ROCKET.CHAT

A modern iOS application that exposes your account token and runs arbitrary commands.

Discovered by HADESS

7 Sep 2023

HADESS

WWW.HADESS.IO

Executive Summary

Exposed API Key in GET URL (/api/v1/users.info?userId):

1. The identified security concern relates to an exposed API key within the GET URL of the /api/v1/users.info endpoint, specifically when requesting user information by providing a userId parameter. This vulnerability could potentially lead to unauthorized access to user data or unauthorized API usage.
2. Immediate remediation steps include:
   - **API Key Protection:** Implement secure API key handling practices to ensure they are not exposed in URLs.
   - **Authorization Controls:** Review and enhance user authorization mechanisms to prevent unauthorized access.
   - **Access Logging:** Implement access logging and monitoring to detect and respond to suspicious activities.
3. Failure to address this issue may result in data breaches, unauthorized access, or misuse of the API.

RTLO Character Injection in Chat:

1. The security concern pertains to the presence of Right-to-Left Override (RTLO) character injection within chat messages. This technique can be exploited to deceive users, potentially leading to phishing attacks or the dissemination of malicious content.
2. Recommended actions to mitigate this threat include:
   - **Input Validation:** Implement input validation and filtering to block or neutralize RTLO characters.
   - **User Education:** Educate users about recognizing and avoiding suspicious content in chat messages.
   - **Content Filtering:** Deploy content filtering mechanisms to detect and prevent the injection of malicious characters.
3. Neglecting to address RTLO character injection can undermine trust among users and expose them to security risks.

01

Advisory

Abstract

Rocket.Chat is an open-source team collaboration platform and messaging system designed to facilitate communication and collaboration within organizations. It provides a flexible and customizable environment for teams to communicate through text, audio, and video, making it suitable for a wide range of use cases, from small businesses to large enterprises. Here are some key features and aspects of Rocket.Chat:

1. **Real-Time Messaging:** Rocket.Chat offers real-time messaging capabilities, allowing users to engage in instant one-on-one or group chats. This feature fosters quick and efficient communication among team members.
2. **Channels and Private Groups:** Users can create public channels for open discussions and private groups for more confidential conversations. This flexibility enables users to organize their conversations based on topics or project teams.
3. **Multi-Platform Support:** Rocket.Chat is available as a web application, desktop application, and mobile app for various platforms, including Windows, macOS, Linux, Android, and iOS. This ensures accessibility from virtually anywhere.
4. **File Sharing:** Users can share files, images, documents, and other media within Rocket.Chat, making it easy to collaborate and exchange information seamlessly.
5. **Integration Hub:** One of the standout features of Rocket.Chat is its extensive integration capabilities. It supports numerous third-party integrations, including popular productivity tools, project management software, and customer relationship management (CRM) systems. This allows organizations to consolidate their workflows and centralize communication.
6. **Customization:** Rocket.Chat is highly customizable, allowing organizations to tailor the platform to their specific needs. It offers the ability to create custom themes, plugins, and integrations.
7. **Security:** Security is a top priority for Rocket.Chat. It provides end-to-end encryption for one-on-one chats and offers various security features to protect sensitive information.
8. **Open Source:** Being open source means that Rocket.Chat's source code is freely available, and organizations can modify it to suit their unique requirements. This also encourages a vibrant community of developers and contributors.

9. Enterprise Features: For larger organizations, Rocket.Chat offers enterprise-grade features such as user management, scalability, and advanced security options.
10. **Community and Support:** Rocket.Chat has an active and supportive community, providing resources, documentation, and forums for users and administrators. There are also options for professional support and managed hosting services.

02

Technical Analysis

Technical Analysis

Account takeover scenarios pose significant threats to the security and privacy of users of any application, particularly those that handle sensitive communication and data, such as the Rocket.Chat iOS application. In this comprehensive analysis, we'll delve into a hypothetical account takeover scenario within the Rocket.Chat iOS app. This scenario involves the inadvertent exposure of an authentication key in a GET request and the subsequent unauthorized access through the **/api/v1/users.info?userId** endpoint. To illustrate the scenario, we'll provide examples of both non-compliant and compliant iOS source code.

Account Takeover Scenario Overview:

1. **Authentication Key in GET Request:**
2. The security issue begins with the unfortunate inclusion of sensitive authentication keys within the URL query string of a GET request. Normally, authentication information, such as API keys or tokens, should never be exposed in this manner. Query parameters in GET requests can be logged in various locations, including web server logs, browser history, and possibly third-party analytics services.
3. **Exploitation in /api/v1/users.info?userId:**
4. Rocket.Chat's **/api/v1/users.info** endpoint allows users to retrieve user information by providing a **userId** parameter. In our scenario, an attacker discovers that the authentication key is exposed in the GET request, likely due to inadvertent exposure, poor security practices, or vulnerabilities in the application.
5. **Unauthorized Authorization:**
6. Equipped with the exposed authentication key, the attacker can craft their own GET requests to the **/api/v1/users.info** endpoint, replacing the legitimate **userId** parameter with the target victim's **userId**. Since the authentication key is included, the system may erroneously authorize these requests, believing they originate from a legitimate source.
7. **Account Takeover and Unauthorized Access:**
8. With unauthorized access to a victim's account information through the **/api/v1/users.info** endpoint, the attacker can potentially gain access to sensitive user data, including personal details, messages, and other confidential information. They may also manipulate the victim's account settings or engage in malicious activities on their behalf, posing severe risks to user privacy and security.

Non-Compliant iOS Source Code Example:
Below is an example of non-compliant iOS source code that demonstrates the vulnerability described in the scenario:

```
let authKey = "YOUR_AUTHENTICATION_KEY"
let userId = "VICTIM_USER_ID"

// Vulnerable GET request with authentication key in the URL
let urlString = "https://your-rocket-chat-api.com/api/v1/users.info?
userId=\(userId)&authKey=\(authKey)"

if let url = URL(string: urlString) {
      let task = URLSession.shared.dataTask(with: url) { data, response,
error in
        // Handle the response here
    }
    task.resume()
}
```

Compliant iOS Source Code Example:
To address this security vulnerability, the following iOS source code example demonstrates best practices by using headers for authentication and secure handling of sensitive data:

```
let authKey = "YOUR_AUTHENTICATION_KEY"
let userId = "VICTIM_USER_ID"

let apiUrl = "https://your-rocket-chat-api.com/api/v1/users.info"
if let url = URL(string: apiUrl) {
    var request = URLRequest(url: url)
    request.httpMethod = "GET"

    // Set headers for authentication
              request.setValue("Bearer \(authKey)", forHTTPHeaderField:
"Authorization")

     let task = URLSession.shared.dataTask(with: request) { data, response,
error in
        // Handle the response here
    }
    task.resume()
}
```

Mitigation and Prevention:
To prevent such an account takeover scenario in the Rocket.Chat iOS application, or any similar application, the following measures should be taken:

1. **API Key Protection:** Avoid including sensitive information like authentication keys in GET requests. Instead, use appropriate methods like HTTP headers for authentication.
2. **Authorization Controls:** Implement robust authorization checks on each API request to ensure that only authorized users can access specific endpoints and resources.
3. **Security Audits:** Regularly audit and review the application's security practices, including API security, to identify and rectify vulnerabilities before they can be exploited.
4. **User Education:** Educate users about security best practices, such as choosing strong passwords and enabling two-factor authentication to add an additional layer of account protection.
5. **Monitoring and Alerts:** Set up monitoring and alerting systems to detect suspicious activities and unauthorized access attempts, allowing for rapid response and mitigation.

Right-to-Left Override (RTLO) character injection is a cunning text manipulation technique that poses a substantial security threat in various applications, including iOS applications like Rocket.Chat. In this in-depth article, we will delve into the intricacies of RTLO character injection. We'll explore its potential risks, provide five example payloads to illustrate its danger, and discuss both non-compliant and compliant iOS source code. Furthermore, we will examine two methods by which malicious actors could potentially exploit RTLO character injection to run arbitrary code.

Understanding RTLO Character Injection:
RTLO character injection is an ingenious method wherein specially crafted Unicode characters are utilized to manipulate the text's directionality. The most notorious character used for this purpose is the "U+202E: Right-To-Left Override" character. When this character is inserted into text, it can reverse the display order of characters following it, leading to deceptive and malicious content.
Potential Risks:
RTLO character injection carries significant risks:

1. **Phishing Attacks:** Attackers can use RTLO characters to disguise malicious URLs, tricking users into clicking on harmful links.
2. **Social Engineering:** By manipulating text directionality, attackers can alter the appearance of names or messages, leading users to trust malicious actors.
3. **Malware Distribution:** Concealing malicious file extensions in filenames can deceive users into downloading and executing harmful files.
4. **Bypassing Security Scanners:** RTLO characters can evade security scanners and filters, allowing malicious content to bypass detection.

Mitigation Strategies:
To protect iOS applications like Rocket.Chat from RTLO character injection, consider the following strategies:

1. **Input Validation:** Implement input validation to filter and neutralize RTLO characters in user-generated content.
2. **Content Filtering:** Deploy content filtering mechanisms to detect and prevent the injection of malicious characters.

- **User Education:** Educate users about recognizing and avoiding suspicious content that may contain RTLO characters.
- **Character Whitelisting:** Allow only known-safe characters in user-generated content and reject anything that deviates from the whitelist.
- **Security Audits:** Regularly audit and review the application's security practices, including API security, to identify and fix vulnerabilities related to RTLO character injection.

Example Payloads:
Here are five example payloads that demonstrate how RTLO characters can be used maliciously:

- **Phishing URL: www.evil.com/***qwertyuiop.com*
- **Social Engineering:** *usernamesofinsider*
- **Misleading File Name: important_document.pdf**
- **Security Confusion:** *fromlegitimation*.**exe**
- **Invisible Payload: malicious-script.**txt
-

Non-Compliant iOS Source Code:
Below is an example of non-compliant iOS source code that does not address RTLO character injection:

```
let message = "Click here to visit our website: www.evil.com/qwertyuiop.com"
label.text = message
```

Compliant iOS Source Code:
To mitigate the risks associated with RTLO character injection, use input validation and character whitelisting in your iOS source code, like this:

```
let message = "Click here to visit our website: www.evil.com/qwertyuiop.com"
let sanitizedMessage = sanitizeRTLOCharacters(message)
label.text = sanitizedMessage

func sanitizeRTLOCharacters(_ input: String) -> String {
    // Implement a function to remove or neutralize RTLO characters
    // Example: Remove U+202E character
     let sanitizedInput = input.replacingOccurrences(of: "\u{202E}", with:
"")
    return sanitizedInput
}
```

Exploiting RTLO Character Injection to Run Arbitrary Code:
Malicious actors may attempt to leverage RTLO character injection to run arbitrary code within an application. Here are two potential methods:

1. URL Redirection to a Malicious JavaScript File:
2. By injecting RTLO characters into a URL, an attacker could craft a deceptive link that appears benign but redirects the user to a malicious JavaScript file. Once executed, this JavaScript can perform arbitrary actions on the user's device, such as data theft or unauthorized access.
3. Example Payload:
4. https://www.example.com/malicious.js%e2%80%ae%e2%80%ae%e2%80%ae%e2%80%ae%e2%80%ae%e2%80%ae%e2%80%ae
5. Social Engineering for Code Execution:
6. By altering the appearance of a message or notification through RTLO character injection, an attacker may deceive the user into executing a malicious action. For instance, the attacker could manipulate a message to look like a system update prompt, tricking the user into executing arbitrary code.

03

Conclusion

In this discussion, we've explored two critical security vulnerabilities: exposing authentication tokens in GET requests with shared functionality and running arbitrary code using Right-to-Left Override (RTLO) character injection. Both of these vulnerabilities can have significant consequences for the security and privacy of applications and their users.

Exposing Auth Token in GET Requests with Share Functionality:
Exposing authentication tokens in GET requests, especially when shared through functionality like sharing links or URLs, is a substantial security risk. It can lead to unauthorized access, data breaches, and misuse of sensitive information. To mitigate this risk:

- Implement secure practices for handling authentication tokens, such as using HTTP headers or secure cookies instead of including them in URLs.
- Apply proper access controls and authorization checks to ensure that sensitive operations are only performed by authorized users.
- Regularly audit and review the application's security measures, including token handling and sharing functionality, to identify and address vulnerabilities.

Running Arbitrary Code with RTLO Character Injection:
RTLO character injection is a sophisticated technique that manipulates text directionality to deceive users and potentially execute malicious actions. It poses risks such as phishing attacks, social engineering, and code execution. To defend against RTLO character injection:

- Implement input validation and content filtering to neutralize or remove RTLO characters from user-generated content.
- Educate users about recognizing suspicious content and links that may contain RTLO characters.
- Stay updated on security best practices and conduct regular security audits to identify and mitigate vulnerabilities.

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

Hadess Cyber Security Solutions is a renowned group of cyber security experts and white hat hackers. With a track record of discovering and responsibly reporting vulnerabilities to prominent companies such as Google, Apple, and Twitter, Hadess has earned a reputation for excellence in the field of cyber security.