other cases, XSS is the result of insecure application architecture, when it trusts data received from unprotected inputs.

Links

https://www.owasp.org/index.php/Mobile_Top_10_2014-M7 https://arxiv.org/ftp/arxiv/papers/1304/1304.7451.pdf

https://cwe.mitre.org/data/definitions/79.html

Injection of arbitrary HTML code

Found in the file **OVIA/AppDelegate.swift**

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```swift
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",
                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)
                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Found in the file **OVIA/Extensions/Foundation+Extensions.swift**

```swift
}

extension NSAttributedString {
    convenience init?(htmlString: String) {
        guard let data = htmlString.data(using: .utf8) else { return nil }
        guard let attributedString = try? NSAttributedString(data: data, options: [.documentType: NSAttributedString.DocumentType.html], documentAttributes: nil) else {
            return nil
        }
        self.init(attributedString: attributedString)
    }
}

```

Found in the file **OVIA/Utils/Alerts.swift**

```swift
        self.alertView.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
    }

    init(title: String, message: NSAttributedString) {
        self.alertView = UIAlertController(title: title, message: "", preferredStyle: .alert)
        self.alertView.setValue(message, forKey: "attributedMessage")
        self.alertView.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
    }

```

Vulnerability description

Remediation

The ability to add arbitrary HTML code in WebView or NSAttributedString can be used by an attacker to create false authentication forms and fake links or formatted events that seem to come from the app. This can lead to the theft of account data, or encourage the user to take action that benefits the attacker.

The app should not render content from untrusted sources, or else it should give some indication that they are untrusted.

Links

https://cwe.mitre.org/data/definitions/79.html

Storing sensitive information in a public directory

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```

        @IBAction func dumpDataAction(_ sender: UIButton) {
            directoryPicker = DirectoryPicker(self)
            directoryPicker.pick{ url in
                guard let url = url else {
                    return
                }
                Files.dumpCacheFile(to: url)
            }
        }

```

Found in the file **OVIA/Controller/Main/DirectoryPicker.swift**

```
}

extension DirectoryPicker: UIDocumentPickerDelegate {
    func documentPicker(_ controller: UIDocumentPickerViewController, didPickDocumentsAt urls: [URL]) {
        self.handler(urls.first)
    }

    func documentPickerWasCancelled(_ controller: UIDocumentPickerViewController) {
```

Found in the file **OVIA/Utils/Files.swift**

```
import Foundation

fileprivate let cacheFileLocation = "oversecured.OVIA/Cache.db"

class Files {
    private init() {
    }

    static func dumpCacheFile(to toDir: URL) {
        guard toDir.startAccessingSecurityScopedResource() else {
            return
        }
        defer {
            toDir.stopAccessingSecurityScopedResource()
        }
        let cacheFile = getCacheDirectoryPath().appendingPathComponent(cacheFileLocation)
        let destination = toDir.appendingPathComponent(cacheFile.lastPathComponent)
        try! FileManager.default.copyItem(at: cacheFile, to: destination)
    }

    static func getCacheDirectoryPath() -> URL {
```

Vulnerability description

The application uses a public directory to read/write data. The data stored in here can be read and modified by any third-party applications installed on the same iOS device, which can lead to information leaks, data tampering, or other security breaches. It was determined that the application stored scripts, or files with potentially sensitive content.

Remediation

Stop storing configuration files, databases, and sensitive user data in a public directory.

Links

https://cwe.mitre.org/data/definitions/921.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

Hardcoded cryptographic key

Found in the file **OVIA/Utils/Crypto.swift**

```
import Foundation
import CommonCrypto

fileprivate let aesKey = "gEKC8qte1FvR3oJV"
fileprivate let iv = "MFQOKFtxfibUuDWo"

class Crypto {
    private static let aesInstance = AESCrypto(key: aesKey, iv: iv)

    private init() {
    }

    static func encrypt(_ string: String) -> String {
        return aesInstance.encrypt(string: string).base64EncodedString()
    }

    static func decrypt(_ data: String) -> String {
        return aesInstance.decrypt(data: Data(base64Encoded: data)!)
    }
}

fileprivate class AESCrypto {
    private let key: Data
    private let iv: Data

    init(key: String, iv: String) {
        guard key.count == kCCKeySizeAES128, let keyData = key.data(using: .utf8) else {
            preconditionFailure("Error: Failed to set a key")
        }
        guard iv.count == kCCBlockSizeAES128, let ivData = iv.data(using: .utf8) else {
            preconditionFailure("Error: Failed to set an initial vector")
        }
        self.key = keyData
        self.iv  = ivData
    }

    func encrypt(string: String) -> Data {
```

Found in the file **OVIA/Utils/Crypto.swift**

```

        let status = cryptData.withUnsafeMutableBytes { cryptBytes in
            data.withUnsafeBytes { dataBytes in
                iv.withUnsafeBytes { ivBytes in
                    key.withUnsafeBytes { keyBytes in
                        CCCrypt(option, CCAlgorithm(kCCAlgorithmAES), options, keyBytes.baseAddress, keyLength,
ivBytes.baseAddress, dataBytes.baseAddress, data.count, cryptBytes.baseAddress, cryptLength, &bytesLength)
                    }
                }
            }
```

Vulnerability description

The cryptographic key is hardcoded in the app. It can be used by an attacker to encrypt or decrypt sensitive data, substitute a different digital signature, etc., reducing this level of data security to nil.

Remediation

The developer shouldn't hardcode encryption keys. Instead, we recommend using secure key creation and storage systems such as the Keychain services API.

Links

https://developer.android.com/training/articles/keystore

https://cwe.mitre.org/data/definitions/312.html

Insecure App Transport Security configuration

Found in the file **OVIA/Info.plist**

```
        </array>
        <key>UIFileSharingEnabled</key>
        <true/>
        <key>NSAppTransportSecurity</key>
        <dict>
                <key>NSAllowsArbitraryLoads</key>
                <true/>
        </dict>
</dict>
</plist>
```

Vulnerability description

The app uses ATS settings that permit an insecure internet connection over HTTP or old versions of SSL. This may lead to Man-in-the-Middle attacks resulting in sensitive user data being leaked and/or modified.

Remediation

You should update the server settings to work via secure data transfer protocols, then be sure to remove the insecure ATS settings.

Links

https://cwe.mitre.org/data/definitions/319.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

---

Fake HTTP request

Found in the file **OVIA/AppDelegate.swift**

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```swift
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",

                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)
                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Found in the file **OVIA/Controller/WebView/WebViewController.swift**

```
        }

        private var request: URLRequest {
            var request = URLRequest(url: url)
            self.extraHeaders.forEach{ name, value in
                request.setValue(value, forHTTPHeaderField: name)
            }
            return request
        }

        func set(url: URL) {
            self.url = url
        }

        override func loadView() {
```

Vulnerability description

The app makes it possible to fake a request, or some of its fields, sent to HTTP, which can lead to a whole series of possible attacks including Cross-Site Request Forgery and HTTP Splitting. This can make it possible to uncover used data, and can also damage the app's business logic by carrying out actions that benefit the attacker.

Remediation

The developer must restrict request data to trusted sources, and make sure these data are necessary, are in the expected format, and do not contain special characters that would violate the structure of an HTTP request.

Links

https://cwe.mitre.org/data/definitions/352.html

https://owasp.org/www-community/attacks/csrf

https://portswigger.net/web-security/request-smuggling

Fake file path

Found in the file **OVIA/AppDelegate.swift**

```
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        if Auth.auth().signedIn {
            let mainViewController = MainViewController.styled
            let deeplink = launchOptions?[.url] as? URL
            mainViewController.set(deeplink: deeplink)
            window?.rootViewController = mainViewController
        }
        else {
```

Found in the file **OVIA/Extensions/Foundation+Extensions.swift**

```
import Foundation

extension FileManager {
    class func documentsUrlForFile(withName name: String) -> URL {
        let documents = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)[0]
        return documents.appendingPathComponent(name)
    }
}

```

Found in the file **OVIA/Controller/Main/MainViewController.swift**

```
        return storyboard.instantiateViewController(withIdentifier: "mainStoryboard") as! MainViewController
    }

    func set(deeplink: URL?) {
        self.deeplink = deeplink
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        if let deeplink = deeplink {
            handleDeeplink(deeplink)
        }
    }

    @IBAction func signOutAction(_ sender: UIButton) {
        Auth.auth().signOut()

        let loginViewController = LoginViewController.styled
        present(loginViewController, animated: true)
    }

    @IBAction func dumpDataAction(_ sender: UIButton) {
        directoryPicker = DirectoryPicker(self)
        directoryPicker.pick{ url in
            guard let url = url else {
                return
            }
            Files.dumpCacheFile(to: url)
        }
    }

    private func handleDeeplink(_ url: URL) {
        let route = url.lastPathComponent
        if route == "alert", let text = getQueryParameter(url, "message"), let message = NSAttributedString(htmlString: text) {
```

```swift
            Alert(title: "Alert!", message: message).display(from: self)
        }
        else if route == "webview", let urlParam = getQueryParameter(url, "url"), let url = URL(string: urlParam) {
            let webVc = WebViewController()
            webVc.set(url: url)
            present(webVc, animated: true)
        }
        else if route == "save",
                let dataParam = getQueryParameter(url, "data"),
                var data = Data(base64Encoded: dataParam),
                let name = getQueryParameter(url, "name") {

            if let offsetParam = getQueryParameter(url, "offset"), let offset = Int(offsetParam) {
                let arr = [UInt8](data)
                data = Data(bytes: &data + offset, count: arr.count - offset)
            }

            try! data.write(to: FileManager.documentsUrlForFile(withName: name))
        }
    }

    func getQueryParameter(_ url: URL, _ name: String) -> String? {
        if let components = NSURLComponents(url: url, resolvingAgainstBaseURL: true), let params = components.queryItems {
            if let value = params.first(where: { $0.name == name })?.value {
                return value
            }
        }
        return nil
```

Vulnerability description

An attacker has the ability to control the path to a file, which can lead to private data being stored in a public directory to which the attacker has access; data coming from the attacker being read as though they were legitimate; modification or deletion of existing files. The developer must remember that even if the attacker has no access to certain protected files, the legitimate app does — so that the attacker's objective is to make the application carry out harmful actions on its own.

Remediation

The developer must make sure file paths can only be obtained from trusted sources.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/22.html

Usage of deeplinks

Found in the file **OVIA/Info.plist**

```xml
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>CFBundleURLTypes</key>
        <array>
                <dict>
                        <key>CFBundleTypeRole</key>
                        <string>Editor</string>
                        <key>CFBundleURLName</key>
                        <string>deeplink</string>
                        <key>CFBundleURLSchemes</key>
                        <array>
                                <string>ovia</string>
                        </array>
                </dict>
        </array>
        <key>UIFileSharingEnabled</key>
        <true/>
        <key>NSAppTransportSecurity</key>
```

Vulnerability description

The app can receive and process files and URLs from external sources, which can lead to a wide range of security problems: injections of various kinds, path traversal, leaking of user sessions and even arbitrary code execution.

Remediation

It is important not to trust data obtained from external sources. Such data should be subjected to the maximum possible verification. It is worth refraining from passing URL data via deeplinks: instead, describe the set of various actions that the user can perform.

Links

https://cwe.mitre.org/data/definitions/939.html

Enabled iTunes file sharing

Found in the file **OVIA/Info.plist**

```
                                </array>
                        </dict>
                </array>
                <key>UIFileSharingEnabled</key>
                <true/>
                <key>NSAppTransportSecurity</key>
                <dict>
                        <key>NSAllowsArbitraryLoads</key>
```

Vulnerability description

iTunes file sharing is turned on in the app, which may lead to app data being leaked. When this option is turned on, the entire Documents directory is used to exchange files. An attacker can use physical access to the iOS device to gain access to them.

Remediation

Make sure files containing sensitive information are not copied to the Documents directory. If your app does not need this functionality, set the flag to false or delete the option.

Links

https://cwe.mitre.org/data/definitions/359.html

https://developer.apple.com/documentation/bundleresources/information_property_list/uifilesharingenabled

---

Remote debugging of WebView is enabled

Found in the file **OVIA/Controller/WebView/WebViewController.swift**

```

        webView = WKWebView(frame: .zero, configuration: config)
        webView.allowsBackForwardNavigationGestures = true
        webView.configuration.preferences.setValue(true, forKey: "developerExtrasEnabled")
        view = webView

        webView.load(self.request)
```

Vulnerability description

The WebView component is started in a way that allows remote debugging, permitting an attacker with physical access to the device to switch the page of any website opened from the app.

Remediation

Disallow remote debugging of the WebView component by calling myWebView.configuration.preferences.setValue(false, forKey: "developerExtrasEnabled").

Links

https://cwe.mitre.org/data/definitions/489.html

Enabled JavaScript

Found in the file **OVIA/Controller/WebView/WebViewController.swift**

```

    override func loadView() {
        let config = WKWebViewConfiguration()
        config.defaultWebpagePreferences.allowsContentJavaScript = true

        webView = WKWebView(frame: .zero, configuration: config)
        webView.allowsBackForwardNavigationGestures = true
```

Vulnerability description

The WebView has JavaScript enabled. It allows the execution of JavaScript in the context of a running application. Performing a Man-in-the-Middle attack or tampering with a server response, an attacker is able to inject and execute arbitrary JavaScript code. This can lead to information leakage or remote code execution. It's not recommended to enable JavaScript unless absolutely necessary. Disable this setting to enforce security.

Remediation

Disable JavaScript, or make sure the server uses an encrypted channel (using https and correct certificate verification) and there are no vulnerabilities in the server part of the application.

Links

https://cwe.mitre.org/data/definitions/79.html

Weak cryptographic algorithms

Found in the file **OVIA/Utils/Crypto.swift**

```
            data.withUnsafeBytes { dataBytes in
                iv.withUnsafeBytes { ivBytes in
                    key.withUnsafeBytes { keyBytes in
                        CCCrypt(option, CCAlgorithm(kCCAlgorithmAES), options, keyBytes.baseAddress, keyLength, ivBytes.baseAddress, dataBytes.baseAddress, data.count, cryptBytes.baseAddress, cryptLength, &bytesLength)
                    }
                }
            }
```

Vulnerability description

The application uses insecure or weak cryptographic algorithms, or such algorithms are used with an insufficiently long encryption key. This creates certain vulnerabilities which mean encrypted data can be decrypted by an attacker, for example

Remediation

Use more reliable encryption algorithms, such as AES with a key length of at least 256 or RSA with a key length of at least 2048.

by means of a brute force attack.

Links

https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

https://www.keylength.com/en/4/

https://cwe.mitre.org/data/definitions/327.html

Weak cryptographic algorithms

Found in the file **OVIA/Utils/Crypto.swift**

```
        guard key.count == kCCKeySizeAES128, let keyData = key.data(using: .utf8) else {
            preconditionFailure("Error: Failed to set a key")
        }
        guard iv.count == kCCBlockSizeAES128, let ivData = iv.data(using: .utf8) else {
            preconditionFailure("Error: Failed to set an initial vector")
        }
        self.key = keyData
```

Vulnerability description

The application uses insecure or weak cryptographic algorithms, or such algorithms are used with an insufficiently long encryption key. This creates certain vulnerabilities which mean encrypted data can be decrypted by an attacker, for example by means of a brute force attack.

Remediation

Use more reliable encryption algorithms, such as AES with a key length of at least 256 or RSA with a key length of at least 2048.

Links

https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

https://www.keylength.com/en/4/

https://cwe.mitre.org/data/definitions/327.html

Weak cryptographic algorithms

Found in the file **OVIA/Utils/Crypto.swift**

```
    private let iv: Data

    init(key: String, iv: String) {
        guard key.count == kCCKeySizeAES128, let keyData = key.data(using: .utf8) else {
            preconditionFailure("Error: Failed to set a key")
        }
        guard iv.count == kCCBlockSizeAES128, let ivData = iv.data(using: .utf8) else {
```

Vulnerability description

The application uses insecure or weak cryptographic algorithms, or such algorithms are used with an insufficiently long encryption key. This creates certain vulnerabilities which mean encrypted data can be decrypted by an attacker, for example by means of a brute force attack.

Remediation

Use more reliable encryption algorithms, such as AES with a key length of at least 256 or RSA with a key length of at least 2048.

Links

https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography https://www.keylength.com/en/4/

https://cwe.mitre.org/data/definitions/327.html

Weak cryptographic algorithms

Found in the file **OVIA/Utils/Crypto.swift**

```
        }

        func crypt(data: Data, option: CCOperation) -> Data {
            let cryptLength = data.count + kCCBlockSizeAES128
            var cryptData   = Data(count: cryptLength)

            let keyLength = key.count
```

Vulnerability description

The application uses insecure or weak cryptographic algorithms, or such algorithms are used with an insufficiently long encryption key. This creates certain vulnerabilities which mean encrypted data can be decrypted by an attacker, for example by means of a brute force attack.

Remediation

Use more reliable encryption algorithms, such as AES with a key length of at least 256 or RSA with a key length of at least 2048.

Links

https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

https://www.keylength.com/en/4/

https://cwe.mitre.org/data/definitions/327.html