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.

Scan report dated **02/20/2022**

iGoat-Swift.zip

App ID OWASP.iGoat-Swifth Version 1.0

Statistics

36 vulnerabilities found

High-severity vulnerabilities 7

Can lead to user data being leaked and to a serious violation of the user's privacy

20%

Medium-severity vulnerabilities 13

May affect the app's level of protection and reduce the level of security

37%

Low-severity vulnerabilities 16

Violate best practices

43%

List of vulnerabilities

	Category	Level	Amount
1	Cross-site scripting	High	1
2	Hardcoded token or password	High	2
3	Hardcoded cryptographic key	High	1
4	SQL injection	High	1
5	Possible cache substitution	High	2
6	Insecure App Transport Security configuration	Medium	1

7	Ability to enter unverified information into databases	Medium	3
8	Usage of deeplinks	Medium	1
9	Content injection	Medium	1
10	Use of insecure HTTP protocol	Medium	7
11	Weak cryptographic algorithms	Low	9
12	Local network address	Low	7

Vulnerabilities in the code

Cross-site scripting

Found in the file **iGoat-Swift/Source/Exercises/Injection Flaws/XSS/CrossSiteScriptingExerciseVC.swift**

```
5       @IBOutlet weak var textField: UITextField!
6
7       @IBAction func loadItemPressed() {
8           let webText = "Welcome to XSS Exercise !!! \n Here is UIWebView ! \(textField.text ?? "")"
9           webview.loadHTMLString(webText, baseURL: nil)
10      }
11  }
12
```

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

Hardcoded token or password

Found in the file **iGoat-Swift/ThirdParty/CouchBase/CouchbaseLite.framework/Headers/CBLReplication.h**

```
101  @property (nonatomic, strong, nullable) id<CBLAuthenticator> authenticator;
102
103  /** The credential (generally username+password) to use to authenticate to the remote database.
104      This can either come from the URL itself (if it's of the form "http://user:pass@example.com")
105      or be stored in the NSURLCredentialStorage, which is a wrapper around the Keychain. */
106  @property (nonatomic, strong, nullable) NSURLCredential* credential;
107
```

Vulnerability description

A token or password was found. It might be used by an attacker to access restricted services which will cause information leakage, unwanted server setting changes, or other kinds of unrestricted service accesses or modifications.

Remediation

The developer should not hardcode such sensitive data, to prevent leakages.

Links

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

Hardcoded token or password

Found in the file **iGoat-Swift/Source/Exercises/Authentication/Remote/RemoteAuthenticationExerciseVC.swift**

```
3  enum RAConstants {
4      enum EndPoints {
5          static func loginUser(name: String, password: String) -> String {
6              return "http://localhost:8080/igoat/token?username=\(name)&password=\(password)"
7          }
8      }
9  }
```

Vulnerability description

A token or password was found. It might be used by an attacker to access restricted services which will cause information leakage, unwanted server setting changes, or other kinds of unrestricted service accesses or modifications.

Remediation

The developer should not hardcode such sensitive data, to prevent leakages.

Links

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

Hardcoded cryptographic key

Found in the file **iGoat-Swift/Source/Extensions/Data+Extension.swift**

```
1    import Foundation
2
3    extension Data{
4        func aes(operation: Int = kCCEncrypt, keyData: Data,
5                         ivData: Data = "abcdefghijklmnop".data(using:String.Encoding.utf8)!
6                         ) -> Data {
7            let dataLength = self.count
8            let cryptLength  = size_t(dataLength + kCCBlockSizeAES128)
9            var cryptData = Data(count:cryptLength)
10
11           let keyLength = size_t(kCCKeySizeAES128)
12           let options = CCOptions(kCCOptionPKCS7Padding)
13
14           var numBytesEncrypted :size_t = 0
15
16           let cryptStatus = cryptData.withUnsafeMutableBytes {cryptBytes in
17               self.withUnsafeBytes {dataBytes in
18                   ivData.withUnsafeBytes {ivBytes in
19                       keyData.withUnsafeBytes {keyBytes in
20                           CCCrypt(CCOperation(operation),
21                                   CCAlgorithm(kCCAlgorithmAES),
22                                   options,
23                                   keyBytes, keyLength,
24                                   ivBytes,
25                                   dataBytes, dataLength,
26                                   cryptBytes, cryptLength,
27                                   &numBytesEncrypted)
```

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Hard Coded Keys/BrokenCryptographyExerciseVC.swift**

```
7        }
8
9        class BrokenCryptographyExerciseVC: UIViewController {
10           var encryptionKey = "myencrytionkey"
11           @IBOutlet weak var passwordTextField: UITextField!
12
13           @IBAction func showItemPressed() {
```

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Hard Coded Keys/BrokenCryptographyExerciseVC.swift**

```
34           let data = password.data(using: .utf8)
35           print(data!)
36
37           let encryptionKeyData = encryptionKey.data(using: .utf8)
38           let encryptedData = data?.aes(operation: kCCEncrypt, keyData: encryptionKeyData!)
39           let url = URL(fileURLWithPath: pathDocumentDirectory(fileName: "encrypted"))
40           try? encryptedData?.write(to: url, options: .atomic)
41       }
```

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://blog.oversecured.com/Use-cryptography-in-mobile-apps-the-right-way/

https://developer.android.com/training/articles/keystore https://cwe.mitre.org/data/definitions/312.html

SQL injection

Found in the file **iGoat-Swift/Source/Exercises/Injection Flaws/SQL Injection/SQLInjectionExerciseVC.swift**

```
14
15          var searchStr = "%"
16          if !(searchField.text?.isEmpty ?? true) {
17               searchStr = "%" + "\(searchField.text!)" + "%"
18          }
19
20          let query = "SELECT title FROM article WHERE title LIKE '\(searchStr)' AND premium=0"
21          var stmt: OpaquePointer?
22          sqlite3_prepare_v2(db, query, -1, &stmt, nil)
23          var articleTitles = [String]()
24          while sqlite3_step(stmt) == SQLITE_ROW {
25              let title = String(cString: sqlite3_column_text(stmt, 0))
```

Vulnerability description

An SQL injection in SQLite databases attack consists of the insertion of an SQL query via the input data from the client, a malicious application, or another compromising input to the target application. A successful SQL injection exploit can read, modify (insert/update/delete) sensitive data, or break the query logic. In SQL injection, commands are put into query input in order to change the predefined commands. An attacker can execute arbitrary SQL statements if the SQL query is concatenated with data received from unsafe sources. It may cause information damage or leakage.

Remediation

The application should filter metacharacters from input and not incorporate input without sanitizing/escaping it properly. Ideally you would be utilizing stored procedures/parameterized queries and a database layer abstraction that doesn't allow SQL injections.

Links

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html https://hub.packtpub.com/knowing-sql-injection-attacks-and-securing-our-android-applications-them/ https://cwe.mitre.org/data/definitions/89.html

Possible cache substitution

Found in the file **iGoat-Swift/Source/Exercises/Server Communication/ServerCommunicationExerciseVC.swift**

```
18
19      @IBAction func submit(_ sender: Any) {
20
21          let accountInfo = ["firstName":firstNameField.text,
22                             "lastName":lastNameField.text,
23                             "socialSecurityNumber":ssnField.text]
24
25          do {
26              //resign first responder before attempting to submit data.
27              firstNameField.resignFirstResponder()
28              lastNameField.resignFirstResponder()
```

```
29              ssnField.resignFirstResponder()
30
31              let jsonData = try JSONSerialization.data(withJSONObject: accountInfo, options: .prettyPrinted)
32              var request = URLRequest(url: URL(string: serverCommunicationUserUrl)!)
33              request.httpMethod = "POST"
34              request.setValue("application/json", forHTTPHeaderField: "Content-Type")
35              request.setValue("application/json", forHTTPHeaderField: "Accept")
36              request.setValue("close", forHTTPHeaderField: "Connection")
37              request.httpBody = jsonData
38
39              URLSession(configuration: .default).dataTask(with: request, completionHandler: { (data, response, error) in
40
41                  if let sslEnabled = (response as? HTTPURLResponse)?.allHeaderFields["X-Goat-Secure"] as? String {
42                      if sslEnabled.boolValue {
```

Vulnerability description

An attacker has the ability to substitute the cache of HTTP requests, which can lead to a whole series of possible attacks. For example, an attacker may replace the cache content for any URL, meaning that when the user tries to visit that address they will see content prepared by the attacker in place of the original. In its turn, this can lead to the user being defrauded or to other attacks being prepared against the device.

Remediation

Restrict the ability of third-parties to affect the cache.

Links

https://capec.mitre.org/data/definitions/141.html

Possible cache substitution

Found in the file **iGoat-Swift/Source/Exercises/Authentication/Remote/RemoteAuthenticationExerciseVC.swift**

```
2
3   enum RAConstants {
4       enum EndPoints {
5           static func loginUser(name: String, password: String) -> String {
6               return "http://localhost:8080/igoat/token?username=\(name)&password=\(password)"
7           }
8       }
9   }
10
11  class RemoteAuthenticationExerciseVC: UIViewController {
12      @IBOutlet weak var usernameTextField: UITextField!
13      @IBOutlet weak var passwordTextField: UITextField!
14
15      @IBAction func submitItemPressed() {
16          if usernameTextField.text?.isEmpty ?? true {
17              UIAlertController.showAlertWith(title: "iGoat", message: "Username Field empty!!")
18              return
19          } else if passwordTextField.text?.isEmpty ?? true {
20              UIAlertController.showAlertWith(title: "iGoat", message: "Password Field empty!!")
21              return
22          }
23
24          let username = usernameTextField.text ?? ""
```

```
25          let password = passwordTextField.text ?? ""
26
27          let urlString = RAConstants.EndPoints.loginUser(name: username, password: password)
28          guard let url = URL(string: urlString) else {
29              print("Error: cannot create URL")
30              return
31          }
32
33          hitRequest(withURL: url)
34      }
35  }
36
37  extension RemoteAuthenticationExerciseVC {
38      func hitRequest(withURL url: URL) {
39          let urlRequest = URLRequest(url: url)
40          let config = URLSessionConfiguration.default
41          let session = URLSession(configuration: config)
42          SVProgressHUD.show()
43          let task = session.dataTask(with: urlRequest, completionHandler: { (data, response, error) in
44              DispatchQueue.main.async {
45                  SVProgressHUD.dismiss()
46                  if let response = response as? HTTPURLResponse {
```

Vulnerability description

An attacker has the ability to substitute the cache of HTTP requests, which can lead to a whole series of possible attacks. For example, an attacker may replace the cache content for any URL, meaning that when the user tries to visit that address they will see content prepared by the attacker in place of the original. In its turn, this can lead to the user being defrauded or to other attacks being prepared against the device.

Remediation

Restrict the ability of third-parties to affect the cache.

Links

https://capec.mitre.org/data/definitions/141.html

Insecure App Transport Security configuration

Found in the file **iGoat-Swift/Info.plist**

```
33              <string>1</string>
34              <key>LSRequiresIPhoneOS</key>
35              <true/>
36              <key>NSAppTransportSecurity</key>
37              <dict>
38                      <key>NSAllowsArbitraryLoads</key>
39                      <true/>
40              </dict>
41              <key>UILaunchStoryboardName</key>
42              <string>LaunchScreen</string>
```

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

Remediation

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

leaked and/or modified.

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

---

Ability to enter unverified information into databases

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Random Key Generation/RandomKeyGenerationExerciseVC.swift**

```
17
18      @IBAction func submit(_ sender: Any) {
19          let username = usernameField.text
20          let password = passwordField.text
21          if credentialStorageSwitch.isOn {
22              storeCredentials(forUsername: username, withPassword: password)
23          }
24          usernameField.text = ""
25          passwordField.text = ""
26      }
27
28      func storeCredentials(forUsername username: String?, withPassword password: String?) {
29          // Write the credentials to a SQLite database.
30          var credentialsDB: OpaquePointer?
31          let path = pathDocumentDirectory(fileName: "credentials.sqlite")
```

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Random Key Generation/RandomKeyGenerationExerciseVC.swift**

```
62
63              if sqlite3_prepare_v2(credentialsDB, addUpdateStmt, -1, &compiledStmt, nil) == SQLITE_OK {
64                  sqlite3_bind_text(compiledStmt, 1, username, -1, SQLITE_TRANSIENT)
65                  sqlite3_bind_text(compiledStmt, 2, password, -1, SQLITE_TRANSIENT)
66                  if userID >= 0 {
67                      sqlite3_bind_int(compiledStmt, 3, Int32(userID))
68                  }
```

Vulnerability description

The ability to enter arbitrary data into certain databases/tables/fields creates the likelihood that a legitimate app will use them in the future. They may also be displayed on the user's screen, making him think this is legitimate server content, or transmitted to the server — allowing the attacker to change the user's profile settings or add content without the user's knowledge.

Remediation

The app should check that the data come either from the end user or from a trusted source. The ability to receive or store data outside the app must be limited to those sources for which it's really necessary.

Links

https://cwe.mitre.org/data/definitions/346.html

Ability to enter unverified information into databases

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Random Key Generation/RandomKeyGenerationExerciseVC.swift**

```
16      }
17
18      @IBAction func submit(_ sender: Any) {
19          let username = usernameField.text
20          let password = passwordField.text
21          if credentialStorageSwitch.isOn {
22              storeCredentials(forUsername: username, withPassword: password)
23          }
24          usernameField.text = ""
25          passwordField.text = ""
26      }
27
28      func storeCredentials(forUsername username: String?, withPassword password: String?) {
29          // Write the credentials to a SQLite database.
30          var credentialsDB: OpaquePointer?
31          let path = pathDocumentDirectory(fileName: "credentials.sqlite")
32          if sqlite3_open(path, &credentialsDB) == SQLITE_OK {
33              var compiledStmt: OpaquePointer?
34              let deviceID = UIDevice.current.identifierForVendor?.uuidString
35              if let aVendor = UIDevice.current.identifierForVendor {
36                  print("encryption key is \(aVendor)")
37              }
38              let encrptStmt = "PRAGMA key = '\(deviceID ?? "")'"
39              sqlite3_exec(credentialsDB, encrptStmt, nil, nil, nil)
40              // Create the table if it doesn't exist.
41              let createStmt = "CREATE TABLE IF NOT EXISTS creds (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password
    TEXT);"
42
43
44              sqlite3_exec(credentialsDB, createStmt, nil, nil, nil)
45              // Check to see if the user exists; update if yes, add if no.
46              let queryStmt = "SELECT id FROM creds WHERE username=?"
47              var userID: Int = -1
48              if sqlite3_prepare_v2(credentialsDB, queryStmt, -1, &compiledStmt, nil) == SQLITE_OK {
49                  sqlite3_bind_text(compiledStmt, 1, username, -1, SQLITE_TRANSIENT)
50                  while sqlite3_step(compiledStmt) == SQLITE_ROW {
51                      userID = Int(sqlite3_column_int(compiledStmt, 0))
52                  }
```

Vulnerability description

The ability to enter arbitrary data into certain databases/tables/fields creates the likelihood that a legitimate app will use them in the future. They may also be displayed on the user's screen, making him think this is legitimate server content, or transmitted to the server — allowing the attacker to change the user's profile settings or add content without the user's knowledge.

Remediation

The app should check that the data come either from the end user or from a trusted source. The ability to receive or store data outside the app must be limited to those sources for which it's really necessary.

Links

https://cwe.mitre.org/data/definitions/346.html

Ability to enter unverified information into databases

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Random Key Generation/RandomKeyGenerationExerciseVC.swift**

```
16      }
17
18      @IBAction func submit(_ sender: Any) {
19          let username = usernameField.text
20          let password = passwordField.text
21          if credentialStorageSwitch.isOn {
22              storeCredentials(forUsername: username, withPassword: password)
23          }
24          usernameField.text = ""
25          passwordField.text = ""
26      }
27
28      func storeCredentials(forUsername username: String?, withPassword password: String?) {
29          // Write the credentials to a SQLite database.
30          var credentialsDB: OpaquePointer?
31          let path = pathDocumentDirectory(fileName: "credentials.sqlite")
```

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Random Key Generation/RandomKeyGenerationExerciseVC.swift**

```
61              }
62
63              if sqlite3_prepare_v2(credentialsDB, addUpdateStmt, -1, &compiledStmt, nil) == SQLITE_OK {
64                  sqlite3_bind_text(compiledStmt, 1, username, -1, SQLITE_TRANSIENT)
65                  sqlite3_bind_text(compiledStmt, 2, password, -1, SQLITE_TRANSIENT)
66                  if userID >= 0 {
67                      sqlite3_bind_int(compiledStmt, 3, Int32(userID))
```

Vulnerability description

The ability to enter arbitrary data into certain databases/tables/fields creates the likelihood that a legitimate app will use them in the future. They may also be displayed on the user's screen, making him think this is legitimate server content, or transmitted to the server — allowing the attacker to change the user's profile settings or add content without the user's knowledge.

Remediation

The app should check that the data come either from the end user or from a trusted source. The ability to receive or store data outside the app must be limited to those sources for which it's really necessary.

Links

https://cwe.mitre.org/data/definitions/346.html

Usage of deeplinks

Found in the file **iGoat-Swift/Info.plist**

```
16              <string>APPL</string>
17              <key>CFBundleShortVersionString</key>
18              <string>1.0</string>
19              <key>CFBundleURLTypes</key>
20              <array>
21                      <dict>
```

```
22                          <key>CFBundleTypeRole</key>
23                          <string>Editor</string>
24                          <key>CFBundleURLName</key>
25                          <string>com.iGoat.myCompany</string>
26                          <key>CFBundleURLSchemes</key>
27                          <array>
28                                  <string>iGoat</string>
29                          </array>
30                  </dict>
31          </array>
32          <key>CFBundleVersion</key>
33          <string>1</string>
34          <key>LSRequiresIPhoneOS</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

Content injection

Found in the file **iGoat-Swift/AppDelegate.swift**

```
13          return true
14      }
15
16      func application(_ app: UIApplication, open url: URL,
17                       options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
18          if url.scheme?.caseInsensitiveCompare("igoat") == .orderedSame,
19              let queryInfo = url.parmetersInfo
20              {
21                  let mobileNo = queryInfo["contactNumber"] ?? ""
22              let message = queryInfo["message"] ?? ""
23              UIAlertController.showAlertWith(title: "iGoat", message: "Message \"\(message)\" sent to \(mobileNo)")
24                  return true
25          }
26          return false
```

Found in the file **iGoat-Swift/Source/Extensions/URL+Extension.swift**

```
2
3   extension URL {
4       var parmetersInfo: [AnyHashable: Any]? {
5           let urlComponents = URLComponents(url: self as URL, resolvingAgainstBaseURL: false)
6           let queryItems = urlComponents?.queryItems
7           var queryStringDictionary = [String: String]()
8           for queryItem: URLQueryItem? in queryItems ?? [URLQueryItem?]() {
9               if let name = queryItem?.name {
10                  let value = queryItem?.value != nil ? queryItem?.value : ""
11                  queryStringDictionary[name] = value
12              }
13          }
```

```
14          return queryStringDictionary
15      }
16  }
17
```

Found in the file **iGoat-Swift/Source/Extensions/UIAlertController+Extension.swift**

```
20          topViewController.present(self, animated: true, completion: completion)
21      }
22
23      @discardableResult class func showAlertWith(title: String, message: String) -> UIAlertController {
24          let alertController = UIAlertController(title: title, message: message, preferredStyle: .alert)
25          alertController.cancel()
26          alertController.showAlert()
27          return alertController
```

Vulnerability description

An attacker has the ability to install arbitrary text within the app, which may lead to phishing attacks: the user will be inclined to trust messages from the app.

Remediation

It is recommended that you add text that is controlled externally into fields like UITextField, rather than UITextView, or limit the possibility of adding such content at all. The user needs to be sure it is not a system message.

Links

https://developer.apple.com/design/human-interface-guidelines/ios/controls/text-fields/ https://wiki.owasp.org/index.php/Phishing

https://cwe.mitre.org/data/definitions/74.html

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Resources/HTML/KRvWAssociates.html**

```
12              &amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;Ken van Wyk (ken@krvw.com)
13              <br/>
14              <h1>Arxan Technologies</h1>
15              <a href="http://www.arxan.com">http://www.arxan.com</a>
16              <br/><br/>
17              Lead Developer:
18              <br/>
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Resources/HTML/rutger.html**

```
5       <body>
6           <div class="large-text">
7               <h1>Rutger</h1>
8               <a href="http://www.linkedin.com/profile/view?id=6000035">My linkedin profile</a>
9               <br/><br/>
10              Architect/Developer:
11              <br/>
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Source/Exercises/InsecureLocalDataStorage/BinaryCookies/BinaryCookiesExerciseVC.sw**

```
8
9       fileprivate
10      enum BCConstants {
11          static let endPoint = "http://www.github.com/OWASP/iGoat"
12
13          static let sessionTokenValue = "dfr3kjsdf5jkjk420544kjkll"
14          static let sessionTokenKey = "sessionKey"
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Source/Exercises/SideChannelDataLeaks/CloudMisconfiguration/CloudMisconfigurationEx**

```
16      }
17
18      func fetchCardImage() {
19          let todoEndpoint = "http://s3.us-east-2.amazonaws.com/igoat774396510/catty.gif"
20          guard let url = URL(string: todoEndpoint) else {
21              print("Error: cannot create URL")
22              return
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Resources/HTML/splash.html**

```
23      To contribute to iGoat, please contact <a href="mailto:swaroop.yermalkar@owasp.org">Swaroop</a>
24
25       <br><br>
26       For more info, visit <a href="http://www.igoatapp.com">www.igoatapp.com</a><br><br>
        Get latest updates about project <a href="https://twitter.com/owaspigoat?ref_src=twsrc%5Etfw" class="twitter-follow-button"
27  data-show-count="false">Follow @owaspigoat</a><script async src="https://platform.twitter.com/widgets.js" charset="utf-8">
    </script>
28      <br><br>
29      <center><form action="https://www.paypal.com/cgi-bin/webscr" method="post" target="_blank">
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/Resources/HTML/KRvWAssociates.html**

```
5       <body>
6           <div class="large-text">
7               <h1>KRvW Associates, LLC</h1>
```

```
8           <a href="http://www.krvw.com">http://www.krvw.com</a>
9           <br/><br/>
10          Architect:
11          <br/>
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Use of insecure HTTP protocol

Found in the file **iGoat-Swift/ThirdParty/CouchBase/CouchbaseLite.framework/Headers/CBLReplication.h**

```
101 @property (nonatomic, strong, nullable) id<CBLAuthenticator> authenticator;
102
103 /** The credential (generally username+password) to use to authenticate to the remote database.
104     This can either come from the URL itself (if it's of the form "http://user:pass@example.com")
105     or be stored in the NSURLCredentialStorage, which is a wrapper around the Keychain. */
106 @property (nonatomic, strong, nullable) NSURLCredential* credential;
107
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device—for instance, if the user is using a public WiFi network.

Remediation

Replace all http links in the application with their https equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

Weak cryptographic algorithms

Found in the file **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
41      func aesDecrypt(key:String, iv:String, options:Int = kCCOptionPKCS7Padding) -> String? {
42          if let keyData = key.data(using: String.Encoding.utf8),
43              let data = NSData(base64Encoded: self, options: .ignoreUnknownCharacters),
44              let cryptData    = NSMutableData(length: Int((data.length)) + kCCBlockSizeAES128) {
45
46              let keyLength              = size_t(kCCKeySizeAES128)
47              let operation: CCOperation = UInt32(kCCDecrypt)
```

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 **iGoat-Swift/Source/Extensions/Data+Extension.swift**

```
5                       ivData: Data = "abcdefghijklmnop".data(using:String.Encoding.utf8)!
6                       ) -> Data {
7           let dataLength = self.count
8           let cryptLength  = size_t(dataLength + kCCBlockSizeAES128)
9           var cryptData = Data(count:cryptLength)
10
11          let keyLength = size_t(kCCKeySizeAES128)
```

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 **iGoat-Swift/Source/Extensions/Data+Extension.swift**

```
8           let cryptLength  = size_t(dataLength + kCCBlockSizeAES128)
9           var cryptData = Data(count:cryptLength)
10
11          let keyLength = size_t(kCCKeySizeAES128)
12          let options = CCOptions(kCCOptionPKCS7Padding)
13
14          var numBytesEncrypted :size_t = 0
```

Vulnerability description

Remediation

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.

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 **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
5       func aesEncrypt(key:String, iv:String, options:Int = kCCOptionPKCS7Padding) -> String? {
6           if let keyData = key.data(using: String.Encoding.utf8),
7               let data = self.data(using: String.Encoding.utf8),
8               let cryptData    = NSMutableData(length: Int((data.count)) + kCCBlockSizeAES128) {
9
10              let keyLength              = size_t(kCCKeySizeAES128)
11              let operation: CCOperation = UInt32(kCCEncrypt)
```

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 **iGoat-Swift/Source/Exercises/InsecureLocalDataStorage/WebKitCache/WebkitCacheExerciseVC.swift**

```
135                     keyData.withUnsafeBytes {keyBytes in
136
137                         CCCrypt(CCOperation(kCCDecrypt),
138                                 CCAlgorithm(kCCAlgorithmAES128),
139                                 options,
140                                 keyBytes, keyLength,
141                                 nil,
```

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

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.

encrypted data can be decrypted by an attacker, for example 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 **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
9
10              let keyLength              = size_t(kCCKeySizeAES128)
11              let operation: CCOperation = UInt32(kCCEncrypt)
12              let algoritm:  CCAlgorithm = UInt32(kCCAlgorithmAES128)
13              let options:   CCOptions   = UInt32(options)
14
15
```

Found in the file **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
45
46              let keyLength              = size_t(kCCKeySizeAES128)
47              let operation: CCOperation = UInt32(kCCDecrypt)
48              let algoritm:  CCAlgorithm = UInt32(kCCAlgorithmAES128)
49              let options:   CCOptions   = UInt32(options)
50
51              var numBytesEncrypted :size_t = 0
```

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 **iGoat-Swift/Source/Extensions/Data+Extension.swift**

```
18                  ivData.withUnsafeBytes {ivBytes in
19                      keyData.withUnsafeBytes {keyBytes in
20                          CCCrypt(CCOperation(operation),
21                                  CCAlgorithm(kCCAlgorithmAES),
22                                  options,
23                                  keyBytes, keyLength,
24                                  ivBytes,
```

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 **iGoat-Swift/Source/Exercises/InsecureLocalDataStorage/WebKitCache/WebkitCacheExerciseVC.swift**

```
117       // The iv is prefixed to the encrypted data
118       func aesCBCDecrypt(data:Data, keyData:Data) -> Data? {
119           let keyLength = keyData.count
120           let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
121           if (validKeyLengths.contains(keyLength) == false) {
122               print("Invalid key length")
123               return nil
```

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 **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
7               let data = self.data(using: String.Encoding.utf8),
8               let cryptData    = NSMutableData(length: Int((data.count)) + kCCBlockSizeAES128) {
9
10              let keyLength              = size_t(kCCKeySizeAES128)
11              let operation: CCOperation = UInt32(kCCEncrypt)
12              let algoritm:  CCAlgorithm = UInt32(kCCAlgorithmAES128)
13              let options:   CCOptions   = UInt32(options)
```

Found in the file **iGoat-Swift/Source/Extensions/NSString+Extension.swift**

```
43              let data = NSData(base64Encoded: self, options: .ignoreUnknownCharacters),
44              let cryptData    = NSMutableData(length: Int((data.length)) + kCCBlockSizeAES128) {
45
```

```
46              let keyLength             = size_t(kCCKeySizeAES128)
47              let operation: CCOperation = UInt32(kCCDecrypt)
48              let algoritm:  CCAlgorithm = UInt32(kCCAlgorithmAES128)
49              let options:   CCOptions   = UInt32(options)
```

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

Local network address

Found in the file **iGoat-Swift/Source/Exercises/Authentication/Remote/RemoteAuthenticationExerciseVC.swift**

```
3  enum RAConstants {
4      enum EndPoints {
5          static func loginUser(name: String, password: String) -> String {
6              return "http://localhost:8080/igoat/token?username=\(name)&password=\(password)"
7          }
8      }
9  }
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/Source/Exercises/Broken Cryptography/CryptoChallenge/CryptoChallengeVC.swift**

```
20              let postStr = postArray.reduce("") { $0 + "&" + $1 }
21              let postData = postStr.data(using: .utf8)
22
23              let url = URL(string: "http://localhost:8081/checkout/")!
24              var urlRequest = URLRequest(url: url)
```

```
25              urlRequest.httpMethod = "POST"
26              urlRequest.httpBody = postData
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/Source/Exercises/Key Management/Key Storage Server Side/KeyStorageServerSideVC.swift**

```
3
4   enum KSSConstants {
5       enum EndPoints {
6           static let cryptoKey = "http://localhost:8081/cryptoKey.php"
7       }
8       static let secretMessage = "SecretPass"
9   }
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/Source/Exercises/Server Communication/ServerCommunicationExerciseVC.swift**

```
14      @IBOutlet weak var lastNameField: UITextField!
15      @IBOutlet weak var ssnField: UITextField!
16
17      let serverCommunicationUserUrl = "http://localhost:8080/igoat/user"
18
19      @IBAction func submit(_ sender: Any) {
20
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/ThirdParty/CouchBase/CouchbaseLite.framework/Headers/CBLReplication.h**

```
101  @property (nonatomic, strong, nullable) id<CBLAuthenticator> authenticator;
102
103  /** The credential (generally username+password) to use to authenticate to the remote database.
104      This can either come from the URL itself (if it's of the form "http://user:pass@example.com")
105      or be stored in the NSURLCredentialStorage, which is a wrapper around the Keychain. */
106  @property (nonatomic, strong, nullable) NSURLCredential* credential;
107
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/Source/Exercises/Server Communication/ServerCommunicationExerciseVC.swift**

```
69   // and 8080 (non-SSL). To secure the POST to the /igoat/user endpoint, change
70   // the SERVERCOMMUNICATION_USER_URL constant at the top of the file to...
71   //
72   // "https://localhost:8443/igoat/user"
73   //
74   // The URLSession class will automatically wrap the connection in an SSL
75   // channel and the communication between client and server will be secure.
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company

Remediation

Delete links to local resources from the release version of your application.

network. These resources may be attacked by malicious parties.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---

Local network address

Found in the file **iGoat-Swift/Source/Exercises/InsecureLocalDataStorage/WebKitCache/WebkitCacheExerciseVC.swift**

```
24          @IBOutlet weak var textField: UITextField!
25          @IBOutlet weak var responseLabel: UILabel!
26
27          let demoEndpoint = "http://localhost:8081/webkit.php"
28
29
30          override func viewDidLoad() {
```

Vulnerability description

The application includes links to local addresses of applications that probably relate to the development infrastructure and indicate the internal structure of of the developer's company network. These resources may be attacked by malicious parties.

Remediation

Delete links to local resources from the release version of your application.

Links

https://cwe.mitre.org/data/definitions/359.html https://cwe.mitre.org/data/definitions/215.html

---