PENETRATION TEST REPORT
EXAMPLE INC

Septemper 6, 2018 AmeerAssadi.com
ameerassadime@gmail.com

Table of Contents

Executive Summary 1

Findings Overview 2

Findings 3

SQL Injection on app.example.com 3

Privilege Escalation via Insecure Direct Object Reference 4

Stored Cross-Site Scripting 5

Cross-Site Request Forgery on Posting Comments 6

Executive Summary

Ameer Assadi conducted a 1 week, 20 hour penetration test of Example Inc., with the goal of discovering vulnerabilities in the following systems:

Web: `*.example.com` - Example Inc. Customer-Facing Website

Web: `*.example.net` - Example Inc. Corporate Website

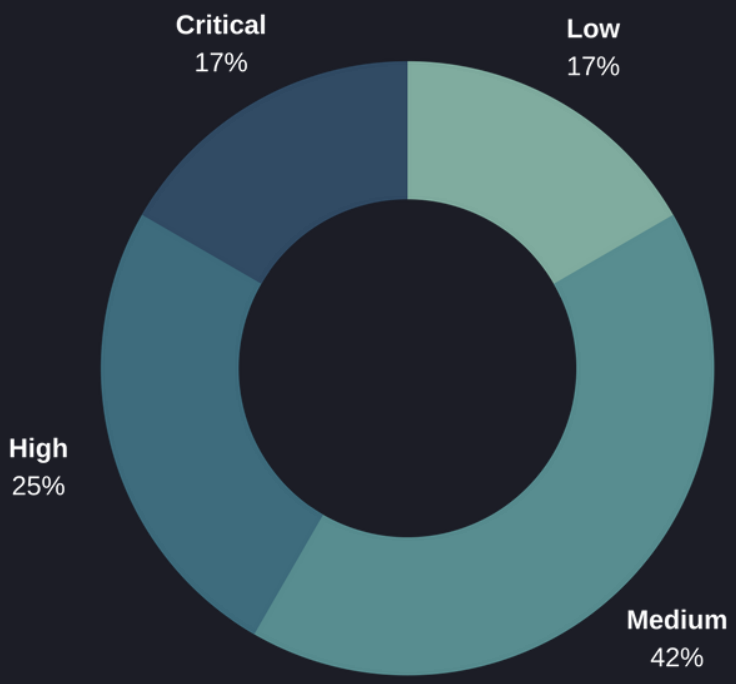
Mobile: Example Inc. iOS application

Priority was given to `app.example.com` and the Example Inc. iOS application, which provide an interface for Example Inc. customers to manage their accounts. As the protection of Personally Identifiable Information (PII) is a priority for Example Inc., the exposure of information such as social security numbers, payment information, and other user data was deemed especially critical.

Ameer Assadi was given access to user accounts with standard permission, in order to simulate a real attack scenario.

Findings Overview

Over the course of testing, Ameer Assadi identified 2 critical severity flaws, 3 high severity flaws, 5 medium severity flaws, and 2 low severity flaws:



The following issues were identified:

Description	Severity	Page
SQL Injection on `app.example.com`	Critical	3
Privilege Escalation via Insecure Direct Object Reference	High	4
Stored Cross-Site Scripting on User Profile	High	5
Cross-Site Request Forgery on Posting Comments	Medium	6

Findings

SQL Injection on app.example.com

Ameer Assadi identified a critical SQL injection vulnerability on `app.example.com`. By making a request to `/order/complete` with a malicious payload in the `productId` parameter, it is possible to extract data from the production database of Example Inc. As a proof of concept, Ameer Assadi was able to extract the hashed password of another account created specifically for testing.

Reproduction Steps

1. Visit the `Order Completion` page at `https://app.example.com/order/complete`.
2. Intercept the request in a proxy such as Burp Suite.
3. Modify the `productId` to the value `1408' AND 'a'='a`.
4. Observe that the request succeeds.
5. Modify the `productId` to the value `1408' AND 'a'='b`.
6. Observe that the request fails, as the expression `'a'=b` evaluates to false.

Impact

This vulnerability allows an attacker to extract all data in the `app.example.com` production database. This includes user information such as social security numbers, hashed passwords, and credit card numbers.

Suggested Remediation

Escape all special characters in SQL queries.

Privilege Escalation viaInsecure Direct Object Reference

Ameer Assadi identified a privilege escalation vulnerability on `app.example.com`. The `/team/:id/makeAdmin` endpoint allows a user to add another user as an administrator of their team. Passing the id of another team allows an attacker to add themselves as an administrator to the team of another user.

Reproduction Steps

1. Visit the `Team Overview` at `https://example.com/team/overview`.
2. Obtain the team id of another user (These are incremented integers, so an attacker could enumerate other team ids).
3. Intercept the request to `/team/:id/makeAdmin` to add an administrator to the current team.
4. Modify the id in the url to the other user's id.
5. Observe that the testing will be added as an administrator of the other user's team.

Impact

This vulnerability allows an attacker to add themselves as an administrator of any user's team, exposing sensitive information associated with the team and allowing the attacker to make modifications to the team.

Suggested Remediation

Ensure a user may only invite other users as administrators to teams of which they are an owner.

Stored Cross-Site Scripting

Ameer Assadi identified a stored cross-site scripting vulnerability on `app.example.com`. By setting their public biography to a malicious payload, an attacker can execute JavaScript on the browser of other users. This allows the attacker to steal cookies of other users, compromising their accounts.

Reproduction Steps

1. Visit the `Profile` page at `https://example.com/profile`.
2. Update the user biography to `<img src=x onerror=alert(document.domain)>`
3. Refresh the profile page.
4. An alert will execute, demonstrating the vulnerability.

Impact

By executing arbitrary JavaScript on the browser of other users, an attacker can completely compromise the accounts of other users.

Suggested Remediation

Escape special characters in user-provided input.

Cross-Site Request Forgery on Posting Comments

Ameer Assadi identified a cross-site request forgery vulnerability on posting comments in team threads. As a result, this allows an attacker to post comments from the accounts of other users.

Reproduction Steps

1. Visit the `Team Overview` at `https://example.com/team/overview`.
2. Intercept the request to post a comment in a proxy such as Burp Suite.
3. Observe that no CSRF validation is present. To confirm, make an HTML form such as the following:

```
<form action="https://example.com/team/comments/new" method="POST">
  <input type="hidden" name="content" value="Demo comment" />
  <input type="submit" value="Submit request" />
</form>
<script>document.forms[0].submit();</script>
```

4. Upon visiting this page, a comment will be posted from the logged-in user's account.

Impact

Exploiting this vulnerability, an attacker is able to impersonate other users by posting comments from their accounts.

Suggested Remediation

Add CSRF validation to all forms to prevent this type of attack. For further reference, see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.