



Web Application Penetration Test Report

TrustMi

March 27, 2025

Table of Contents

Overview

• Track Changes	2
• Executive Summary	3
• Scope	4
• Conclusions	5
• Vulnerabilities Index	7
• Appendix A Work Scheme	23

Findings - Infrastructure

• Lack of software updates	8
• IAM Users with Overlooked Permissions to Create Access Keys for Admin Accounts	10
• Username Enumeration via WPScan	12
• Exposure of Slack Bot Token and Slack Webhooks in AWS Environment Variables	15
• Outdated OpenSSH Utilized in the Internal Network	18
• Username Enumeration via Login Page Error Message	21

Track Changes

#	Name	Date	Action
1.	Roe Dikes, Haili Sambrano	February 18, 2025	Test Operators
2.	Roe Dikes, Haili Sambrano	February 26, 2025	Report Writers
3.	Meital Guzi	March 27, 2025	Report Reviewer

Executive Summary

General

TrustMi hired 2BSecure to perform penetration tests on the Trustmi Portals. The purpose of the tests was to examine the application's immunity and measure its security level, while considering both technology and business logic. Penetration tests aspire to reveal most of the existing security risks, such as confidential information exposure, data corruption, interference with availability, or damage to servers, applications, organizations, clients, etc.

System Description

TrustMi platform was designed to protect the end-to-end business payment process from all threats, internal and external, across the entire payment flow making sure payments go through seamlessly.

Activity Description

During February 2025, 2BSecure performed a penetration test to Trustmi Portals applications and Trustmi infrastructure. The tests were performed manually and automatically in order to examine all the application's requests, actions, and procedures, aspiring to expose all of the security flaws in the system.

The audit took place at 2BSecure's labs on a production environment, and was implemented with a highly sensitive approach to avoid interfering with the routine activity of the system.

Scope

The table below lists the areas designated for this penetration test, based on the parameters defined in the quote.

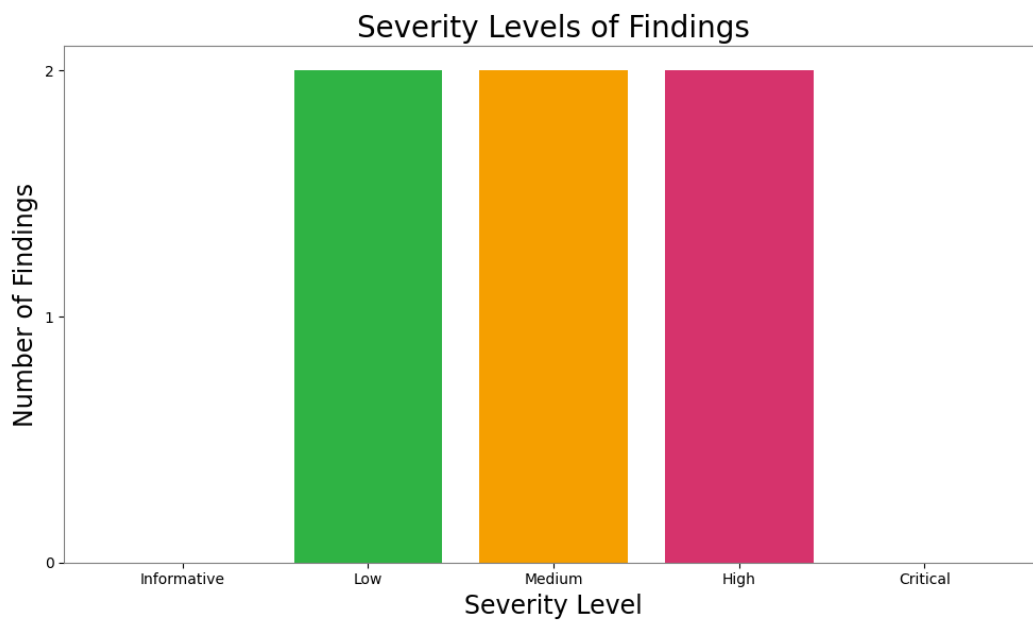
#	Target Name	Value
1.	External Addresses	files.trustmi.ai
2.	External Addresses	workato.trustmi.ai
3.	Dev	10.116.0.0/16
4.	Networking	10.115.0.0/22
5.	Shared-Services	10.115.12.0/22
6.	Trustmi-MGMT-Dev	10.12.0.0/16
7.	TrustmiAdmin	10.0.0.0/16, 10.10.0.0/16, 10.11.0.0/16, 192.168.0.0/16
8.	AWS User Account	PenTestUser2Bsecure
9.	External Addresses	trustmi.ai

Conclusions

The audit found that the Trustmi Portals application is exposed to numerous vulnerabilities. The findings were well-analyzed and aggregated based on two factors, Risk and Probability. Together, these factors produce the severity level of the vulnerability, based on the global OWASP convention.

		Overall Risk Severity		
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Informative	Low	Medium
		LOW	MEDIUM	HIGH
		Likelihood		

The final security level of the entire Trustmi Portals application is based on all the vulnerabilities found and the severity level of each one. The following chart presents the distribution of the severity levels of the vulnerabilities found:



In conclusion, the Trustmi Portals application severity level is **High**.

The vulnerabilities found might present a risk to the organization, the application, and its users. It is recommended to study the findings presented in this document and correct them accordingly.

Vulnerabilities Index

The following table presents the vulnerabilities identified and the severity level of each one. Detailed explanations and recommendations can be found in the sections below.

#	Title	Risk Level
1.	Lack of software updates	High
2.	IAM Users with Overlooked Permissions to Create Access Keys for Admin Accounts	High
3.	Username Enumeration via WPScan	Medium
4.	Exposure of Slack Bot Token and Slack Webhooks in AWS Environment Variables	Medium
5.	Outdated OpenSSH Utilized in the Internal Network	Low
6.	Username Enumeration via Login Page Error Message	Low

Application Vulnerability Description

1. Lack of software updates

High Severity Likelihood: Medium, Impact: High

Threat Description

During the assessment, an outdated SSH service version that contains a known vulnerability was found to be running on a target. This outdated configuration can potentially expose the system to attacks and unauthorized access.

Impact Description

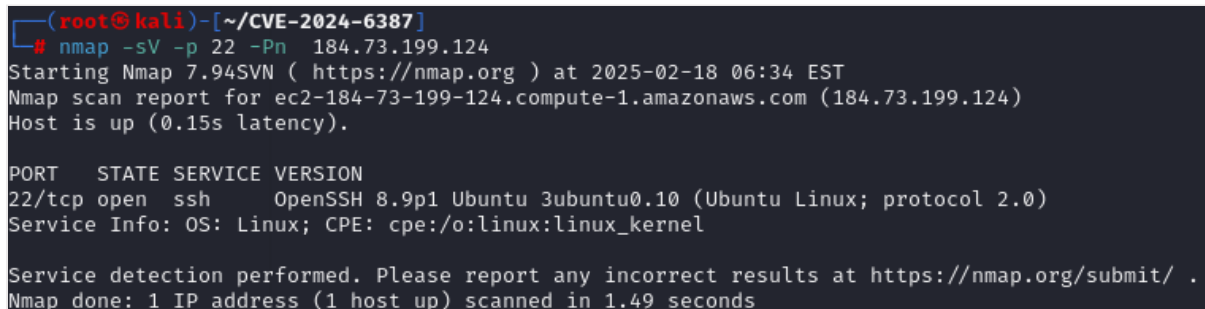
If exploited, the vulnerability might enable an attacker to bypass authentication mechanisms and gain unauthorized access to the system.

Likelihood Description

Exploitation is moderately likely, especially in environments in which the SSH service is directly exposed to the internet and patch management is lax. Automated tools that scan for outdated SSH versions further increase the risk of this vulnerability being discovered and exploited by adversaries.

Attack description

The following photo shows the exposure of the vulnerable SSH version on the target:



```
(root@kali)~[~/CVE-2024-6387]
# nmap -sV -p 22 -Pn 184.73.199.124
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-18 06:34 EST
Nmap scan report for ec2-184-73-199-124.compute-1.amazonaws.com (184.73.199.124)
Host is up (0.15s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.49 seconds
```

This version is vulnerable, and has exploits that are publicly available.

The following photo shows that the scanner of CVE-2024-6387 identify the target as vulnerable:

```
(root@kali)-[~/CVE-2024-6387]
# python3 CVE-2024-6387.py scan -T 184.73.199.124 -p 22

.aMMb dMMb dMMMMMP dMMb .dMMb .dMMb dMP dMP
dMP"dMP dMP.dMP dMP dMP dMP" VP dMP" VP dMP dMP
dMP dMP dMMMMP" dMMMP dMP dMP VMMb VMMb dMMMMMP
dMP,aMP dMP dMP dMP dMP dP .dMP dP .dMP dMP dMP
VMMMP" dMP dMMMMMP dMP dMP VMMMP" VMMMP" dMP dMP

De ReggreSSHion CVE-2024-6387 Vulnerability Checker / Exploiter
2.0 - Optimized by @Kz
File System

[+] Servers likely vulnerable: 1
[+] Server at 184.73.199.124 (N/A):22 (running SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.10)

[+] Servers not vulnerable: 0

[+] Servers likely not vulnerable (possible LoginGraceTime remediation): 0
^ Servers with unknown SSH version: 0

Summary:
[+] Total scanned hosts: 1
[+] Total vulnerable hosts: 1
```

Countermeasures

- Update the SSH service to the latest stable version.
- Consider additional measures such as network segmentation and strict access controls.

2. IAM Users with Overlooked Permissions to Create Access Keys for Admin Accounts

High Severity

Likelihood: Medium, Impact: High

Threat Description

Certain legacy IAM users retain permissions to create access keys for a higher-privileged, effectively administrative user. Although the organization has transitioned to using Single Sign-On (SSO) for day-to-day access, these legacy IAM users still exist in the AWS environment, and their permissions remain in place.

Impact Description

An attacker or malicious insider with access to one of these legacy IAM accounts could exploit the permission to create an access key for the administrative user. By generating a new access key, the attacker could bypass SSO controls and gain unauthorized administrative access. This would potentially allow the attacker to modify configurations, exfiltrate sensitive data, or perform other high-impact actions within the AWS environment.

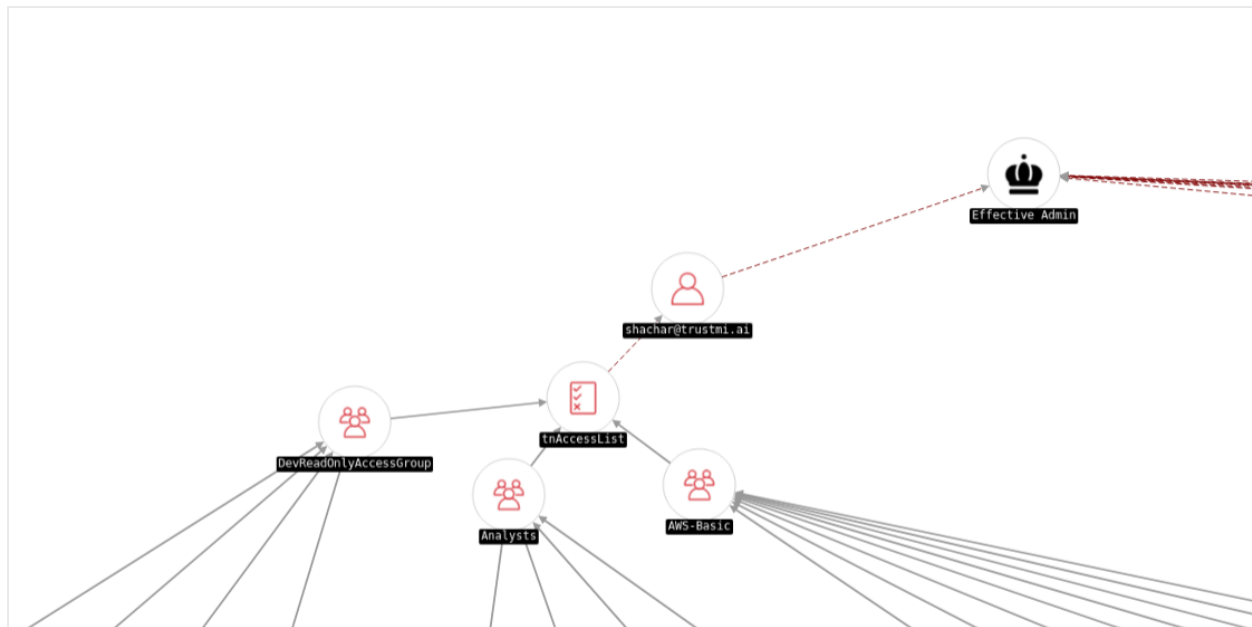
Likelihood Description

Although the legacy IAM users are not actively used due to the migration to SSO, their residual permissions remain unaddressed. This oversight provides an exploitable attack vector if these legacy credentials are compromised or if an insider misuses their access rights.

Attack description

An attacker or malicious insider with access to one of these legacy IAM accounts could exploit the permission to create an access key for the administrative user (Shachar). By generating a new access key, the attacker could gain unauthorized administrative access. This would potentially allow them to modify configurations, exfiltrate sensitive data, or perform other high-impact actions within the AWS environment.

The attack vector that can be utilized to gain unauthorized access can be seen in the following photo:



Groups with the tnAccess list permission can create an access key to Shacher's user and gain an admin access to the AWS environment.

Attack Path	Notes
<p>Step 1: Create an access key for the target user and authenticate as them using the API:</p> <pre>aws iam create-access-key \ --user-name shachar@trustmi.ai</pre>	

Countermeasures

- It is recommended to delete users that are no longer in use such as the Analysts and DevReadOnly who has access to the permissions to create access key for Shachar's user.
- Conduct a comprehensive review of IAM policies to ensure adherence to the principle of least privilege, removing any unnecessary or outdated permissions.

3. Username Enumeration via WPScan

Medium Severity Likelihood: High, Impact: Low

Threat Description

The audit found that usernames are disclosed on the marketing site. The system allows an attacker to obtain a list of the system's users by scanning the WP instance with WPScan.

Impact Description

The exposure of valid usernames enables attackers to more easily conduct brute force or credential stuffing attacks. Knowledge of legitimate user accounts can be combined with social engineering or phishing strategies to compromise additional security layers..

Likelihood Description

This vulnerability is highly likely to be exploited since WPScan and similar automated scanning tools are freely available and commonly used by attackers. Additionally, many WordPress installations retain default configurations that expose author information, making the attack both easy and effective.

Attack description

Attackers can automate the discovery of valid usernames by running WPScan or similar tools against the WordPress instance as can be seen in the following picture:

```
[+] URL: https://trustmi.ai/ [141.193.213.10]
[+] Started: Sun Feb 23 08:24:33 2025

Interesting Finding(s):

[+] Headers
| Interesting Entries:
| - x-powered-by: WP Engine
| - x-cacheable: bot
| - x-cache-group: bot
| - cf-cache-status: HIT
| - server: cloudflare
| - cf-ray: 9167850cffffdc224-TLV
| - alt-svc: h3=":443"; ma=86400
| Found By: Headers (Passive Detection)
| Confidence: 100%

[+] robots.txt found: https://trustmi.ai/robots.txt
| Found By: Robots Txt (Aggressive Detection)
| Confidence: 100%

Fingerprinting the version - Time: 00:00:49
[i] The WordPress version could not be detected.

[+] WordPress theme in use: bootscore
| Location: https://trustmi.ai/wp-content/themes/bootscore/
| Style URL: https://trustmi.ai/wp-content/themes/bootscore/style
| Style Name: Bootscore
| Style URI: https://bootscore.me/
| Description: Flexible Bootstrap WordPress starter-theme for dev
| Author: Bootscore
| Author URI: https://bootscore.me
| Found By: Css Style In Homepage (Passive Detection)
| Version: 6.0.4 (80% confidence)
| Found By: Style (Passive Detection)
| - https://trustmi.ai/wp-content/themes/bootscore/style.css?ver

[+] Enumerating Users (via Passive and Aggressive Methods)
Brute Forcing Author IDs - Time: 00:00:00

[i] User(s) Identified:

[+] adamdev
| Found By: Yoast Seo Author Sitemap (Aggressive Detection)
| - https://trustmi.ai/author-sitemap.xml
```


Once valid usernames are enumerated, adversaries may employ brute force, dictionary attacks, or credential stuffing to compromise accounts or use the information for targeted social engineering campaigns.

Countermeasures

- Disable or limit access to endpoints that expose author information. Use plugins or custom code to obfuscate usernames.
- Review and update WordPress security settings to minimize information leakage. consider using plugins such as "Stop User Enumeration"

4. Exposure of Slack Bot Token and Slack Webhooks in AWS Environment Variables

Medium Severity Likelihood: Low, Impact: High

Threat Description

Slack bot tokens and Slack webhook URLs are stored directly in the environment variables of AWS Lambda functions rather than being managed via a secure secrets manager. This misconfiguration allows users with certain AWS IAM permissions to access these sensitive credentials, increasing the risk of unauthorized use.

Impact Description

Malicious actors can use the bot to disseminate malicious content or phishing links, compromising the integrity of internal communications.

Likelihood Description

An attacker would first need to obtain an AWS user with sufficient permissions to read the data from the lambda function.

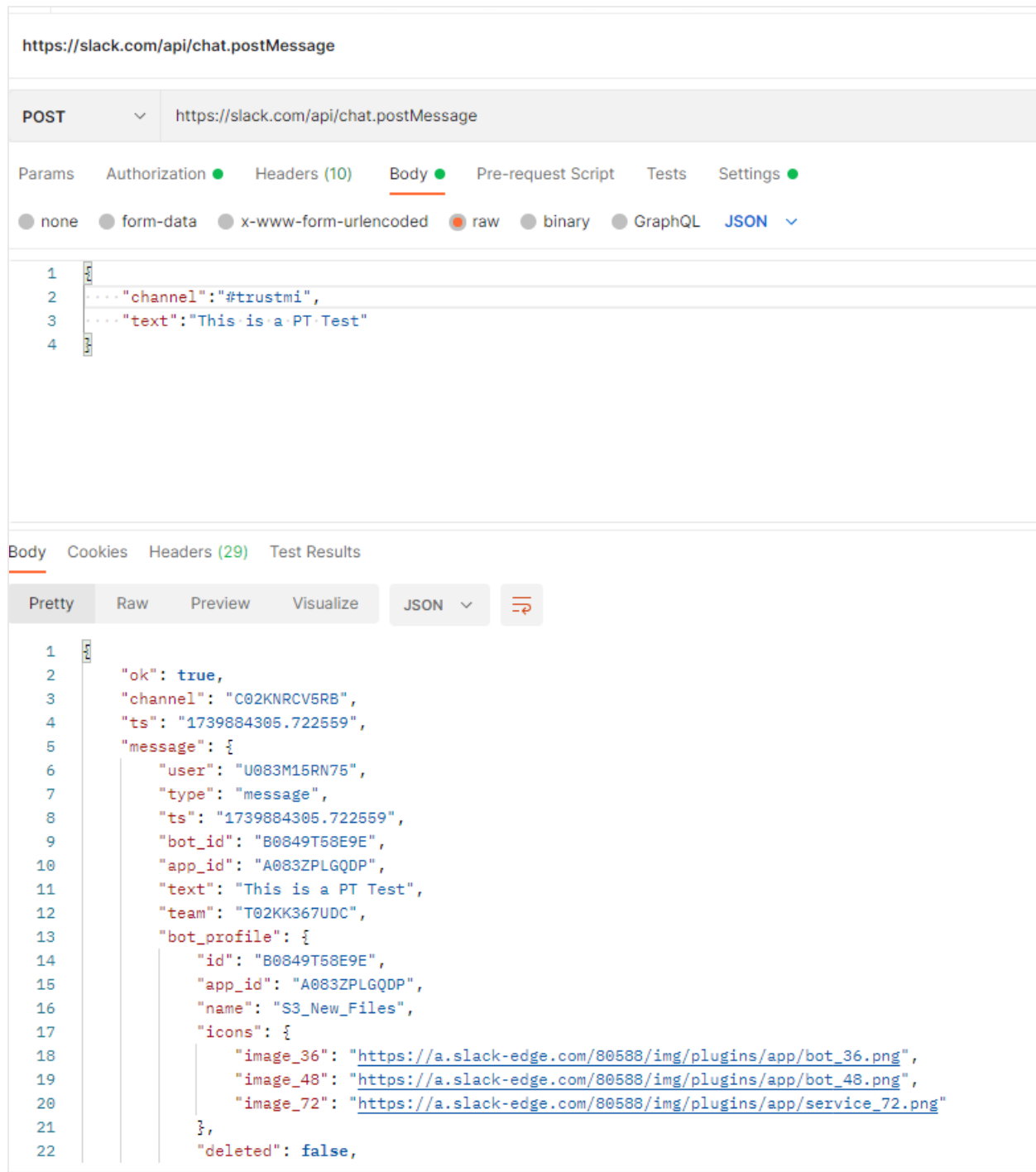
Attack description

An attacker or a malicious insider with sufficient AWS permissions can inspect Lambda function configurations to retrieve these environment variables, as can be seen in the following photo:

```
Pacu (Trustmi25:trustmi) > run lambda_enum
Running module lambda_enum...
[lambda_enum] Starting region us-east-1...
[lambda_enum] Enumerating data for colgate_daily_files
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[lambda_enum] Enumerating data for aws-controltower-NotificationForwarder
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[+] Secret (ENV): sns_arn= arn:aws:sns:us-east-1:233250375748:aws-controltower-AggregateSecurityNotifications
[lambda_enum] Enumerating data for test
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[lambda_enum] Enumerating data for rotate_service_tokens
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[+] Secret (ENV): PREPROD_CM_DOMAIN= https://customer-management.trustmi.dev
[+] Secret (ENV): PROD_CM_DOMAIN= https://customer-management.trustmi.ai
[+] Secret (ENV): PROD_KMS_KEY_ID= mrk-52f9168c56a441b49d1a837d5cf7ae4d
[+] Secret (ENV): PREPROD_KMS_KEY_ID= 21cf709a-dde7-4def-9760-f81e27df7b16
[+] Secret (ENV): SECRETS_MANAGER_ENDPOINT= https://secretsmanager.us-east-1.amazonaws.com
[+] Secret (ENV): SLACK_WEBHOOK_URL= https://hooks.slack.com/services/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
[lambda_enum] Enumerating data for auto_restart_tableau-DEV-893
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[lambda_enum] Enumerating data for New_File_Notification
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[+] Secret (ENV): SLACK_BOT_TOKEN= xoxb-2665108266454-8123039872243-M4I83T7FqzbqDnVNG75jJFvk
[+] Secret (ENV): SLACK_WEBHOOK_URL= https://hooks.slack.com/services/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
[lambda_enum] Enumerating data for tableau_connection_test
[lambda_enum] FAILURE:
[lambda_enum] MISSING NEEDED PERMISSIONS
[lambda_enum] lambda_enum completed.

[lambda_enum] MODULE SUMMARY:
7 functions found in us-east-1. View more information in the DB
```

With access to the Slack bot token and webhook URLs, the attacker can impersonate the bot, sending unauthorized messages, malicious links, or phishing content to Slack channels.



The screenshot displays a REST client interface for a POST request to `https://slack.com/api/chat.postMessage`. The request body is a JSON object with the following structure:

```
1 {
2   "channel": "#trustmi",
3   "text": "This is a PT Test"
4 }
```

The response body is a JSON object with the following structure:

```
1 {
2   "ok": true,
3   "channel": "C02KNRCV5RB",
4   "ts": "1739884306.722559",
5   "message": {
6     "user": "U083M15RN75",
7     "type": "message",
8     "ts": "1739884306.722559",
9     "bot_id": "B0849T58E9E",
10    "app_id": "A083ZPLGQDP",
11    "text": "This is a PT Test",
12    "team": "T02KK367UDC",
13    "bot_profile": {
14      "id": "B0849T58E9E",
15      "app_id": "A083ZPLGQDP",
16      "name": "S3_New_Files",
17      "icons": {
18        "image_36": "https://a.slack-edge.com/80588/img/plugins/app/bot_36.png",
19        "image_48": "https://a.slack-edge.com/80588/img/plugins/app/bot_48.png",
20        "image_72": "https://a.slack-edge.com/80588/img/plugins/app/service_72.png"
21      }
22    },
23    "deleted": false,
24  }
25 }
```

This impersonation could be leveraged to deceive team members, propagate misinformation, or serve as a foothold for further attacks.

Countermeasures

- Migrate all sensitive credentials to AWS Secrets Manager or an equivalent secure storage solution designed for managing secrets.
- If immediate migration is not feasible, ensure that environment variables are encrypted and accessible only to a minimal set of privileged users.

5. Outdated OpenSSH Utilized in the Internal Network

Low Severity

Likelihood: Medium, Impact: Low

Threat Description

During the internal network assessment, it was identified that several systems are running an outdated version of OpenSSH. Although these systems are protected by internal network defenses, the use of an outdated SSH version exposes them to known vulnerabilities and security limitations that have been resolved in later releases.

Impact Description

Once inside the network, an attacker may use the outdated SSH service to gain control over the system and escalate privileges, or as a pivot point for infiltrating other systems.




































Likelihood Description

A malicious actor would need to gain a foothold within the internal network.

Attack description

An attacker who gains internal access or successfully bypasses network segmentation could target these outdated SSH services. Exploitation may involve leveraging known vulnerabilities in OpenSSH.

The following photo show a list of targets that are running an out dated OpenSSH:

Nmap Output		Ports / Hosts	Topology	Host Details	Scans
	Hostname	Port	Protocol	State	Version
	192.168.174.132	22	tcp	open	OpenSSH 9.6p1 Ubuntu 3ubuntu13.8 (Ubuntu Linux; protocol 2.0)
	10.11.231.238	22	tcp	open	OpenSSH 8.9p1 Ubuntu 3ubuntu0.11 (Ubuntu Linux; protocol 2.0)
	192.168.69.130	22	tcp	open	OpenSSH 8.9p1 Ubuntu 3ubuntu0.11 (Ubuntu Linux; protocol 2.0)
	10.11.0.110	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.4.87	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.5.171	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.6.107	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.8.30	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.8.134	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.10.115	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.12.160	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.16.128	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.23.185	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.24.21	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.25.101	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.26.191	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.27.144	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.29.206	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.30.124	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	10.11.31.40	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.128.63	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.136.92	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.145.90	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.149.203	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.151.26	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.151.189	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.151.224	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.153.173	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.162.212	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.165.7	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.166.184	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.166.215	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.178.212	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.181.84	22	tcp	open	OpenSSH 7.4 (protocol 2.0)
	192.168.191.58	22	tcp	open	OpenSSH 7.4 (protocol 2.0)

Countermeasures

- Update all systems running OpenSSH to the latest stable version.

- Implement a robust patch management process to ensure that all internal systems are regularly updated and secured.

6. Username Enumeration via Login Page Error Message

Low Severity

Likelihood: Medium, Impact: Low

Threat Description

A vulnerability was observed on the login page where different error messages are returned depending on the existence of the username provided. This discrepancy enables an attacker to determine the validity of a username, thereby facilitating targeted brute force and credential stuffing attacks.

Impact Description

Gaining valid usernames can help attackers focus their efforts on compromising accounts through brute force or credential stuffing.

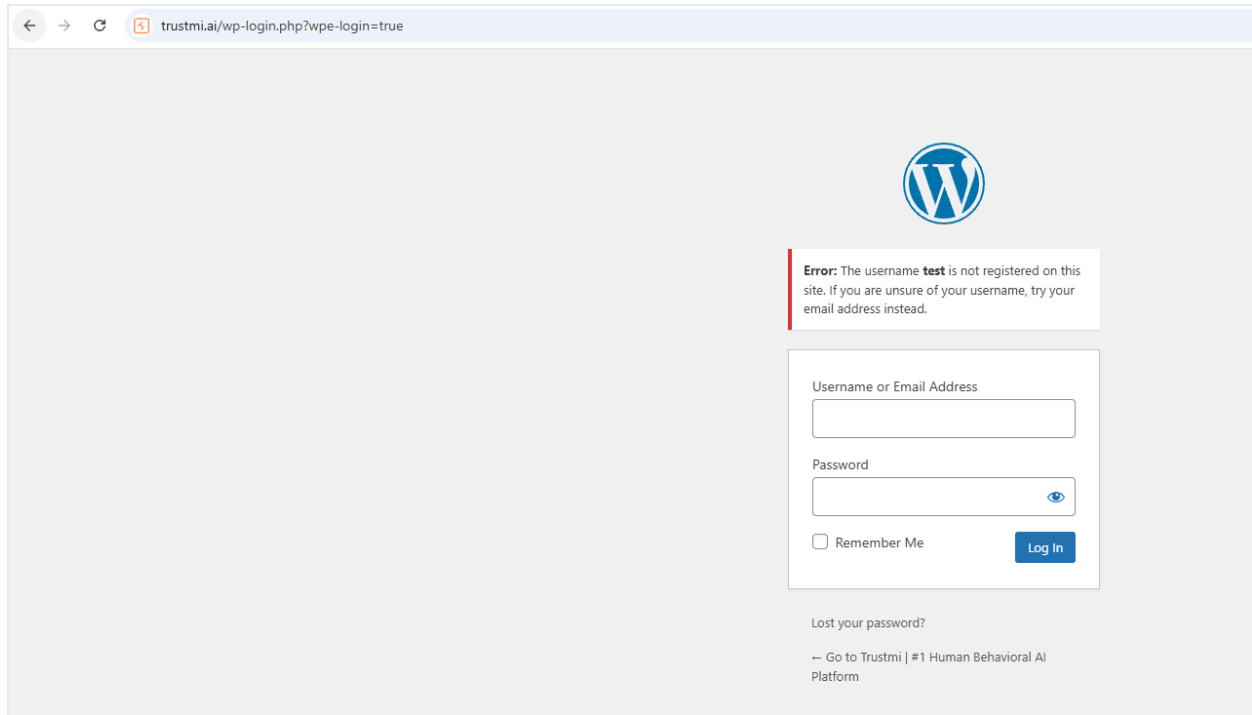
Likelihood Description

The attack is simple to execute using both automated tools and manual methods and relies on the attacker successfully guessing usernames.

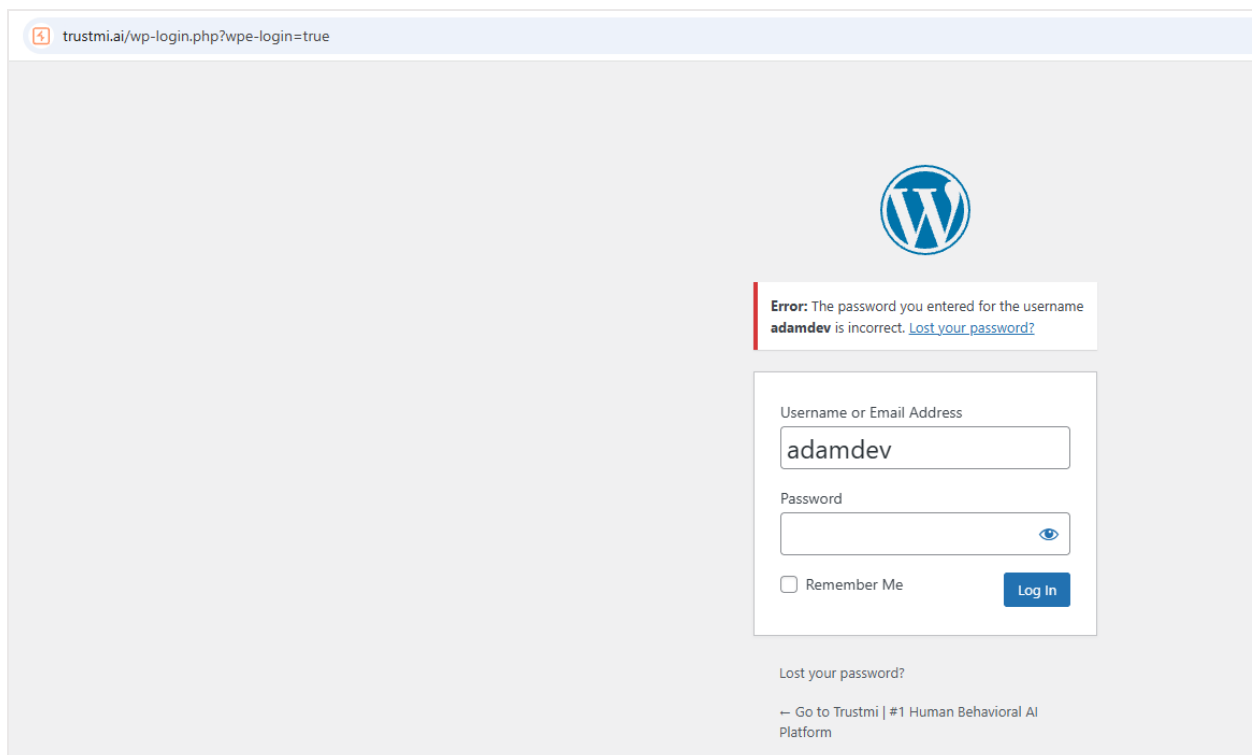
Attack description

By systematically submitting login attempts with various usernames, an attacker can differentiate between valid and invalid accounts based on the error messages received. Once valid usernames are confirmed, attackers can focus on exploiting those accounts through brute force, phishing, or other credential-based attacks.

The following photo shows the error message displayed when attempting to login with a username that does not exist:



The following photo shows that the username exists but the password is incorrect:



Countermeasures

- Send a uniform error message (e.g., “Invalid login credentials”) for all failed login attempts, regardless of whether the username or password is incorrect.

Appendix A Work Scheme

Infrastructure Testing Methodology

2BSecure's inspection methodology relies on ISC², which involves vast testing at the infrastructure level with an emphasis on the development stage. The inspection technique relies on manual tests as well as automatic tools. Testing was performed using a black box approach, meaning the testing team had no prior knowledge of the tested servers. This unique approach offers an actual simulation of a real hacker trying to attack the application, infrastructure, and end users. This test enables determining the security level and vulnerability of the servers and network.

Application Testing Methodology

2BSecure's inspection methodology relies on OWASP, which involves vast testing at the application level with an emphasis on the development stage. The inspection technique relies on manual tests as well as automatic tools. Testing was performed using a black box approach, meaning the testing team had no prior knowledge about the application. This unique approach offers an actual simulation of a real hacker trying to attack the application and the end users. This test enables determining the security level and vulnerability of the application.

List of tests performed during the penetration test:

The following is a list of attacks and tests performed during the investigation. The list includes all known attacks at the present time. It is important to state that the system has been found vulnerable to the attacks described in this document, however there may be vulnerabilities that were not revealed during the test due to time limitations and the test methodology used.

The following table lists the tests performed as part of the penetration tests:

Authentication Verification Requirements	
Test that user-set passwords are at least 12 characters long.	Password Security Requirements
Verify that password change requires the user's current and new password.	
Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks.	General Authenticator Requirements
Test that use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval.	
Verify that the system securely and randomly generates initial passwords and activation codes that are at least 6 characters long, and expire after a short period of time.	Authenticator Lifecycle Requirements
Verify that system-generated initial activation or recovery secrets are not sent in clear text to the user.	
Verify that password credential recovery does not reveal the current password in any way.	Credential Recovery Requirements
Shared or default accounts are not present (e.g.root",admin", orsa").	

Verify that forgotten password and other recovery paths use a secure recovery mechanism.	
Verify that time-based OTPs expire after a pre-defined lifetime.	Single or Multi-factor One Time Verifier Requirements
Session Management Verification Requirements	
Verify that the application generates a new session token upon user authentication.	
Verify that the application only stores session tokens in the browser using secure methods such as appropriately secured cookies or HTML 5 session storage.	Session Binding Requirements
Verify that logout and expiration invalidate the session token, including across relaying parties.	Session Logout and Timeout Requirements
Verify that cookie-based session tokens have the 'Secure' and 'HttpOnly' attributes set.	
Test if the application shares a domain name with other applications that set or use session cookies that overwrite or disclose the session cookies.	Cookie-based Session Management
Verify that the application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions.	Defenses Against Session Management Exploits
Access Control Verification Requirements	
Verify that the application enforces access control rules on a trusted service layer.	
Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless they have been specifically authorized.	
Verify that the principle of least privilege exists. This implies protection against spoofing and elevation of privilege.	General Access Control Design
Verify that the principle of deny by default exists.	
Verify that access controls fail securely, including when an exception occurs.	
Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks.	
Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality.	Operation Level Access Control
Verify that administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	
Verify that directory browsing is disabled unless deliberately desired.	Other Access Control Considerations
Validation, Sanitization and Encoding Verification Requirements	
Verify that the application has defenses against HTTP parameter pollution attacks.	
Verify that the frameworks protect against mass parameter assignment attacks.	
Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc.) is validated.	Input Validation Requirements
Verify that structured data is strongly typed and validated against a defined schema, including allowed characters, length, and pattern.	
Verify that the URL redirects and forwards only allowed destinations that appear on an allow list, or shows a warning when redirecting to untrusted content.	
Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.	
Verify that the application sanitizes user input before passing it on to mail systems, to protect against SMTP or IMAP injection.	
Verify that the application protects against template injection attacks.	
Verify that the application protects against SSRF attacks.	
Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.	
Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) are protected from database injection attacks.	Sanitization and Sandboxing Requirements
Test for JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, Content Security Policy (CSP) bypasses, DOM XSS, and JavaScript expression evaluation.	
Test for LDAP injection vulnerabilities, or that specific security controls that prevent LDAP injection have been implemented.	
Test for OS command injection.	
Test for Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	
Test for XPath injection or XML injection attacks.	
Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.	Deserialization Prevention Requirements
Test for XML eXternal Entity (XXE) attacks.	

Verify that deserialization of untrusted data is avoided or protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	
Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document.	
Error Handling and Logging Verification Requirements	
Verify that a generic message is shown when an unexpected or security sensitive error occurs.	Error Handling
Data Protection Verification Requirements	
Verify that sensitive data is sent to the server in the HTTP message body or headers.	Sensitive Private Data
Verify that all sensitive data created and processed by the application has been identified.	
Communications Verification Requirements	
Verify that secured TLS is used for all client connectivity.	Client Communications Security Requirements
Verify that only strong algorithms, ciphers, and protocols are enabled.	
Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled.	
Malicious Code Verification Requirements	
Test that the application doesn't load or execute code from untrusted sources or the internet, such as loading includes, modules, plugins, code, or libraries.	Deployed Application Integrity Controls
Verify that the application has protection against subdomain takeovers.	Business Logic Security Requirements
Verify that the application will only process business logic flows for the same user in sequential step order and without skipping steps.	
Verify that the application has appropriate limits for specific business actions or transactions, which are correctly enforced on a per-user basis.	
Test for data exfiltration, excessive business logic requests, excessive file uploads, or denial of service attacks.	
Verify that the application has business logic limits or validation, identified using threat modeling or similar methodologies.	
File and Resources Verification Requirements	
Test for acceptance of large files that could fill up storage or cause a denial of service.	File Upload Requirements
Verify that user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path traversal.	File Execution Requirements
Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, update, or removal of local files (LFI).	
Test for disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-Side Request Forgery (SSRF) attacks.	
Test for Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter.	
Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.	
Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.	File Storage Requirements
Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	File Download Requirements
Verify that the web tier is configured to serve only files with specific file extensions, to prevent unintentional information and source code leakage.	
Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.	SSRF Protection Requirements
Verify that the web or application server is configured with an allow list of resources or systems that can send requests or load data/files from.	
API and Web Service Verification Requirements	
Test for parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	Generic Web Service Security Verification Requirements
Verify that access to administration and management functions is limited to authorized administrators.	
Verify that API URLs do not expose sensitive information, such as the API key, session tokens, etc.	RESTful Web Service Verification Requirements
Verify that JSON schema validation is in place and verified before accepting input.	
Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery.	
Configuration Verification Requirements	
Verify that all components are up to date.	Dependency

Verify that all unneeded features, documentation, samples, and configurations are removed.	
Verify that error messages are not showing any unintended security disclosures.	
Verify that debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.	Unintended Security Disclosure Requirements
Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	
Verify that a Content Security Policy (CSP) response header is in place that helps mitigate impact of XSS attacks like HTML, DOM, JSON, and JavaScript injection.	HTTP Security Headers Requirements
Verify that a Strict-Transport-Security header is included in all responses and for all subdomains	
Test that the content of a web application cannot be embedded in a third-party site by default, and frame-ancestors and X-Frame-Options response headers are set.	
Verify that the supplied Origin header is not used for authentication or access control decisions.	Validate HTTP Request Header Requirements
Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header uses a strict allow list of trusted domains and subdomains to match against, and does not support thenull" origin.	