DATA TRICKS

Penetration Testing Report

Prepared By
Mohamed Aziz Mansour & Sahar Ben Cheikh Ahmed

TABLE OF CONTENTS

1.0. Executive Summary

We were entrusted by Data-Tricks to conduct a security assessment of their infrastructure and web application. This assessment was carried out with their explicit permission and strictly adhered to the designated testing scope. The primary objective was to identify potential security vulnerabilities that could be exploited by malicious actors.
The testing process employed a grey box testing approach, providing a balanced perspective. We were granted limited access to certain internal systems and information, simulating the potential knowledge an authorized user might possess. Additionally, industry-standard tools and manual testing techniques were utilized for a comprehensive evaluation. Throughout the process, we maintained strict ethical guidelines and ensured no disruption to the live website or user data.

Scope	Security Level	Grade
Web Application Perimeter	Fair	C

Grading Criteria:

Grade	Security	Criteria Description
A	Excellent	The security exceeds “Industry Best Practice” standards. The overall posture was found to be excellent with only a few low-risk findings identified.
B	Good	The security meets with accepted standards for “Industry Best Practice.” The overall posture was found to be strong with only a handful of medium- and low- risk shortcomings identified.
C	Fair	Current solutions protect some areas of the enterprise from security issues. Moderate changes are required to elevate the discussed areas to “Industry Best Practice” standards
D	Poor	Significant security deficiencies exist. Immediate attention should be given to the discussed issues to address exposures identified. Major changes are required to elevate to “Industry Best Practice” standards.
F	Inadequate	Serious security deficiencies exist. Shortcomings were identified throughout most or even all of the security controls examined. Improving security will require a major allocation of resources.

1.1. Assumptions & Constraints

As the environment changes, and new vulnerabilities and risks are discovered and made public, an organization's overall security posture will change. Such changes may affect the validity of this letter. Therefore, the conclusion reached from our analysis only represents a "snapshot" in time.

1.2. Objective & Scope

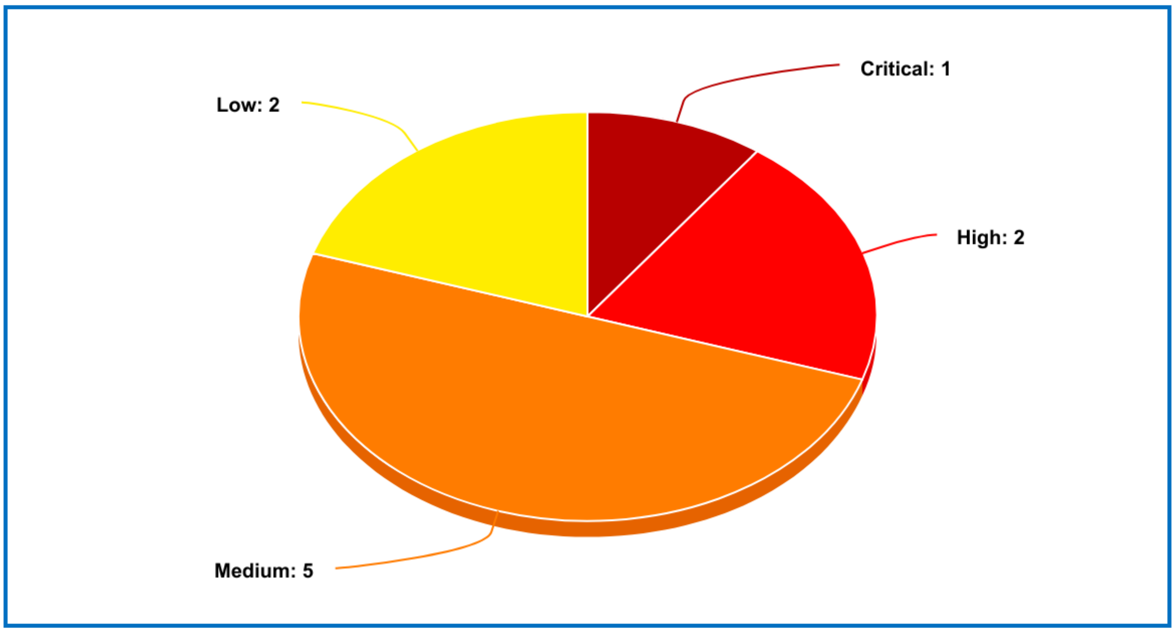
Organization:	Data-Tricks
Audit Type:	Grey Box Penetration Testing
Asset URL:	https://test-kyra.data-tricks.net
Audit Period:	June 20 - August 4 2024

We performed a thorough discovery process to gather crucial information about the target, focusing on identifying information disclosure vulnerabilities. Utilizing this gathered data, We conducted the bulk of the testing manually. This process included input validation tests, impersonation tests (encompassing authentication and authorization), and session state management tests. The primary objective of this penetration testing was to highlight security risks by exploiting weaknesses within the environment, leading to unauthorized access and/or the retrieval of sensitive information. The identified vulnerabilities during the assessment were then used to develop recommendations and mitigation strategies aimed at enhancing the overall security posture.

1.3. Results Overview

The penetration testing uncovered several critical vulnerabilities within the web application. These vulnerabilities pose significant risks, including potential full web application compromise, broken confidentiality and integrity, availability issues, user profile compromise, and user data leakage.

Vulnerability by severity



🟥 Critical 🟥 High 🟧 Medium 🟨 Low

We performed manual security testing according to OWASP Web Application Testing Methodology, which demonstrate the following results.

Severity	**Critical**	**High**	**Medium**	**Low**
# of issues	1	2	5	2

Severity scale:

● **CRITICAL** - Poses immediate danger to systems, network, and/or data security and should be addressed as soon as possible. Exploitation requires little to no special knowledge of the target. Exploitation doesn't require highly advanced skill, training, or tools.

● **HIGH** - Poses significant danger to systems, network, and/or data security. Exploitation commonly requires some advanced knowledge, training, skill, and/or tools. Issue(s) should be addressed promptly.

● **MEDIUM** - Vulnerabilities should be addressed in a timely manner. Exploitation is usually more difficult to achieve and requires special knowledge or access. Exploitation may also require social engineering as well as special conditions.

● **LOW** - Danger of exploitation is unlikely as vulnerabilities offer little to no opportunity to compromise system, network, and/or data security. Can be handled as time permits.

1.3. Methodology:

Our Penetration Testing Methodology grounded on following guides and standards:

- Pentration Testing Execution Standard
- OWASP Top 10 Application Secirity Risks - 2021
- OWASP Testing Guide

Open Web Application Security Project (OWASP) is an industry initiative for web application security. OWASP has identified the 10 most common attacks that succeed against web applications. These comprise the OWASP Top 10.

Application penetration test includes all the items in the OWASP Top 10 and more. The penetration tester remotely tries to compromise the OWASP Top 10 flaws. The flaws listed by OWASP in its most recent Top 10 and the status of the application against those are depicted in the table below.

1.4. Performed Tests:

● All set of applicable OWASP Top 10 Security Threats

Criteria Label	Status
A01 Broken Access Control	Fails Criteria
A02 Cryptographic Failures	Fails Criteria
A03 Injection	Fails Criteria
A04 Insecure Design	Meets Criteria
A05 Security Misconfiguration	Fails Criteria
A06 Vulnerable And Outdated Components	Fails Criteria
A07 Identificationand Authentication Failures	Fails Criteria
A08 Software and Data Integrity Failures	Meets Criteria
A09 Security Logging and Monitoring Failures	Meets Criteria
A10 Server Side Request Forgery (SSRF)	Meets Criteria

1.5. Security Tools Used

OS Distributions:

- BlackArch Linux Distro
- Kali Linux Distro

Web Application Scanners:

- Burp Suite (Community Edition)
- OWASP ZAP
- Nessus (Essentials)
- Sn1per

Enumeration Tools:

- Nmap

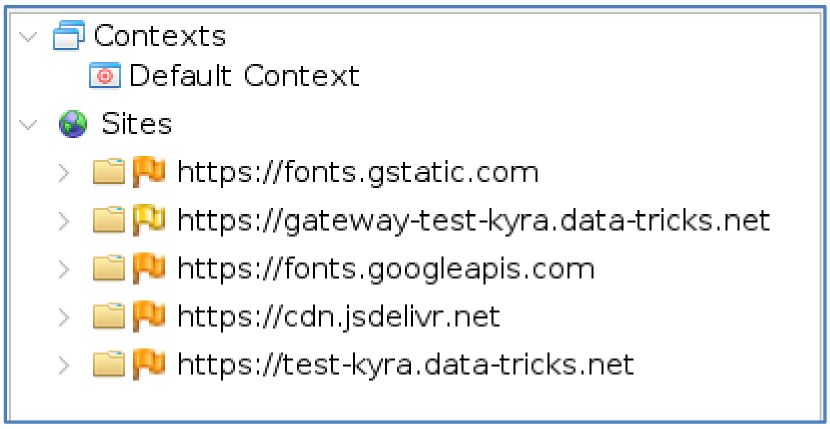
XSS and SQL Scanners:

- XSPEAR
- XSSTRIKE
- PPMAP
- SQLMAP

2.0. Findings Details:

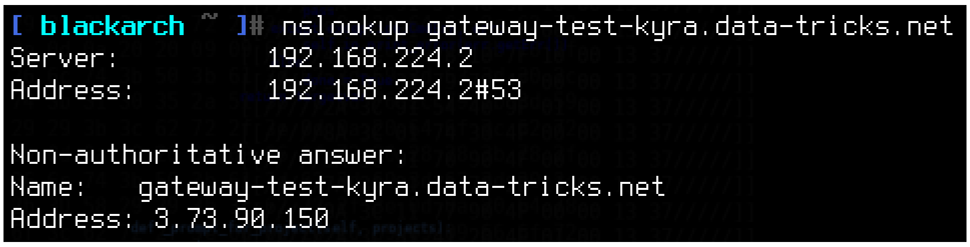
2.1. INFORMATION GATHERING:

A standard user account was provided by the application developers for penetration testing purposes. Using OWASP ZAP, this account was successfully used to authenticate and identify the backend URL:



```
Contexts
    Default Context
Sites
    https://fonts.gstatic.com
    https://gateway-test-kyra.data-tricks.net
    https://fonts.googleapis.com
    https://cdn.jsdelivr.net
    https://test-kyra.data-tricks.net
```

To identify the IP address associated with the domain gateway-test-kyra.data-tricks.net, the nslookup command was executed, revealing the IP address as **3.73.90.158**



```
[ blackarch ~ ]# nslookup gateway-test-kyra.data-tricks.net
Server:         192.168.224.2
Address:        192.168.224.2#53

Non-authoritative answer:
Name:   gateway-test-kyra.data-tricks.net
Address: 3.73.90.150
```

2.2. ENUMERATION/SCANNING:

Following the identification of the backend URL's domain (gateway-test-kyra.data-tricks.net), an Nmap scan was conducted using sn1per to enumerate open ports on the corresponding IP address (3.73.90.150). The scan revealed several open ports commonly associated with web services, including port 22 (SSH), 80 (HTTP), and 443 (HTTPS). Additionally, the scan identified open or filtered UDP ports often used for various network services like DNS (53), DHCP (67 & 68), TFTP (69), Kerberos (88), NetBIOS (137-139), SNMP (161 & 162), LDAP (389), ISAKMP (500), routing (520), and NFS (2049). These findings provide valuable insight into the potential services running on the backend infrastructure and will be crucial for further penetration testing efforts.

```
====================================================================================•x[2024-
07-05](12:29)x•
 PERFORMING TCP PORT SCAN
====================================================================================•x[2024-
07-05](12:29)x•
Starting Nmap 7.80 ( https://nmap.org ) at 2024-07-05 12:29 CET
Nmap scan report for ec2-3-73-90-150.eu-central-1.compute.amazonaws.com (3.73.90.150)
Host is up (0.16s latency).
Not shown: 65532 filtered ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE         SERVICE
22/tcp   open          ssh
80/tcp   open          http
443/tcp  open          https
53/udp   open|filtered domain
67/udp   open|filtered dhcps
68/udp   open|filtered dhcpc
69/udp   open|filtered tftp
88/udp   open|filtered kerberos-sec
137/udp  open|filtered netbios-ns
138/udp  open|filtered netbios-dgm
139/udp  open|filtered netbios-ssn
161/udp  open|filtered snmp
162/udp  open|filtered snmptrap
389/udp  open|filtered ldap
500/udp  open|filtered isakmp
520/udp  open|filtered route
2049/udp open|filtered nfs

Nmap done: 1 IP address (1 host up) scanned in 1496.16 seconds
```

2.3. VULNERABILITIES/MITIGATION:

Forwarded SSH-Agent Remote Code Execution

SEVERITY: CRITICAL

ISSUE DESCRIPTION:

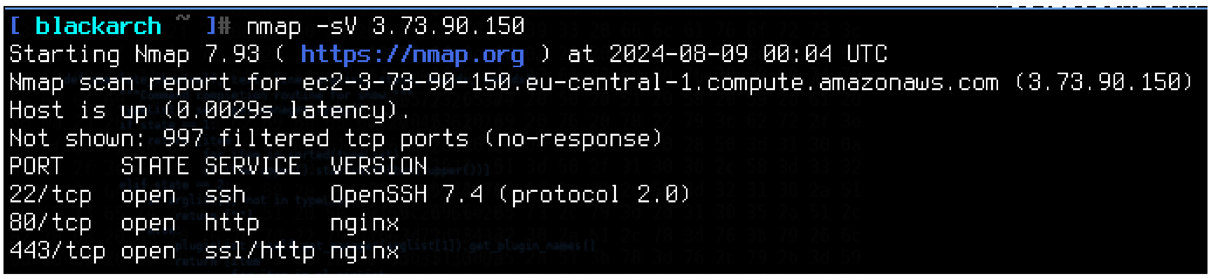
A critical remote code execution (RCE) vulnerability (CVE-2023-38408) exists in OpenSSH versions before 9.3p2 due to an insufficiently trustworthy search path within the PKCS#11 feature of the forwarded ssh-agent. This vulnerability allows an attacker to execute arbitrary code on the target system if a user forwards their SSH agent to a compromised system.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2023-38408, a critical severity remote code execution vulnerability

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, CVE-2023-
38408 9.8 https://vulners.com/cve/CVE-2023-38408
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, B8190CDB-
3EB9-5631-9828-8064A1575B23 9.8 https://vulners.com/githubexploit/B8190CDB-3EB9-
5631-9828-8064A1575B23
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, 8FC9C5AB-
3968-5F3C-825E-E8DB5379A623 9.8 https://vulners.com/githubexploit/8FC9C5AB-3968-
5F3C-825E-E8DB5379A623
```

To verify the OpenSSH version running on the target system, an Nmap scan was performed against the IP address 3.73.90.150. The results indicate that OpenSSH version 7.4 is in use, which is vulnerable to CVE-2023-38408.



```
[ blackarch ~ ]# nmap -sV 3.73.90.150
Starting Nmap 7.93 ( https://nmap.org ) at 2024-08-09 00:04 UTC
Nmap scan report for ec2-3-73-90-150.eu-central-1.compute.amazonaws.com (3.73.90.150)
Host is up (0.0029s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      OpenSSH 7.4 (protocol 2.0)
80/tcp  open  http     nginx
443/tcp open  ssl/http nginx
```

ATTACKER PROFILE:

- **Skill Level:** High (requires understanding of shared library side effects and potentially fuzzing techniques).
- **Motivation:** Gaining unauthorized access to the victim's workstation, data theft, system compromise.
- **Resources:** Access to a compromised remote server where Alice forwards her SSH agent.

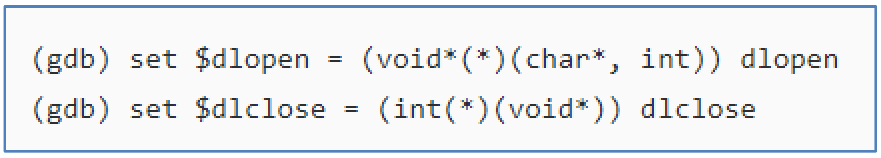
ATTACK SCENARIO:

Imagine this: You have a laptop (your local machine) with your SSH keys stored securely. You want to access a server without typing your password every time. To make this easier, you enable **SSH agent forwarding** when connecting to the server. This means your laptop's SSH agent is now accessible from the server.

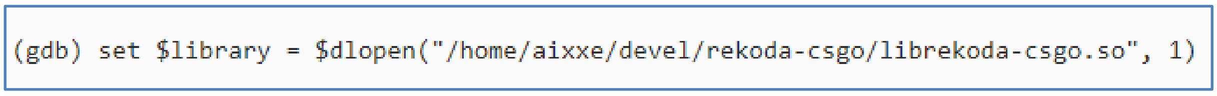
While this is convenient, it also introduces risks. If the remote server is compromised, an attacker could potentially exploit this access to gain unauthorized access to other systems using the same SSH keys.

The following scenario outlines a specific attack that leverages this vulnerability on Alice's workstation:

1. Attacker gains access to a remote server where Alice forwards her SSH agent.
2. Attacker stores a malicious shared library somewhere on Alice's workstation
3. The side effects of loading **(dlopen())** and unloading **(dlclose())** these libraries trigger malicious code execution within the SSH agent process (ssh-pkcs11-helper).



```
(gdb) set $dlopen = (void*(*)(char*, int)) dlopen
(gdb) set $dlclose = (int(*)(void*)) dlclose
```



```
(gdb) set $library = $dlopen("/home/aixxe/devel/rekoda-csgo/librekoda-csgo.so", 1)
```

4. The attacker gains remote code execution on Alice's workstation, potentially leading to data exfiltration, system takeover, or other malicious actions.

MITIGATION STRATEGIES:

- Disable SSH agent forwarding unless absolutely necessary.
- Upgrade OpenSSH to versions patched for CVE-2023-38408 (version 9.3p2 or later).
- Implement strict access controls on systems where SSH agent forwarding might be required.
- Consider alternative authentication methods that don't require agent forwarding.

SCP Command Injection Vulnerability

SEVERITY: HIGH

ISSUE DESCRIPTION:

OpenSSH versions prior to 8.3p1 are susceptible to a command injection vulnerability within the scp command's toremote function. By manipulating the destination filename argument with specific characters, such as backticks, an attacker can execute arbitrary commands on the remote system. This flaw is attributed to the intentional omission of input validation for "anomalous argument transfers" by the OpenSSH developers, citing potential disruptions to existing workflows. Successful exploitation of this vulnerability could lead to severe consequences, including remote code execution, data exfiltration, and privilege escalation.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2020-15778, a high severity command injection vulnerability affecting the OpenSSH scp command.

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, CVE-2020-
15778 7.8 https://vulners.com/cve/CVE-2020-15778
```

ATTACKER PROFILE:

- **Skill Level:** Medium (requires basic understanding of command injection vulnerabilities and shell scripting).
- **Motivation:** Data exfiltration, system compromise, financial gain.
- **Resources:** Network access to target systems.

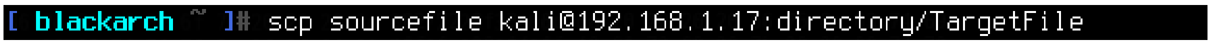
ATTACK SCENARIO:

About SCP:

scp is a program for copying files between computers. It uses the SSH protocol. It is included by default in most Linux and Unix distributions

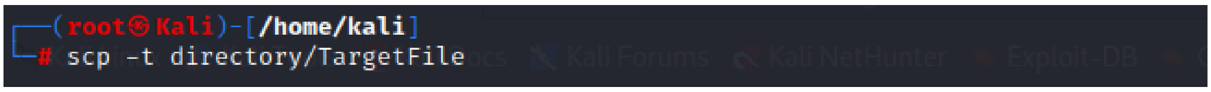
Issue:

While copying files to remote server, file path is appended at end of local scp command. For example, if you execute following command:



```
[ blackarch ~ ]# scp sourcefile kali@192.168.1.17:directory/TargetFile
```

it will execute a local command:

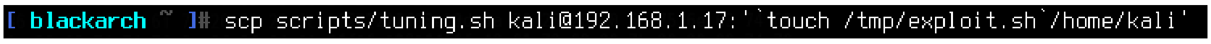


```
┌──(root㉿Kali)-[/home/kali]
└─# scp -t directory/TargetFile
```

At time of creating local scp command, it does not sanitise file name. An attacker can pass a backtick enabled payload as file name and when local scp command is executed, local shell will also execute backtick enabled payload.

Example:

We execute following command with scp:



```
[ blackarch ~ ]# scp scripts/tuning.sh kali@192.168.1.17:'`touch /tmp/exploit.sh`/home/kali'
```

After executing this command, go to remote server and you will see in /tmp/ directory that file exploit.sh is present. Putting single quotes in file name is important to prevent payload execution on local shell or using escape character like single quotes in file name can prevent payload execution on local shell.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH:** The most effective mitigation is to upgrade OpenSSH to a version that addresses the vulnerability. This is typically version 8.3p1 or later. Upgrading ensures the developers' implemented fixes are in place.
- **Restrict Usage of scp:** If immediate upgrade isn't feasible, consider restricting the use of scp by disabling it for specific users or groups. This might be suitable for environments where other secure file transfer methods are available for authorized users.
- **Implement Input Validation:** For advanced users and administrators, consider implementing server-side input validation for the scp command. This can involve modifying the OpenSSH source code to sanitize user-provided filenames before processing. However, this requires expertise and thorough testing to avoid unintended consequences.

Privilege Escalation via AuthorizedKeysCommand Misconfiguration

SEVERITY: HIGH

ISSUE DESCRIPTION:

A privilege escalation vulnerability exists in OpenSSH versions 6.2 through 8.7 due to the incorrect initialization of supplemental groups when executing the ‘AuthorizedKeysCommand’ or ‘AuthorizedPrincipalsCommand’ with the ‘AuthorizedKeysCommandUser’ or 'AuthorizedPrincipalsCommandUser’ directives set to run the command as a different user. This issue allows these commands to inherit the group memberships of the sshd process, potentially leading to unintended privilege escalation.

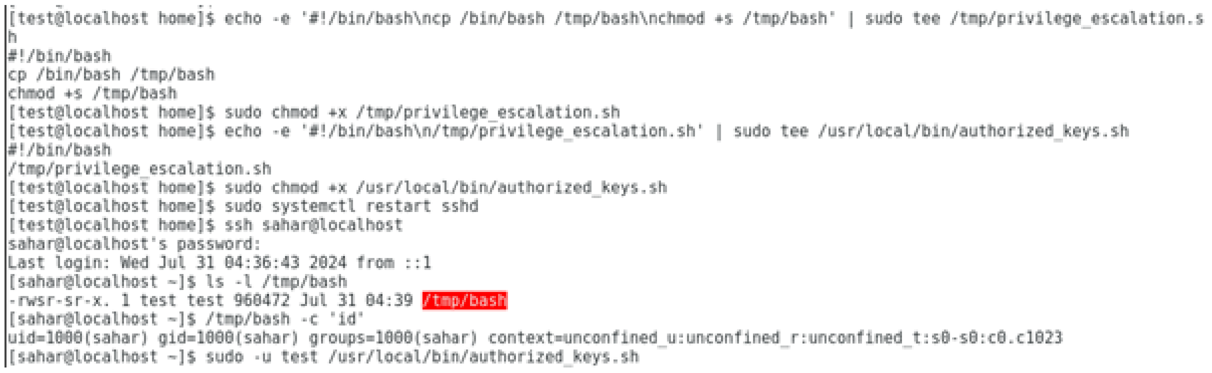
PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2021-41617, a high severity Privilege escalation vulnerability affecting OpenSSH, specifically related to the misconfiguration of the ‘AuthorizedKeysCommand’.

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, CVE-2021-
41617 7.0 https://vulners.com/cve/CVE-2021-41617
```

ATTACK SCENARIO:

Consider a scenario where an attacker gains limited access to a system, as a non-root user. If the system is configured with an insecure AuthorizedKeysCommand, the attacker could modify the command to execute arbitrary code with root privileges. This would allow the attacker to gain full control over the system, leading to severe security breaches.



```
[test@localhost home]$ echo -e '#!/bin/bash\ncp /bin/bash /tmp/bash\nchmod +s /tmp/bash' | sudo tee /tmp/privilege_escalation.s
h
#!/bin/bash
cp /bin/bash /tmp/bash
chmod +s /tmp/bash
[test@localhost home]$ sudo chmod +x /tmp/privilege_escalation.sh
[test@localhost home]$ echo -e '#!/bin/bash\n/tmp/privilege_escalation.sh' | sudo tee /usr/local/bin/authorized_keys.sh
#!/bin/bash
/tmp/privilege_escalation.sh
[test@localhost home]$ sudo chmod +x /usr/local/bin/authorized_keys.sh
[test@localhost home]$ sudo systemctl restart sshd
[test@localhost home]$ ssh sahar@localhost
sahar@localhost's password:
Last login: Wed Jul 31 04:36:43 2024 from ::1
[sahar@localhost ~]$ ls -l /tmp/bash
-rwsr-sr-x. 1 test test 960472 Jul 31 04:39 /tmp/bash
[sahar@localhost ~]$ /tmp/bash -c 'id'
uid=1000(sahar) gid=1000(sahar) groups=1000(sahar) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[sahar@localhost ~]$ sudo -u test /usr/local/bin/authorized_keys.sh
```

The commands shown in the screenshot illustrate an attempt to exploit a misconfigured AuthorizedKeysCommand by using it to run a custom script that modifies the permissions of the bash binary in the /tmp/ directory. The goal was to create a SUID-enabled binary that would allow the attacker to execute commands with root privileges.

However, the exploitation attempt did not yield a successful result. This could be due to several factors:

1. **File Permissions:** The bash binary in /tmp/ was created with SUID, but it appears that the script execution did not fully escalate privileges. This could be due to underlying security mechanisms, such as SELinux , that prevent the SUID binary from being exploited as intended.
2. **Environment Constraints:** The exploitation was tested in a local environment where certain restrictions might be in place that prevent the successful execution of the SUID binary. For example, the system might be configured to restrict the execution of SUID scripts, or there might be additional security layers that prevent the privilege escalation.
3. **Hardening and Auditing Tools:** The local environment might include additional hardening and auditing measures that detect and prevent such exploitation attempts. These measures could include file integrity checks, process monitoring, or other security controls that neutralize the exploit.

This example highlights the importance of securing the AuthorizedKeysCommand configuration to prevent unauthorized modifications and potential privilege escalations. Even though the exploitation attempt in this case was not successful, it underscores the need for robust security practices to mitigate such risks.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH:** Ensure that the system runs the latest version of OpenSSH, as newer versions may contain security enhancements that mitigate this and other related vulnerabilities.
- **Secure AuthorizedKeysCommand Configuration:** Ensure that the script or command used in AuthorizedKeysCommand is owned by root and is not writable by other users. This will prevent unauthorized modification of the command.

- **Monitor and Audit Configurations:** Regularly audit the OpenSSH configuration files to detect any potential misconfigurations. Ensure that all scripts and commands executed with elevated privileges are secure.
- **Implement Additional Security Measures:** Implement additional security measures, such as SELinux or AppArmor, to restrict the actions that can be performed by processes running under the SSH service.

OpenSSH User Enumeration Vulnerability

SEVERITY: MEDIUM

ISSUE DESCRIPTION:

The vulnerability allows an attacker to determine whether a specific username exists on a target system by analyzing the response time of authentication attempts. This flaw is particularly dangerous as it enables attackers to gather valid usernames, which can later be used in brute-force attacks or other forms of unauthorized access attempts.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2018-15473, a high severity user enumeration vulnerability in OpenSSH. This flaw allows attackers to verify the existence of specific usernames on the system.

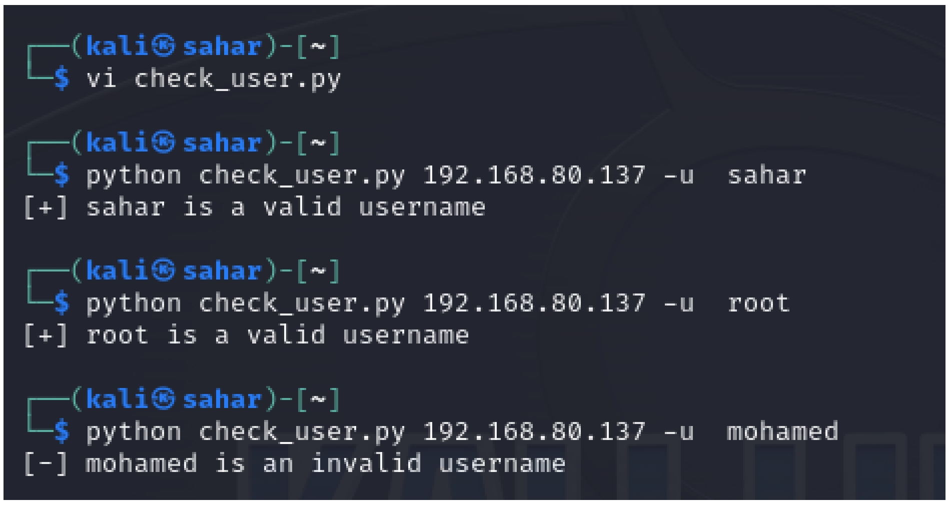
```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150, CVE-2018-15473 5.3
https://vulners.com/cve/CVE-2018-15473
```

ATTACK SCENARIO:

Imagine an attacker wanting to compromise a system protected by OpenSSH. The first step would often involve identifying valid usernames to target in subsequent attacks. By exploiting CVE-2018-15473, an attacker can systematically test usernames and, based on response times, accurately determine which ones are valid. This knowledge dramatically increases the success rate of subsequent brute-force or credential-stuffing attacks.

To verify this, a Python script available on GitHub was used to perform the exploitation.

- **Script Download:** The exploitation script used for this test was downloaded from GitHub.
- **Script Execution:** The script was executed against the target system to test various usernames. The results demonstrated that the response could be used to identify whether a specific username exists on the target system.
- Below is a screenshot showing the script being used to identify valid and invalid usernames on the target system.



```
┌──(kali㉿sahar)-[~]
└─$ vi check_user.py

┌──(kali㉿sahar)-[~]
└─$ python check_user.py 192.168.80.137 -u  sahar
[+] sahar is a valid username

┌──(kali㉿sahar)-[~]
└─$ python check_user.py 192.168.80.137 -u  root
[+] root is a valid username

┌──(kali㉿sahar)-[~]
└─$ python check_user.py 192.168.80.137 -u  mohamed
[-] mohamed is an invalid username
```

MITIGATION STRATEGIES:

- **Upgrade OpenSSH:** The most effective way to mitigate this vulnerability is by upgrading OpenSSH to a version that no longer contains the flaw. Version 7.8 or later addresses this vulnerability.
- **Disable Public Key Authentication:** If an upgrade isn't immediately possible, consider disabling public key authentication or using alternative authentication methods less susceptible to timing attacks.
- **Implement Additional Security Measures:** To further mitigate risks, consider implementing additional security layers, such as account lockouts after multiple failed attempts, two-factor authentication, or restricting SSH access to trusted IP addresses only.
- **Monitor and Audit Logs:** Regularly monitoring and auditing SSH logs can help detect unusual login attempts, enabling quicker responses to potential attacks.

Man-in-the-Middle Attack via Arbitrary stderr in OpenSSH

SEVERITY: MEDIUM

ISSUE DESCRIPTION:

OpenSSH is susceptible to Man-in-the-Middle (MITM) attacks due to two specific vulnerabilities:

- **CVE-2019-6110**: This vulnerability arises from improper handling of arbitrary stderr output. A malicious server or MITM attacker can exploit this flaw to inject ANSI control codes into the stderr stream, potentially manipulating the terminal display and hiding malicious activities or unauthorized file transfers.
- **CVE-2019-6109**: This issue is related to the absence of character encoding in the progress display during file transfers. An attacker can craft specific object names to exploit this

vulnerability, obscuring file transfers and potentially concealing additional malicious activities from the user.

Both vulnerabilities pose significant risks in environments where OpenSSH is used for secure file transfers or system management. By leveraging these flaws, an attacker could obscure their activities, making it difficult for users to detect unauthorized actions or malicious content.

PROOF OF VULNERABILITY:

A vulnerability scan using the **sn1per** tool identified the target system (3.73.90.150) as vulnerable to **CVE-2019-6110** and **CVE-2019-6109**.

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150,
CVE-2019-6110 6.8 https://vulners.com/cve/CVE-2019-6110
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150,
CVE-2019-6109 6.8 https://vulners.com/cve/CVE-2019-6109
```

ATTACK SCENARIO:

1. **Intercept Connection**: An attacker positions themselves between a legitimate user and the OpenSSH server, typically using MITM techniques such as ARP spoofing or packet sniffing.
2. **Exploit CVE-2019-6110**: By manipulating stderr output, the attacker can inject crafted ANSI control codes into the SSH session. This manipulation can hide unauthorized file transfers or other malicious activities from the user's terminal.
3. **Exploit CVE-2019-6109**: During a file transfer, the attacker uses crafted object names to manipulate the progress meter display. This can obscure the status of file transfers and conceal additional files being moved or manipulated.
4. **Impact**: The user may remain unaware of the attack due to the obscured output, leading to potential unauthorized actions and compromised security.

Note:

During the penetration testing process, while the presence of vulnerabilities CVE-2019-6110 and CVE-2019-6109 was confirmed, no active exploitation of these vulnerabilities was observed. This indicates that, under the current testing conditions, the vulnerabilities remain unexploited. However, it is important to recognize that these vulnerabilities are still valid and could be exploited under different circumstances. Therefore, immediate mitigation and continuous monitoring are strongly recommended to prevent potential exploitation.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH**: The most effective mitigation strategy is to upgrade to OpenSSH version 8.0 or later, where these vulnerabilities have been addressed.
- **Monitor SSH Connections**: Implement continuous monitoring of SSH connections and session logs. Look for signs of irregular stderr output or unusual behavior in the progress meter during file transfers. Enhanced logging and alerting can help identify potential exploitation attempts.

- **Harden SSH Configuration**: Limit SSH access to trusted IP addresses only, disable unused authentication methods, and enable verbose logging to capture detailed session activity. This can help detect and prevent suspicious actions.
- **Implement Additional Security Measures**: Consider employing multi-factor authentication (MFA) to strengthen authentication processes. Regularly audit SSH configurations and security practices to minimize vulnerabilities and reduce the attack surface.
- **Conduct Regular Security Assessments**: Periodically perform vulnerability assessments and penetration testing to identify and address any new or existing security issues in your SSH implementations.

Remote Access Restriction Bypass

SEVERITY: MEDIUM

ISSUE DESCRIPTION:

CVE-2018-20685 is a vulnerability in OpenSSH 7.9's scp client that allows remote SSH servers to bypass access restrictions on the client side. The vulnerability arises from improper handling of filenames that are either empty or represented by a dot (.). An attacker can exploit this flaw to modify the permissions of the target directory on the client side, potentially leading to unauthorized access or modification of the client's filesystem.
This vulnerability is particularly concerning in scenarios where an attacker controls the SSH server or is able to perform a Man-in-the-Middle (MITM) attack, as it allows them to manipulate the target directory's permissions during an scp operation without the client's consent.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2018-20685. This flaw allows a malicious server or attacker to bypass access restrictions and modify the target directory's permissions or inject malicious files during an scp operation.

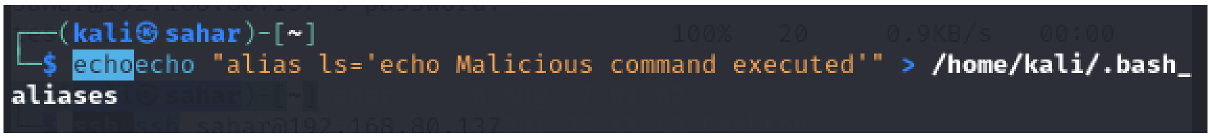
```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150,
CVE-2018-20685 5.3 https://vulners.com/cve/CVE-2018-20685
```

ATTACK SCENARIO:

Overview of Exploitation Steps

In this exploitation scenario, the attacker aims to gain unauthorized control over the victim’s system by utilizing a vulnerable scp operation. The steps below demonstrate how the attacker creates a malicious .bash_aliases file, transfers it to the victim’s home directory, and verifies that the malicious alias has been applied. This process leverages vulnerabilities in the scp protocol to execute arbitrary commands on the victim’s machine.

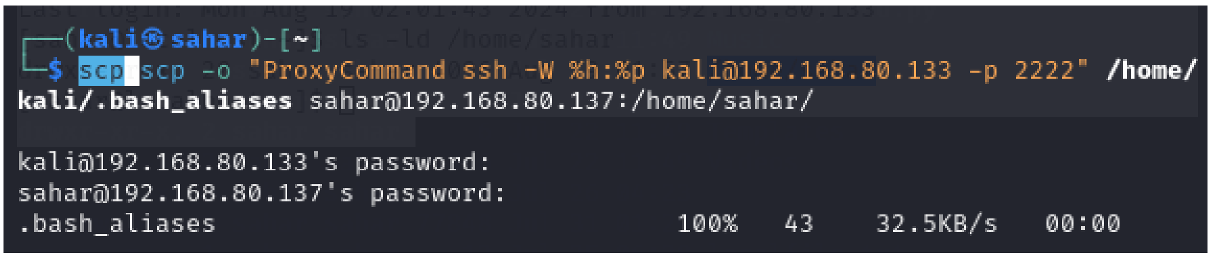
Step 1: Creating the Malicious .bash_aliases File



```
┌──(kali㉿sahar)-[~]
└─$ echoecho "alias ls='echo Malicious command executed'" > /home/kali/.bash_
aliases
```

This command creates a file called .bash_aliases on the attacker's machine, containing an alias that alters the behavior of the ls command. The alias is set to print "Malicious command executed" when the ls command is run.

Step 2: Transferring the Malicious File

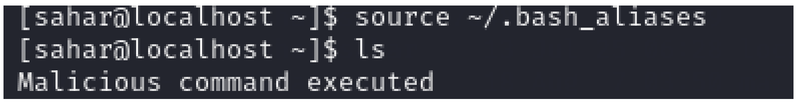


```
┌──(kali㉿sahar)-[~]
└─$ scp scp -o "ProxyCommand ssh -W %h:%p kali@192.168.80.133 -p 2222" /home/
kali/.bash_aliases sahar@192.168.80.137:/home/sahar/

kali@192.168.80.133's password:
sahar@192.168.80.137's password:
.bash_aliases                                 100%   43    32.5KB/s   00:00
```

This command uses scp to transfer the .bash_aliases file from the attacker's machine to the victim's home directory on the victim's machine. The ProxyCommand option ensures the connection is routed through the appropriate SSH proxy.

Step 3: Verifying the Malicious Alias



```
[sahar@localhost ~]$ source ~/.bash_aliases
[sahar@localhost ~]$ ls
Malicious command executed
```

Purpose:

- source ~/.bash_aliases: Applies the alias changes defined in the .bash_aliases file on the victim's machine.
- ls: Executes the ls command to verify that the malicious alias is working. The expected result is "Malicious command executed," indicating that the alias has been successfully applied.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH**: The most effective mitigation strategy is to upgrade to OpenSSH version 8.0 or later, where this vulnerability has been addressed.
- **Switch to SFTP**: Whenever possible, use SFTP (Secure File Transfer Protocol) instead of SCP, as it is not affected by this vulnerability.
- **Harden SCP Configuration**: Apply patches or configuration changes to prevent directory permission changes during SCP operations. Ensure that only trusted SSH servers are used for file transfers.
- **Regular Audits and Monitoring**: Regularly audit file permissions and monitor for any unusual changes to directory structures on client systems.

Critical Bypass of Integrity Checks

SEVERITY: MEDIUM

ISSUE DESCRIPTION:

CVE-2023-48795 is a vulnerability in certain versions of SSH, including OpenSSH before 9.6, that allows remote attackers to bypass integrity checks during the SSH handshake. This vulnerability, known as the Terrapin attack, occurs because of improper handling of the SSH Binary Packet Protocol (BPP) during the handshake phase. An attacker can exploit this flaw to strip critical packets from the handshake, potentially downgrading security features or disabling them altogether.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2023-48795. This flaw allows an attacker to bypass integrity checks during the SSH handshake, which can lead to downgraded security features or disabled protections.

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150,
CVE-2023-48795 5.9 https://vulners.com/cve/CVE-2023-48795
```

ATTACK SCENARIO:

Imagine an attacker, Alice, wants to exploit CVE-2023-48795 on a company's SSH server. She sets up a MitM attack by positioning herself between the SSH client Bob and the server. Alice uses ARP spoofing to redirect Bob's traffic through her machine.

Alice then runs custom scripts designed to inject SSH_MSG_IGNORE packets during the SSH handshake. These scripts also remove these packets immediately after the handshake completes. As Alice monitors the traffic with Wireshark, she can see that the integrity checks of the SSH connection have been bypassed. This manipulation could allow Alice to exploit the weakened security features of the SSH connection.

Detailed steps:

1. **Setup Man-in-the-Middle (MitM) Environment:**
- **Positioning:**
  - Alice configures her system to intercept traffic between the SSH client Bob and the SSH server. She achieves this by using ARP spoofing to redirect Bob's traffic through her machine.
2. **Prepare Custom Scripts:**
- **Script Development:**
  - Alice writes scripts that can inject SSH_MSG_IGNORE packets during the SSH handshake phase and remove these packets after the handshake is complete.
- **Script Execution:**
  - These scripts need to handle the precise timing for packet injection and removal to exploit the vulnerability effectively.
3. **Inject and Remove Packets:**
- **Inject Packets:**

- During the SSH handshake, Alice runs her scripts to inject SSH_MSG_IGNORE packets. This is done to manipulate the handshake process and bypass integrity checks.
- **Remove Packets:**
  - Immediately after the handshake is complete, Alice's scripts remove the injected packets to finalize the attack.

4. **Monitor and Analyze Traffic:**

- **Traffic Capture:**
  - Alice uses network analysis tools like Wireshark to capture the SSH traffic. She ensures that her packet manipulations are effectively bypassing the integrity checks.
- **Traffic Analysis:**
  - She verifies that the SSH connection has been downgraded or that security features have been disabled as a result of the attack.

Note:
Exploiting CVE-2023-48795 requires sophisticated techniques, including custom scripting and advanced traffic manipulation. Standard tools may not be adequate for detecting or exploiting this vulnerability, and specific testing setups are necessary.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH:** Update to version 9.6 or later, where this vulnerability has been addressed.
- **Use Strict Key Exchange:** Apply "strict kex" configurations to prevent unauthenticated message introduction.
- **Switch to Secure Algorithms:** Temporarily disable affected algorithms and use alternatives like AES GCM until patches are available.
- **Regular Audits:** Regularly check and monitor SSH configurations and logs for any unusual activities or changes.

Bypassing Read-Only Restrictions in OpenSSH SFTP Server Write Operations

SEVERITY: MEDIUM

ISSUE DESCRIPTION:

CVE-2017-15906 is a vulnerability found in OpenSSH versions prior to 7.6 that affects the SFTP server's handling of file operations in read-only mode. The issue arises due to improper handling of file operations within the process_open function in sftp-server.c. This flaw allows attackers to bypass the intended read-only restrictions and create zero-length files, despite the directory being marked as read-only.

PROOF OF VULNERABILITY:

A vulnerability scan using the sn1per tool identified the target system (3.73.90.150) as vulnerable to CVE-2017-15906. This flaw allows an attacker to bypass the intended read-only restrictions on the SFTP server, enabling them to create files in a directory that should be restricted to read-only operations.

```
P3 - MEDIUM, Components with Known Vulnerabilities - NMap, 3.73.90.150,
CVE-2017-15906 5.3 https://vulners.com/cve/CVE-2017-15906
```

ATTACK SCENARIO:

Imagine an attacker, Alic, seeks to exploit CVE-2017-15906 to bypass security controls on a company's SFTP server. Alic sets up an environment with the following steps:

1. **Set Up Read-Only Directory:**

Alice configures the SFTP server’s directory to be read-only by setting permissions to 555.

2. **Test for Exploitation:**

Alice attempts to create zero-length files in the read-only directory using SFTP commands:
sftp> put /dev/null zero_length_file.txt

If the server allows this operation, it confirms the presence of the vulnerability, as it should not permit file creation in a read-only directory.

3. **Exploit Application:**

Alice could use this vulnerability to create files that could potentially disrupt file management practices, such as filling up disk space or bypassing security mechanisms intended to prevent unauthorized file operations.

MITIGATION STRATEGIES:

- **Upgrade OpenSSH:** Update to OpenSSH version 7.6 or later, where this vulnerability has been addressed and proper handling of read-only modes is enforced.
- **Review Directory Permissions:** Regularly audit directory permissions and ensure that read-only directories are properly configured and enforced.
- **Implement Access Controls:** Use more granular access controls and policies to ensure that directories intended to be rad-only are strictly maintained as such.
- **Monitor and Log:** Monitor and log file operations on SFTP servers to detect any unusual activity that could indicate an attempt to exploit this or similar vulnerabilities.

Note:

Testing for CVE-2017-15906 was conducted, but the expected results of write operations in a read-only directory were not observed. The directory was set to read-only with correct permissions, yet file uploads were not successfully completed.

SSH Weak Key Exchange Algorithms Enabled

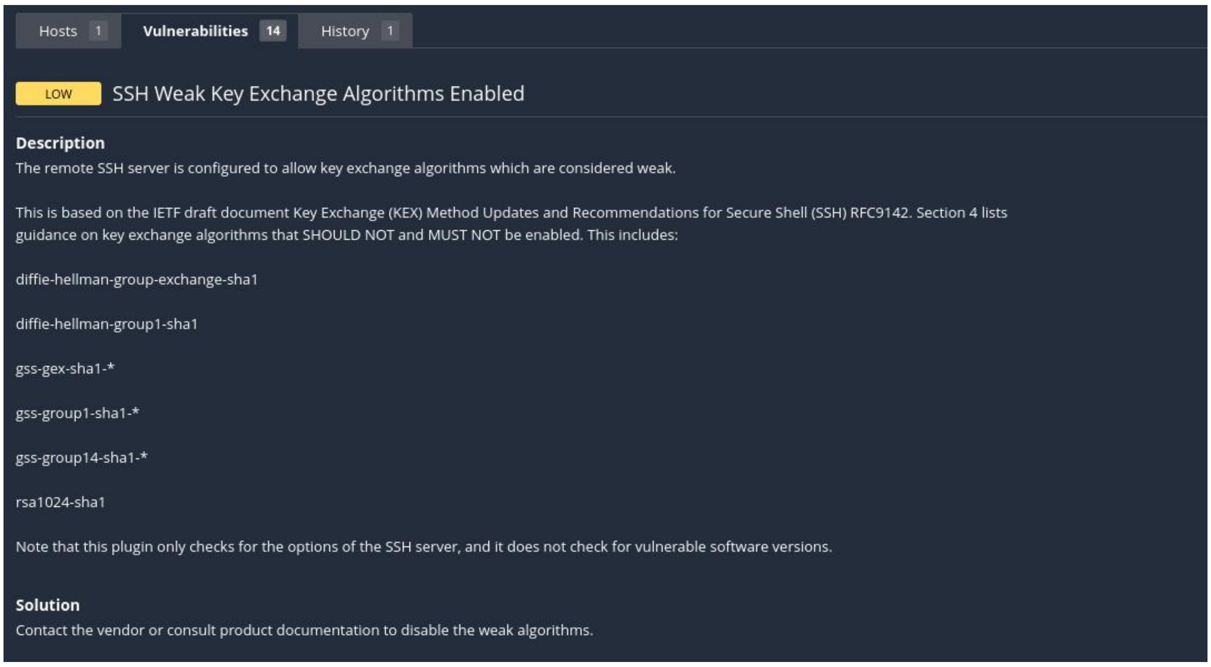
SEVERITY: LOW

ISSUE DESCRIPTION:

Weak key exchange algorithms in SSH can compromise the security of the SSH connection by allowing attackers to perform cryptographic attacks, such as exploiting weak algorithms to derive encryption keys. If an SSH server is configured to support outdated or vulnerable key exchange algorithms, it increases the risk of attackers intercepting or decrypting the data transmitted between the client and server.

PROOF OF VULNERABILITY:

A Nessus scan of the target system identified weak key exchange algorithms enabled on the SSH server. The scan results, detailed in the attached screenshot, list the specific weak algorithms supported, which are vulnerable to known cryptographic attacks.



SSH Weak Key Exchange Algorithms Enabled

Description
The remote SSH server is configured to allow key exchange algorithms which are considered weak.

This is based on the IETF draft document Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH) RFC9142. Section 4 lists guidance on key exchange algorithms that SHOULD NOT and MUST NOT be enabled. This includes:

diffie-hellman-group-exchange-sha1

diffie-hellman-group1-sha1

gss-gex-sha1-*

gss-group1-sha1-*

gss-group14-sha1-*

rsa1024-sha1

Note that this plugin only checks for the options of the SSH server, and it does not check for vulnerable software versions.

Solution
Contact the vendor or consult product documentation to disable the weak algorithms.

This screenshot from the Nessus scan highlights the detection of weak key exchange algorithms and the associated risk.

MITIGATION STRATEGIES:

- **Update SSH Configuration**: Disable weak or deprecated key exchange algorithms in the SSH server configuration file (sshd_config). Ensure only strong, modern algorithms are enabled, such as curve25519-sha256 or diffie-hellman-group14-sha256.

- **Upgrade OpenSSH**: Use the latest version of OpenSSH, which includes improvements and security enhancements to key exchange mechanisms.

SSH Server CBC Ciphers Enabled

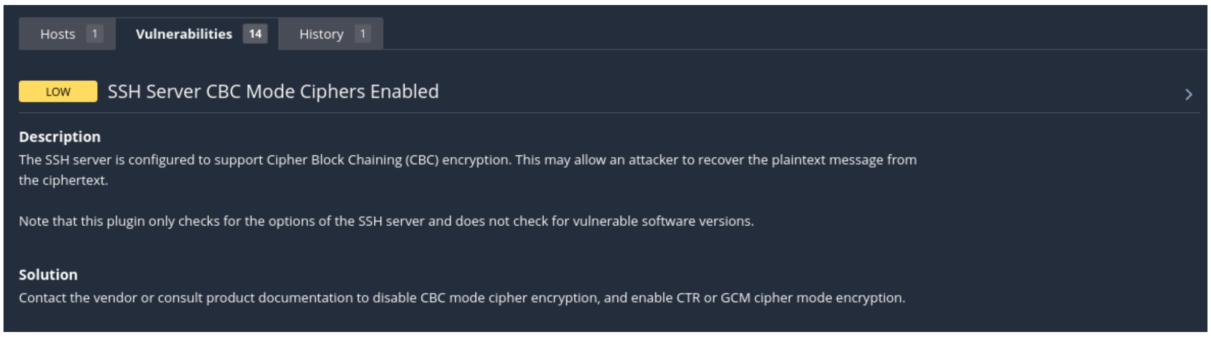
SEVERITY: LOW

ISSUE DESCRIPTION:

Cipher Block Chaining (CBC) mode ciphers in SSH have known vulnerabilities, such as susceptibility to padding oracle attacks. If CBC ciphers are enabled on an SSH server, it may expose the communication to cryptographic attacks that can potentially decrypt or manipulate the data being transmitted. This makes the SSH connection less secure and more vulnerable to interception.

PROOF OF VULNERABILITY:

The Nessus scan also detected that CBC ciphers are enabled on the SSH server. This configuration poses a risk of vulnerability to padding oracle attacks. The screenshot from Nessus provides details on the enabled CBC ciphers, indicating potential weaknesses in the encryption configuration.



SSH Server CBC Mode Ciphers Enabled

Description

The SSH server is configured to support Cipher Block Chaining (CBC) encryption. This may allow an attacker to recover the plaintext message from the ciphertext.

Note that this plugin only checks for the options of the SSH server and does not check for vulnerable software versions.

Solution

Contact the vendor or consult product documentation to disable CBC mode cipher encryption, and enable CTR or GCM cipher mode encryption.

```
The following client-to-server Cipher Block Chaining (CBC) algorithms
are supported :

  3des-cbc
  aes128-cbc
  aes192-cbc
  aes256-cbc
  blowfish-cbc
  cast128-cbc

The following server-to-client Cipher Block Chaining (CBC) algorithms
are supported :
```

3des-cbc
aes128-cbc
aes192-cbc
aes256-cbc
blowfish-cbc
cast128-cbc

MITIGATION STRATEGIES:

- **Update SSH Configuration**: Disable CBC ciphers in the SSH server configuration file (sshd_config). Replace them with more secure cipher modes, such as those based on Galois/Counter Mode (GCM) or ChaCha20. Recommended ciphers include aes256-gcm@openssh.com and chacha20-poly1305@openssh.com.
- **Upgrade OpenSSH**: Ensure the SSH server is updated to the latest version of OpenSSH, which defaults to stronger cipher suites and deprecates weaker ones.

2.4. RECOMMENDATIONS:

Implementation of Strong Access Controls

- **Access Control Testing**: Conduct thorough testing of access controls to ensure they are properly implemented and enforced. Regular testing should include checks for unauthorized access and privilege escalation.
- **Role-Based Access Control (RBAC)**: Develop and implement a role-based authentication and access control system. This system should define user roles and permissions, ensuring users have access only to the resources necessary for their roles. RBAC helps in managing and auditing access rights more efficiently.
- **Least Privilege**: Ensure that users have the minimum necessary permissions to perform their functions. Implement a least privilege model that restricts users to only the access needed for their specific job functions.

Routine Backups and Data Recovery Planning

- **Regular Backups**: Implement a regular backup schedule for critical data to prevent data loss in the event of a security breach or system failure.
- **Recovery Plan**: Develop and maintain a data recovery plan that can be quickly implemented if a data loss incident occurs.

Logging and Monitoring

- **Comprehensive Logging**: Implement comprehensive logging practices to capture all critical security events.
- **Alerting and Monitoring**: Set up real-time alerts for suspicious activities and regularly review logs to detect potential security incidents.

Establishing Incident Response Procedures

- **Response Plan**: Develop a comprehensive incident response plan that outlines steps to be taken in the event of a security breach.
- **Response Team**: Form an incident response team with clearly defined roles and responsibilities.

2.4. CONCLUSION:

We have completed the penetration testing of the Kyra application, owned by Data Tricks. This testing was conducted based on the most current technologies and known threats as of the date of this report. All discovered security issues have been thoroughly analyzed and are documented in this report.
As technologies and risks continue to evolve, the vulnerabilities associated with systems like Kyra, along with the necessary actions to mitigate these risks, will also change over time. Regular security reviews and updates are essential to maintaining a secure application environment.