Threat Modeling Report

Project: halodoc-ios-master

Property	Value
Project Name	halodoc-ios-master
Analysis Date	2025-10-18T21:45:23.124275
Methodology	STRIDE
Total Findings	4
Risk Level	HIGH

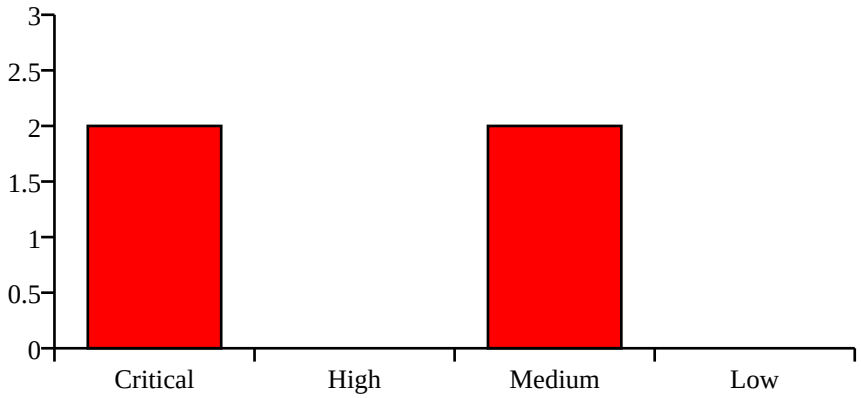
Executive Summary

This security assessment of **halodoc-ios-master** identified **4** potential security threats using the STRIDE methodology. The overall risk level is assessed as **HIGH**.

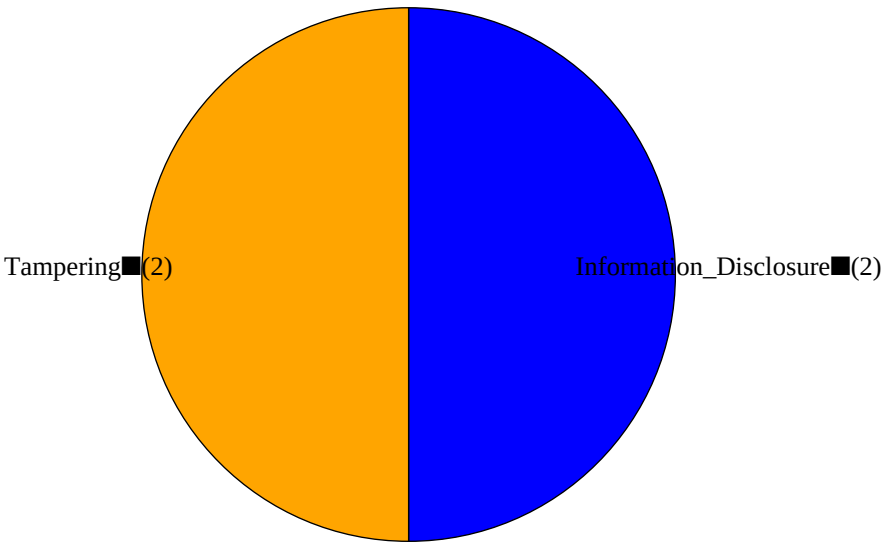
Key Findings:
• Critical vulnerabilities: 2
• High-severity issues: 0
• Medium-priority concerns: 2
• Low-priority items: 0

Immediate attention is required for all critical and high-severity vulnerabilities to prevent potential security breaches.

Threat Severity Distribution



STRIDE Category Distribution



STRIDE Methodology Overview

STRIDE is a threat modeling methodology developed by Microsoft that categorizes security threats into six main areas:

S - Spoofing Identity: Impersonating someone or something else to gain unauthorized access
T - Tampering with Data: Malicious modification of data or code
R - Repudiation: Users denying they performed an action without the system being able to prove otherwise
I - Information Disclosure: Exposure of information to individuals who shouldn't have access
D - Denial of Service: Attacks that deny or degrade service for legitimate users
E - Elevation of Privilege: A user gains capabilities without proper authorization

Each identified threat is categorized into one of these areas and assessed for severity and impact.

Project Architecture Analysis

Code Analysis Summary:
• Files analyzed: 3
• Programming languages: JavaScript
• Threat detection patterns: STRIDE-based security analysis
• Analysis depth: Source code static analysis with context awareness

Detailed Security Findings

Finding #1: Code injection via eval

Property	Details
Severity	Critical
STRIDE Category	Tampering
CWE ID	CWE-95
Confidence Score	0.90
File Location	halodoc-ios-master/crash_monkey_result/result_000/result_view.js:42
Attack Vector	Data modification, code injection, integrity violations

Description:

Code injection via eval detected in JavaScript code

Code Evidence:

```
index: i }); >>> return eval(log.message); }); return img.src =
log.screen_image + '.png';
```

Proof of Concept:

Steps to Reproduce:
1. Locate the vulnerable eval() function in the source code
2. Identify user input that reaches the eval() function
3. Craft malicious JavaScript payload
4. Execute payload through the vulnerable input vector
5. Observe code execution in the application context

Impact: Remote code execution, full application compromise

Remediation:

Avoid dynamic code execution, use safe alternatives

Business Impact:

Data corruption, financial loss, operational disruption, legal liability

Finding #2: Code injection via eval

Property	Details
Severity	Critical
STRIDE Category	Tampering
CWE ID	CWE-95
Confidence Score	0.90
File Location	halodoc-ios-master/crash_monkey_result/result_001/result_view.js:42
Attack Vector	Data modification, code injection, integrity violations

Description:

Code injection via eval detected in JavaScript code

Code Evidence:

```
index: i }); >>> return eval(log.message); }); return img.src =
log.screen_image + '.png';
```

Proof of Concept:

Steps to Reproduce:
1. Locate the vulnerable eval() function in the source code
2. Identify user input that reaches the eval() function
3. Craft malicious JavaScript payload
4. Execute payload through the vulnerable input vector
5. Observe code execution in the application context

Impact: Remote code execution, full application compromise

Remediation:

Avoid dynamic code execution, use safe alternatives

Business Impact:

Data corruption, financial loss, operational disruption, legal liability

Finding #3: Information disclosure in errors

Property	Details
Severity	Medium
STRIDE Category	Information_Disclosure
CWE ID	CWE-209
Confidence Score	0.70
File Location	halodoc-ios-master/crash_monkey_result/UIAutoMonkey.js:165
Attack Vector	Data leakage, privacy violations, sensitive exposure

Description:

Information disclosure in errors detected in JavaScript code

Code Evidence:

```
if (!event) { UIALogger.logMessage("Attempted to " + name) >>> throw new
Error("Attempted to fire an undefined event '" + name + "'!") }
event.apply(this);
```

Proof of Concept:

Steps to Reproduce:
1. Identify information exposure point
2. Analyze data access controls
3. Attempt unauthorized data access
4. Extract sensitive information
5. Verify information disclosure

Impact: Data breach, privacy violation

Remediation:

Implement proper error handling without information disclosure

Business Impact:

Minor data exposure, potential privacy concerns

Finding #4: Insecure HTTP usage

Property	Details
Severity	Medium

STRIDE Category	Information_Disclosure
CWE ID	CWE-319
Confidence Score	0.70
File Location	halodoc-ios-master/crash_monkey_result/UIAutoMonkey.js:2
Attack Vector	Data leakage, privacy violations, sensitive exposure

Description:

Insecure HTTP usage detected in JavaScript code

Code Evidence:

```
>>> // Copyright (c) 2013 Jonathan Penn (http://cocoamanifest.net/) //
Permission is hereby granted, free of charge, to any person obtaining a copy
```

Proof of Concept:

Steps to Reproduce:
1. Identify information exposure point
2. Analyze data access controls
3. Attempt unauthorized data access
4. Extract sensitive information
5. Verify information disclosure

Impact: Data breach, privacy violation

Remediation:

Use HTTPS/TLS for all communications

Business Impact:

Minor data exposure, potential privacy concerns

Remediation Summary

Remediation Priority Matrix

■ IMMEDIATE (0-7 days) - Critical Issues: 2
Critical vulnerabilities pose immediate risk to business operations and must be addressed urgently. Recommended actions: Emergency patches, temporary mitigations, incident response preparation.

■ HIGH PRIORITY (1-4 weeks) - High Severity: 0
High-severity issues should be addressed in the next sprint cycle. Recommended actions: Security patches, code reviews, testing validation.

■ MEDIUM PRIORITY (1-3 months) - Medium Severity: 2
Medium-severity issues can be addressed in regular development cycles. Recommended actions: Security improvements, best practice implementation, monitoring enhancement.

Implementation Guidelines:
• Establish security champion within development team
• Implement security testing in CI/CD pipeline
• Conduct regular security code reviews
• Provide security training for developers
• Monitor for new vulnerabilities and threat intelligence
• Regular penetration testing and security assessments