Universidad Carlos III

Security Assessment Findings Report

Made for Universidad Carlos III

By jcasado Beyond IT Security

UNCONTROLLED PRINT WHEN COPIED

Business Confidential

- Date of the project: 2023-08-19
- Project name: Demo for Master Thesis (TFM)

TABLE OF CONTENTS

CONFIDENTIALITY STATEMENT

This document is the exclusive property of Universidad Carlos III and jcasado Beyond IT Security (jcasado).

This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Universidad Carlos III and jcasado).

Universidad Carlos III may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

DISCLAIMER

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. jcasado prioritized the assessment to identify the weakest security controls an attacker would exploit. jcasado recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

CONTACT INFORMATION

Name	Title	Contact Information
Universidad Carlos III		
Alfonso Sanchez Macian	Director	Email:
jcasado Beyond IT Security		
Jacobo Casado de Gracia	Penetration Tester	Email: casadojacobo@gmail.com

ASSESMENT OVERVIEW

From to 2023-08-19, Universidad Carlos III engaged jcasado Beyond IT Security, from now on, jcasado, to evaluate the security posture of its infrastructure compared to current industry best practices that included an internal network penetration test. All testing performed is based on the NIST SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customized testing frameworks.

Phases of penetration testing activities include the following:

- **Planning** – Customer goals are gathered and rules of engagement obtained.
- **Discovery** – Perform **scanning** and **enumeration** to identify potential vulnerabilities, weak areas, and exploits.
- **Attack** – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- **Reporting** – **Document** all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

ASSESSMENT COMPONENTS

Internal Penetration Test

An internal penetration test emulates the role of an attacker from inside the network. An engineer will scan the network to identify potential host vulnerabilities and perform common and advanced internal network attacks, such as: LLMNR/NBT-NS poisoning and other man- in-the-middle attacks, token impersonation, kerberoasting, pass-the-hash, golden ticket, and more. The engineer will seek to gain access to hosts through lateral movement, compromise domain user and admin accounts, and exfiltrate sensitive data.

FINDING SEVERITY RATINGS

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0–10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0 – 8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0 – 6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1 – 3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A (Informational data)	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

RISK FACTORS

Risk is measured by two factors: Likelihood and impact.

Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

Impact

Impact measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

SCOPE

Hosts analyzed

Host	Information about the host			
	Ports	Service	Version	CVEs
192.168.56.101	21 22 139 445 3306 4848 8080 8282	ftp ssh netbios-ssn microsoft-ds mysql http http http	Unknown 7.1 Unknown Unknown 5.5.20-log 4.0 4.0 1.1	CVE-2016-8858 CVE-2016-6515 CVE-2016-1908 CVE-2016-10009 CVE-2016-10012 CVE-2015-8325 CVE-2016-10010 CVE-2019-6111 CVE-2016-3115 CVE-2022-48437 CVE-2018-15919 CVE-2018-15473 CVE-2017-15906 CVE-2016-1907 CVE-2016-10708 CVE-2016-0778 CVE-2021-41617 CVE-2023-29323 CVE-2020-14145 CVE-2016-6210 CVE-2019-6110 CVE-2019-6109 CVE-2016-0777 CVE-2018-20685 CVE-2016-10011 CVE-2017-0144
192.168.203.129	21 22 80 139 445 3306	ftp ssh http netbios-ssn netbios-ssn mysql	2.3.4 4.7p1 Debian 8ubuntu1 2.2.8 3.X - 4.X	CVE-2010-4478 CVE-2008-1657 CVE-2010-5107 CVE-2012-0814 CVE-2011-5000

			3.X - 4.X 5.0.51a-3ubuntu5	CVE-2008-5161 CVE-2011-4327 CVE-2008-3259 CVE-2011-3192 CVE-2017-7679 CVE-2017-3167 CVE-2009-1891 CVE-2009-1890 CVE-2012-0883 CVE-2016-5387 CVE-2014-0226 CVE-2017-9788 CVE-2009-1956 CVE-2009-3555 CVE-2013-1862 CVE-2017-9798 CVE-2016-8743 CVE-2014-0231 CVE-2014-0098 CVE-2013-6438 CVE-2013-5704 CVE-2011-3368 CVE-2010-1452 CVE-2010-0408 CVE-2009-3095 CVE-2009-2699 CVE-2008-2364 CVE-2007-6750 CVE-2009-1195 CVE-2012-0031 CVE-2011-3607 CVE-2016-4975 CVE-2014-0118 CVE-2013-1896 CVE-2012-4558 CVE-2012-3499 CVE-2012-0053 CVE-2011-4317 CVE-2011-3639 CVE-2011-0419 CVE-2010-0434 CVE-2008-2939 CVE-2008-0455 CVE-2008-0005 CVE-2012-2687

				CVE-2009-3094 CVE-2008-0456 CVE-2011-4415 CVE-2017-0144 CVE-2009-2446 CVE-2008-0226 CVE-2009-5026 CVE-2009-4028 CVE-2010-1848 CVE-2010-1850 CVE-2008-7247 CVE-2010-3833 CVE-2010-1849 CVE-2008-4098 CVE-2008-2079 CVE-2012-0490 CVE-2012-0484 CVE-2012-0102 CVE-2012-0101 CVE-2012-0087 CVE-2010-3838 CVE-2010-3837 CVE-2010-3836 CVE-2010-3834 CVE-2010-3682 CVE-2010-3677 CVE-2009-4019 CVE-2008-3963 CVE-2010-1626 CVE-2012-0114 CVE-2012-0075
192.168.203.135	21 22 445 3306 8080	ftp ssh netbios-ssn mysql http	1.3.5 6.6.1p1 Ubuntu 2ubuntu2.13 3.X - 4.X Unknown 8.1.7.v20120910	CVE-2015-3306 CVE-2020-9272 CVE-2019-19272 CVE-2019-19271 CVE-2019-19270 CVE-2019-18217 CVE-2016-3125 CVE-2013-4359 CVE-2017-7418 CVE-2021-46854 CVE-2015-5600 CVE-2015-6564 CVE-2018-15919

				CVE-2021-41617 CVE-2020-14145 CVE-2015-5352 CVE-2015-6563

Scope Exclusions

Per client request, jcasado did not perform any of the following attacks during testing:

- Denial of service (DoS)
- Phishing/Social Engineering

All other attacks not specified above were permitted by Universidad Carlos III.

Client Allowances

Universidad Carlos III provided jcasado the following allowances:

- Internal access to network via TBD.

EXECUTIVE SUMMARY

jcasado evaluated Universidad Carlos III's internal security posture through penetration testing from to 2023-08-19. The following sections provide a high-level overview of vulnerabilities discovered, successful and unsuccessful attempts, and strengths and weaknesses.

This comprehensive executive summary amalgamates findings from three separate penetration testing assessments, providing a holistic view of the organization's security posture. The assessments probed into different IPs, scrutinizing open ports, vulnerabilities (CVEs), health status of services, and offer practical recommendations for mitigating the identified risks.

1. Description of **Found IPs and Ports:**

a. IP **192.168.203.135:**

Open Ports: 21 (FTP), 22 (SSH), 445 (NetBIOS-SSN), 3306 (MySQL), 8080 (HTTP)

b. IP **192.168.56.101:**

Open Ports: 21 (FTP), 22 (SSH), 139 (NetBIOS-SSN), 445 (Microsoft-DS), 3306 (MySQL), 4848 (HTTP), 8080 (HTTP), 8282 (HTTP)

c. IP **192.168.203.129:**

Open Ports: 21 (FTP), 22 (SSH), 80 (HTTP), 443 (HTTPS), 445 (NetBIOS-SSN), 3306 (MySQL)

2**. Analysis of Vulnerabilities (CVEs):**

Across the IPs, high-severity vulnerabilities were detected. These vulnerabilities encompass various risks, from potential service disruption to unauthorized access, underlining the need for action.

3. **Health Posture of Each Service:**

Assessing the health of each service identified significant vulnerabilities. These vulnerabilities, particularly in critical services like SSH and NetBIOS-SSN, demand swift attention for safeguarding against potential attacks.

4. **Recommendations for Mitigating Vulnerabilities:**

Addressing these vulnerabilities necessitates proactive measures:

- Regularly updating software to keep pace with evolving threats.
- Applying multi-factor authentication for secure access.
- Employing intrusion detection to flag suspicious activities.
- Restricting access to sensitive functions, thus limiting potential damage.
- Regularly auditing and mitigating vulnerabilities to ensure robust defense.
- Considering discontinuation or securing of unnecessary services.
- Employing network segmentation to isolate critical systems.
- Establishing a responsive incident management plan.
- Promoting security awareness among staff to foster a vigilant culture.

5. Conclusion:

These assessments shed light on critical vulnerabilities in the organization's security setup. By taking timely action and adhering to recommendations, the organization can significantly bolster its defense against potential security breaches, safeguard sensitive data, and foster a culture of cybersecurity awareness. A proactive approach, combined with continuous monitoring, will go a long way in reinforcing the organization's resilience in the face of evolving threats.
By implementing these recommendations, the organization can significantly improve its security posture and reduce the risk of unauthorized access, data breaches, and system disruptions. It is essential to prioritize the remediation of the identified vulnerabilities based on their severity and potential impact on the organization's systems and data.

Scoping and time limitations

Scoping during the engagement did not permit denial of service or social engineering across all testing components.

Time limitations were in place for testing. Internal network penetration testing was permitted for days.

Testing summary

The network assessment evaluated Universidad Carlos III's internal security posture. From an internal perspective, the jcasado performed vulnerability scanning against the IP addresses provided by Universidad Carlos III to evaluate the overall patching health of the network.

TBD

Tester Notes and Recommendations

TBD overall security (good/regular/bad).

TBD constants that stood out in the process

TBD reccomendations.

We recommend that the Universidad Carlos III team reviews the patching recommendations made in the TBD section of the report along with reviewing the provided scans for a full overview of the items to be patched. We also recommend that Demo corp improve their patch management policies and procedures to help prevent potential attacks within their network.

TBD alerts triggered.

Overall, the Universidad Carlos III network performed as expected for the penetration test. We recommend that the Universidad Carlos III team thoroughly review the recommendations made in this report, patch the findings, and re-test annually to improve their overall security posture.

VULNERABILITY SUMMARY & REPORT CARD

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Internal Penetration Test Findings

4	20	56	18	0
Critical	**High**	**Moderate**	**Low**	**Informational**

Finding 1

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2010-4478**
- Severity score: **7.5** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in OpenSSH 5.6 and earlier, when J-PAKE is enabled, has been classified as "High" severity due to its potential impact on the organization's security. A high severity vulnerability indicates a significant risk to the organization's systems and data.

The vulnerability allows remote attackers to bypass the need for knowledge of the shared secret and successfully authenticate by sending crafted values in each round of the J-PAKE protocol. This means that an attacker can exploit this vulnerability to gain unauthorized access to the organization's systems and potentially compromise sensitive data.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can bypass the authentication mechanism and authenticate as a legitimate user without knowledge of the shared secret. This can lead to unauthorized access to the organization's systems, allowing the attacker to perform malicious activities such as stealing or modifying sensitive data, installing malware, or disrupting critical services.

The severity is also influenced by the ease of exploitation. The vulnerability identified in this report can be exploited remotely, meaning that an attacker does not require physical access to the organization's network. This increases the risk of opportunistic attacks and targeted attacks by sophisticated threat actors.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in unauthorized access, data breaches, financial losses, reputational damage, and potential legal consequences. It is crucial for the organization to prioritize the patching or upgrading of affected OpenSSH versions to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

OpenSSH 5.6 and earlier, when J-PAKE is enabled, does not properly validate the public parameters in the J-PAKE protocol, which allows remote attackers to bypass the need for knowledge of the shared secret, and successfully authenticate, by sending crafted values in each round of the protocol, a related issue to CVE-2010-4252.

Mitigation steps

Vulnerability: OpenSSH J-PAKE Protocol Validation Bypass

Description:
The OpenSSH version 5.6 and earlier, when J-PAKE (Password Authenticated Key Exchange by Juggling) is enabled, does not properly validate the public parameters in the J-PAKE protocol. This allows remote attackers to bypass the need for knowledge of the shared secret and successfully authenticate by sending crafted values in each round of the protocol. This vulnerability is related to CVE-2010-4252.

Mitigation Steps:
To mitigate the risk of OpenSSH J-PAKE protocol validation bypass, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 5.6.

2. Disable J-PAKE: If J-PAKE is not required for your environment, consider disabling it. This can be done by modifying the OpenSSH configuration file (typically located at /etc/ssh/sshd_config) and removing or commenting out the line that enables J-PAKE. The line to be modified is usually "KexAlgorithms".

3. Enable Strict Mode: Enable strict mode in the OpenSSH configuration to enforce stronger security settings. This can be done by adding the following line to the sshd_config file:

```
StrictModes yes
```

Strict mode ensures that various file and directory permissions are set correctly, reducing the risk of privilege escalation attacks.

4. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to bypass authentication mechanisms.

5. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential attacks.

Example Command:
To disable J-PAKE in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that enables J-PAKE. It may look like this:

```
KexAlgorithms diffie-hellman-group-exchange-sha256,j-PAKE
```

3. Comment out or remove the line by adding a "#" at the beginning:

```
# KexAlgorithms diffie-hellman-group-exchange-sha256,j-PAKE
```

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 2

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2008-1657**
- Severity score: **6.5** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in OpenSSH 4.4 up to versions before 4.9 has been classified as "Medium" severity due to its potential impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability allows remote authenticated users to bypass the sshd_config ForceCommand directive by modifying the .ssh/rc session file. This means that an attacker with authenticated access to the organization's systems can manipulate the session file to execute arbitrary commands, potentially compromising the security and integrity of the system.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can bypass the intended restrictions imposed by the ForceCommand directive. This can allow the attacker to execute unauthorized commands, potentially leading to unauthorized access, data manipulation, or privilege escalation.

The severity is also influenced by the requirement for authenticated access. While the vulnerability requires the attacker to have authenticated access to the system, it still poses a significant risk as it allows an authenticated user to bypass security measures and gain unauthorized privileges.

However, the severity is mitigated by the fact that the vulnerability requires the attacker to have access to modify the .ssh/rc session file. This means that the attacker needs to have a certain level of access to the system, limiting the potential scope of exploitation.

In summary, the medium severity of this vulnerability highlights the need for the organization to address and remediate the issue. While the risk is not as severe as a critical or high severity vulnerability, it still poses a moderate risk to the organization's systems and data. It is important for the organization to ensure that proper access controls and monitoring mechanisms are in place to prevent unauthorized modification of session files and to regularly update OpenSSH to the latest version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

OpenSSH 4.4 up to versions before 4.9 allows remote authenticated users to bypass the sshd_config ForceCommand directive by modifying the .ssh/rc session file.

Mitigation steps

Vulnerability: OpenSSH ForceCommand Bypass via .ssh/rc Session File

Description:
OpenSSH versions 4.4 up to versions before 4.9 allow remote authenticated users to bypass the sshd_config ForceCommand directive by modifying the .ssh/rc session file. This vulnerability allows attackers to execute arbitrary commands on the remote system.

Mitigation Steps:
To mitigate the risk of OpenSSH ForceCommand bypass via the .ssh/rc session file, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 4.9.

2. Disable .ssh/rc Execution: Modify the OpenSSH configuration to disable the execution of the .ssh/rc session file. This can be done by adding or modifying the following line in the sshd_config file (typically located at /etc/ssh/sshd_config):

```
PermitUserRC no
```

This configuration option prevents the execution of the .ssh/rc file for all users.

3. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential attacks.

4. Limit User Privileges: Follow the principle of least privilege and ensure that users have only the necessary privileges to perform their tasks. Restricting user privileges can help mitigate the impact of a successful attack.

5. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to bypass security mechanisms.

Example Command:
To disable the execution of the .ssh/rc session file in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that controls the execution of the .ssh/rc file. It may look like this:

```
PermitUserRC yes
```

3. Change the value to "no" to disable the execution:

```
PermitUserRC no
```

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 3

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2010-5107**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the default configuration of OpenSSH through 6.1 has been classified as "Medium" severity due to its potential impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and operations.

The vulnerability is related to the enforcement of a fixed time limit between establishing a TCP connection and completing a login in OpenSSH. This can be exploited by remote attackers to cause a denial of service (connection-slot exhaustion) by periodically making many new TCP connections.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can exhaust the available connection slots, preventing legitimate users from establishing new connections to the OpenSSH server. This can result in a denial of service (DoS) condition, impacting the availability and accessibility of the organization's systems.

The severity is also influenced by the ease of exploitation. The vulnerability identified in this report can be exploited remotely, meaning that an attacker does not require physical access to the organization's network. This increases the risk of opportunistic attacks and targeted attacks by malicious actors.

However, the severity is mitigated by the fact that the vulnerability requires the attacker to periodically make many new TCP connections. This means that the attacker needs to have the capability to generate a high volume of connection attempts, which may limit the potential impact and scope of exploitation.

In summary, the medium severity of this vulnerability highlights the need for the organization to address and remediate the issue. While the risk is not as severe as a critical or high severity vulnerability, it still poses a moderate risk to the availability and accessibility of the organization's systems. It is important for the organization to review and adjust the default configuration of OpenSSH to mitigate the risk of connection-slot exhaustion and implement appropriate network monitoring and access controls to detect and prevent such DoS attacks.

Public exploits related to this finding

Description of the finding

The default configuration of OpenSSH through 6.1 enforces a fixed time limit between establishing a TCP connection and completing a login, which makes it easier for remote attackers to cause a denial of service (connection-slot exhaustion) by periodically making many new TCP connections.

Mitigation steps

Vulnerability: OpenSSH Denial of Service (DoS) via Connection-Slot Exhaustion

Description:
The default configuration of OpenSSH through version 6.1 enforces a fixed time limit between establishing a TCP connection and completing a login. This can make it easier for remote attackers to cause a denial of service (DoS) by periodically making many new TCP connections, leading to connection-slot exhaustion.

Mitigation Steps:
To mitigate the risk of OpenSSH denial of service (DoS) via connection-slot exhaustion, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 6.1.

2. Configure Connection Limits: Modify the OpenSSH configuration to limit the number of concurrent connections allowed. This can be done by adding or modifying the following line in the sshd_config file (typically located at /etc/ssh/sshd_config):

```
MaxSessions <number>
```

Replace `<number>` with the maximum number of concurrent sessions you want to allow. This helps prevent connection-slot exhaustion by limiting the number of connections that can be established.

3. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which new TCP connections can be established. This can help prevent attackers from overwhelming the system with a large number of connections.

4. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential DoS attacks.

5. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

Example Command:
To configure a maximum number of concurrent sessions in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that controls the maximum number of sessions. It may look like this:

```
#MaxSessions 10
```

3. Uncomment the line by removing the "#" at the beginning, and set the desired maximum number of sessions:

```
MaxSessions 50
```

In this example, the maximum number of concurrent sessions is set to 50.

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 4

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2012-0814**
- Severity score: **3.5** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the auth_parse_options function in sshd in OpenSSH before 5.7 has been classified as "Low" severity due to its limited impact on the organization's security. A low severity vulnerability indicates a relatively low risk to the organization's systems and data.

The vulnerability allows remote authenticated users to obtain potentially sensitive information by reading debug messages containing authorized_keys command options. This means that an authenticated user can access debug messages that may contain information about authorized_keys command options, potentially revealing sensitive configuration details.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they may gain access to authorized_keys command options, which could potentially reveal information about the configuration of the OpenSSH server. However, the impact of this information disclosure is limited, as it does not directly provide access to sensitive data or allow for unauthorized access to the system.

The severity is also influenced by the requirement for remote authenticated access. The vulnerability requires the attacker to have authenticated access to the system, limiting the potential for opportunistic attacks. Additionally, the vulnerability does not allow for privilege escalation or crossing privilege boundaries, as it only exposes information within the context of the authenticated user.

However, it is worth noting that in certain scenarios where a user account intentionally has no shell or filesystem access, this vulnerability may allow the user to obtain information that they would not have otherwise been able to access. This could potentially cross privilege boundaries in specific cases.

In summary, the low severity of this vulnerability indicates a relatively low risk to the organization's systems and data. While the information disclosure may reveal some sensitive configuration details, it does not directly lead to unauthorized access or compromise of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of OpenSSH that includes the fix for this vulnerability to prevent potential information disclosure.

Public exploits related to this finding

Description of the finding

The auth_parse_options function in auth-options.c in sshd in OpenSSH before 5.7 provides debug messages containing authorized_keys command options, which allows remote authenticated users to obtain potentially sensitive information by reading these messages, as demonstrated by the shared user account required by Gitolite. NOTE: this can cross privilege boundaries because a user account may intentionally have no shell or filesystem access, and therefore may have no supported way to read an authorized_keys file in its own home directory.

Mitigation steps

Vulnerability: OpenSSH Debug Messages Information Disclosure

Description:
The auth_parse_options function in auth-options.c in sshd in OpenSSH before version 5.7 provides debug messages that contain authorized_keys command options. This allows remote authenticated users to obtain potentially sensitive information by reading these messages. This vulnerability can be exploited by attackers with remote access to the system, potentially leading to the disclosure of sensitive information.

Mitigation Steps:
To mitigate the risk of OpenSSH debug messages information disclosure, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 5.7.

2. Disable Debug Messages: Modify the OpenSSH configuration to disable the output of debug messages. This can be done by adding or modifying the following line in the sshd_config file (typically located at /etc/ssh/sshd_config):

```
LogLevel INFO
```

Setting the log level to "INFO" will disable debug messages while still providing necessary logging information.

3. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or disclose sensitive information.

4. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential information disclosure incidents.

5. Limit User Privileges: Follow the principle of least privilege and ensure that users have only the necessary privileges to perform their tasks. Restricting user privileges can help mitigate the impact of a successful attack.

Example Command:
To disable debug messages in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that controls the log level. It may look like this:

```
#LogLevel INFO
```

3. Uncomment the line by removing the "#" at the beginning, and set the log level to "INFO":

```
LogLevel INFO
```

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 5

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2011-5000**
- Severity score: **3.5** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the ssh_gssapi_parse_ename function in gss-serv.c in OpenSSH 5.8 and earlier has been classified as "Low" severity due to its limited impact on the organization's security. A low severity vulnerability indicates a relatively low risk to the organization's systems and data.

The vulnerability occurs when gssapi-with-mic authentication is enabled and allows remote authenticated users to cause a denial of service (memory consumption) by providing a large value in a certain length field.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a denial of service by consuming excessive memory resources. However, the impact of this denial of service is limited, as it does not result in unauthorized access, data compromise, or disruption of critical services.

The severity is also influenced by the limited scenarios in which this issue is relevant. The vulnerability requires the specific use of gssapi-with-mic authentication and the ability to provide a large value in a specific length field. This narrows the potential scope of exploitation and reduces the likelihood of opportunistic attacks.

It is worth noting that while the impact of this vulnerability is low, denial of service attacks can still have some operational impact on the organization. The excessive consumption of memory resources may result in degraded performance or unavailability of the affected OpenSSH service.

In summary, the low severity of this vulnerability indicates a relatively low risk to the organization's systems and data. While the denial of service may cause some operational inconvenience, it does not directly lead to unauthorized access or compromise of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of OpenSSH that includes the fix for this vulnerability to prevent potential denial of service attacks.

Public exploits related to this finding

Description of the finding

The ssh_gssapi_parse_ename function in gss-serv.c in OpenSSH 5.8 and earlier, when gssapi-with-mic authentication is enabled, allows remote authenticated users to cause a denial of service (memory

consumption) via a large value in a certain length field. NOTE: there may be limited scenarios in which this issue is relevant.

Mitigation steps

Vulnerability: OpenSSH Denial of Service (DoS) via Memory Consumption

Description:
The ssh_gssapi_parse_ename function in gss-serv.c in OpenSSH versions 5.8 and earlier, when gssapi-with-mic authentication is enabled, allows remote authenticated users to cause a denial of service (memory consumption) by providing a large value in a certain length field. This vulnerability can lead to excessive memory usage, potentially resulting in system instability or unresponsiveness.

Mitigation Steps:
To mitigate the risk of OpenSSH denial of service (DoS) via memory consumption, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 5.8.

2. Disable GSSAPI Authentication: If GSSAPI authentication is not required for your environment, consider disabling it. This can be done by modifying the OpenSSH configuration file (typically located at /etc/ssh/sshd_config) and removing or commenting out the line that enables GSSAPI authentication. The line to be modified is usually "GSSAPIAuthentication".

3. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which new SSH connections can be established. This can help prevent attackers from overwhelming the system with a large number of connections.

4. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential DoS attacks.

5. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

Example Command:
To disable GSSAPI authentication in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

```
```

2. Locate the line that enables GSSAPI authentication. It may look like this:

```
GSSAPIAuthentication yes
```

3. Comment out or remove the line by adding a "#" at the beginning:

```
# GSSAPIAuthentication yes
```

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 6

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2008-5161**
- Severity score: **2.6** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the error handling of the SSH protocol in various SSH implementations, including SSH Tectia, ConnectSecure, Server for Linux on IBM System z, Server for IBM z/OS, and OpenSSH, has been classified as "Low" severity due to its limited impact on the organization's security. A low severity vulnerability indicates a relatively low risk to the organization's systems and data.

The vulnerability occurs when using a block cipher algorithm in Cipher Block Chaining (CBC) mode and makes it easier for remote attackers to recover certain plaintext data from an arbitrary block of ciphertext in an SSH session. However, the specific vectors and techniques used to exploit this vulnerability are unknown.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they may be able to recover certain plaintext data from an SSH session. However, the impact of this data recovery is limited, as it does not directly result in unauthorized access, data compromise, or disruption of critical services.

The severity is also influenced by the limited information available about the vectors and techniques used to exploit this vulnerability. Without specific details, it is difficult to assess the likelihood and ease of exploitation, which further reduces the risk associated with this vulnerability.

It is worth noting that the vulnerability affects multiple SSH implementations, which may increase the potential attack surface. However, the low severity indicates that the risk is still relatively low.

In summary, the low severity of this vulnerability indicates a relatively low risk to the organization's systems and data. While the data recovery may have some privacy implications, it does not directly lead to unauthorized access or compromise of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of the affected SSH implementation that includes the fix for this vulnerability to prevent potential data recovery attacks.

Public exploits related to this finding

Description of the finding

Error handling in the SSH protocol in (1) SSH Tectia Client and Server and Connector 4.0 through 4.4.11, 5.0 through 5.2.4, and 5.3 through 5.3.8; Client and Server and ConnectSecure 6.0 through 6.0.4; Server for Linux on IBM System z 6.0.4; Server for IBM z/OS 5.5.1 and earlier, 6.0.0, and 6.0.1; and Client 4.0-J through 4.3.3-J and 4.0-K through 4.3.10-K; and (2) OpenSSH 4.7p1 and possibly other versions, when using a block cipher algorithm in Cipher Block Chaining (CBC) mode, makes it easier for remote attackers to recover certain plaintext data from an arbitrary block of ciphertext in an SSH session via unknown vectors.

Mitigation steps

Vulnerability: SSH Protocol Plaintext Recovery in CBC Mode

Description:
Error handling in the SSH protocol in various SSH implementations, including SSH Tectia, ConnectSecure, Server for Linux on IBM System z, Server for IBM z/OS, and OpenSSH, when using a block cipher algorithm in Cipher Block Chaining (CBC) mode, can make it easier for remote attackers to recover certain plaintext data from an arbitrary block of ciphertext in an SSH session. This vulnerability can potentially lead to the disclosure of sensitive information.

Mitigation Steps:
To mitigate the risk of SSH protocol plaintext recovery in CBC mode, the following steps can be taken:

1. Update SSH Implementations: Upgrade to a version of the SSH implementation that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Use Counter Mode (CTR) or Galois/Counter Mode (GCM): If possible, switch to using counter mode (CTR) or Galois/Counter Mode (GCM) instead of Cipher Block Chaining (CBC) mode. These modes provide better security and are not vulnerable to the plaintext recovery attack.

3. Disable CBC Mode: If counter mode or GCM is not an option, consider disabling CBC mode in the SSH implementation. This can be done by modifying the SSH configuration file (e.g., /etc/ssh/sshd_config) and removing or commenting out the line that enables CBC mode. The line to be modified is usually "Ciphers".

4. Regularly Update and Patch Software: Keep the SSH implementation and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or disclose sensitive information.

5. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential information disclosure incidents.

Example Command:
To disable CBC mode in the SSH configuration, follow these steps:

1. Open the SSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that enables CBC mode. It may look like this:

```
Ciphers aes128-cbc,aes192-cbc,aes256-cbc
```

3. Comment out or remove the line by adding a "#" at the beginning:

```
# Ciphers aes128-cbc,aes192-cbc,aes256-cbc
```

4. Save the changes and exit the text editor.

5. Restart the SSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the SSH implementation and version being used. It is important to consult the documentation and best practices for your specific environment.

Finding 7

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2011-4327**
- Severity score: **2.1** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in ssh-keysign.c in ssh-keysign in OpenSSH before 5.8p2 has been classified as "Low" severity due to its limited impact on the organization's security. A low severity vulnerability indicates a relatively low risk to the organization's systems and data.

The vulnerability occurs on certain platforms and involves the execution of ssh-rand-helper with unintended open file descriptors. This can allow local users to obtain sensitive key information by leveraging the ptrace system call.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability, they may be able to obtain sensitive key information. However, the impact of this information disclosure is limited, as it requires local access and does not directly result in unauthorized access or compromise of critical systems.

The severity is also influenced by the limited platforms on which this vulnerability occurs. This narrows the potential scope of exploitation and reduces the likelihood of opportunistic attacks.

It is worth noting that the vulnerability involves the ptrace system call, which is a debugging and process control mechanism. Exploiting this vulnerability requires the local user to have the necessary privileges to use ptrace, which further limits the potential impact and reduces the risk associated with this vulnerability.

In summary, the low severity of this vulnerability indicates a relatively low risk to the organization's systems and data. While the information disclosure may have some privacy implications, it does not directly lead to unauthorized access or compromise of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of OpenSSH that includes the fix for this vulnerability to prevent potential information disclosure by local users.

Public exploits related to this finding

Description of the finding

ssh-keysign.c in ssh-keysign in OpenSSH before 5.8p2 on certain platforms executes ssh-rand-helper with unintended open file descriptors, which allows local users to obtain sensitive key information via the ptrace system call.

Mitigation steps

Vulnerability: OpenSSH ssh-keysign Information Disclosure via ptrace

Description:
The ssh-keysign program in OpenSSH before version 5.8p2, when executed on certain platforms, may unintentionally open file descriptors when executing ssh-rand-helper. This can allow local users to obtain sensitive key information by leveraging the ptrace system call. This vulnerability can potentially lead to the disclosure of sensitive key information.

Mitigation Steps:
To mitigate the risk of OpenSSH ssh-keysign information disclosure via ptrace, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 5.8p2.

2. Limit Access to ssh-keysign: Restrict access to the ssh-keysign program to trusted users only. This can be done by modifying the file permissions and ownership of the ssh-keysign binary. Ensure that only privileged users have execute permissions on the binary.

3. Monitor and Log System Activity: Implement a centralized logging system to monitor and log system activity, including the execution of privileged programs like ssh-keysign. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential information disclosure incidents.

4. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or disclose sensitive information.

5. Implement Least Privilege: Follow the principle of least privilege and ensure that users have only the necessary privileges to perform their tasks. Restricting user privileges can help mitigate the impact of a successful attack.

Example Command:
To restrict access to the ssh-keysign program, follow these steps:

1. Locate the ssh-keysign binary on your system. It is typically located in the /usr/libexec/openssh directory.

2. Set the appropriate file permissions and ownership to restrict access to the binary. For example, you can set the ownership to root and the permissions to 700:

```
sudo chown root:root /usr/libexec/openssh/ssh-keysign
sudo chmod 700 /usr/libexec/openssh/ssh-keysign
```

3. Verify that only privileged users have execute permissions on the ssh-keysign binary:

```
ls -l /usr/libexec/openssh/ssh-keysign
```

The output should show that only the owner (root) has execute permissions.

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 8

- Affected IP: **192.168.203.129**
- Affected port and service: **22** (service **ssh**)
- Related CVE: **CVE-2008-3259**
- Severity score: **1.2** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in OpenSSH before 5.1 has been classified as "Low" severity due to its limited impact on the organization's security. A low severity vulnerability indicates a relatively low risk to the organization's systems and data.

The vulnerability occurs when the X11UseLocalhost configuration setting is disabled, and OpenSSH sets the SO_REUSEADDR socket option. This can allow local users on some platforms, such as HP-UX, to hijack the X11 forwarding port by binding to a single IP address.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability, they may be able to hijack the X11 forwarding port. However, the impact of this hijacking is limited, as it requires local access and does not directly result in unauthorized access or compromise of critical systems.

The severity is also influenced by the limited platforms on which this vulnerability occurs. This narrows the potential scope of exploitation and reduces the likelihood of opportunistic attacks.

It is worth noting that the vulnerability specifically affects the X11 forwarding feature of OpenSSH and requires the X11UseLocalhost configuration setting to be disabled. This means that organizations that do not use X11 forwarding or have the X11UseLocalhost setting enabled are not affected by this vulnerability.

In summary, the low severity of this vulnerability indicates a relatively low risk to the organization's systems and data. While the hijacking of the X11 forwarding port may have some impact on the affected platform, it does not directly lead to unauthorized access or compromise of critical systems. It is still important for organizations that use X11 forwarding and have the X11UseLocalhost setting disabled to address and remediate the issue by updating to a version of OpenSSH that includes the fix for this vulnerability to prevent potential hijacking of the X11 forwarding port.

Public exploits related to this finding

Description of the finding

OpenSSH before 5.1 sets the SO_REUSEADDR socket option when the X11UseLocalhost configuration setting is disabled, which allows local users on some platforms to hijack the X11 forwarding port via a bind to a single IP address, as demonstrated on the HP-UX platform.

Mitigation steps

Vulnerability: OpenSSH X11 Forwarding Port Hijacking via SO_REUSEADDR

Description:
OpenSSH versions before 5.1, when the X11UseLocalhost configuration setting is disabled, set the SO_REUSEADDR socket option. This allows local users on certain platforms to hijack the X11 forwarding port by binding to a single IP address. This vulnerability has been demonstrated on the HP-UX platform. Exploiting this vulnerability can lead to unauthorized access to X11 forwarding sessions.

Mitigation Steps:
To mitigate the risk of OpenSSH X11 forwarding port hijacking via SO_REUSEADDR, the following steps can be taken:

1. Update OpenSSH: Upgrade to a version of OpenSSH that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.

2. Enable X11UseLocalhost: Modify the OpenSSH configuration to enable the X11UseLocalhost setting. This can be done by adding or modifying the following line in the sshd_config file (typically located at /etc/ssh/sshd_config):

```
X11UseLocalhost yes
```

Enabling this setting ensures that X11 forwarding is restricted to the localhost interface, preventing potential hijacking of the X11 forwarding port.

3. Regularly Update and Patch Software: Keep OpenSSH and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

4. Monitor and Log SSH Activity: Implement a centralized logging system to monitor and log SSH activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit User Privileges: Follow the principle of least privilege and ensure that users have only the

necessary privileges to perform their tasks. Restricting user privileges can help mitigate the impact of a successful attack.

Example Command:
To enable the X11UseLocalhost setting in the OpenSSH configuration, follow these steps:

1. Open the OpenSSH configuration file using a text editor. For example, using the nano editor:

```
sudo nano /etc/ssh/sshd_config
```

2. Locate the line that controls the X11UseLocalhost setting. It may look like this:

```
#X11UseLocalhost no
```

3. Uncomment the line by removing the "#" at the beginning, and set the value to "yes":

```
X11UseLocalhost yes
```

4. Save the changes and exit the text editor.

5. Restart the OpenSSH service for the changes to take effect:

```
sudo service ssh restart
```

Note: The specific steps may vary depending on the operating system and version of OpenSSH being used. It is important to consult the documentation and best practices for your specific environment.

Finding 9

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-3192**
- Severity score: **7.8** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the byterange filter in the Apache HTTP Server has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability allows remote attackers to cause a denial of service (memory and CPU consumption) by sending a Range header that expresses multiple overlapping ranges. This can lead to excessive memory and CPU usage, resulting in the unavailability or degradation of the Apache HTTP Server.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can consume significant system resources, leading to a denial of service condition. This can result in the unavailability of the web server, impacting the organization's online presence, services, and potentially causing financial losses.

The severity is also influenced by the fact that this vulnerability has been exploited in the wild in the past. This indicates that it is actively targeted by malicious actors, increasing the urgency for organizations to address and remediate the issue.

It is worth noting that this vulnerability is different from CVE-2007-0086, indicating that it is a distinct and separate vulnerability with its own unique impact.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in a significant disruption of services, financial losses, and reputational damage. It is crucial for the organization to update the affected versions of the Apache HTTP Server to a patched version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Exploit ID 17696.

- Description: Apache - Remote Memory Exhaustion (Denial of Service)
- Download URL: https://www.exploit-db.com/exploits/17696

Exploit ID 18221.

- Description: Apache - Denial of Service
- Download URL: https://www.exploit-db.com/exploits/18221

Description of the finding

The byterange filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges, as exploited in the wild in August 2011, a different vulnerability than CVE-2007-0086.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via Multiple Overlapping Ranges

Description:
The byterange filter in the Apache HTTP Server versions 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service (memory and CPU consumption) by sending a Range header that expresses multiple overlapping ranges. This vulnerability was exploited in the wild in August 2011. It is a different vulnerability than CVE-2007-0086.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server denial of service (DoS) via multiple overlapping ranges, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which requests with Range headers can be processed. This can help prevent attackers from overwhelming the server with multiple overlapping ranges.

4. Implement Web Application Firewalls (WAF): Deploy a WAF in front of the Apache HTTP Server to detect and block malicious requests that exploit this vulnerability. WAFs can provide an additional layer of protection by inspecting and filtering incoming traffic.

5. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

Example Command:
To configure rate limiting for requests with Range headers, you can use the Apache HTTP Server module called mod_reqtimeout. Here is an example configuration:

1. Open the Apache HTTP Server configuration file using a text editor. The location and name of the file may vary depending on your operating system and installation. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the section where modules are loaded. It may look like this:

```
LoadModule reqtimeout_module modules/mod_reqtimeout.so
```

If the line is commented out (starts with "#"), remove the "#" to enable the module.

3. Add the following configuration directives to enable rate limiting for requests with Range headers:

```
<IfModule mod_reqtimeout.c>
  RequestReadTimeout header=10-20,minrate=500
</IfModule>
```

This configuration sets a timeout of 10 to 20 seconds for processing headers, with a minimum data rate of 500 bytes per second. Adjust the values according to your specific requirements.

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 10

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2017-7679**
- Severity score: **7.5** (**CRITICAL** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26 has been classified as "Critical" severity due to its significant impact on the organization's security. A critical severity vulnerability indicates a severe risk to the organization's systems and data.

The vulnerability occurs in the mod_mime module and allows an attacker to read one byte past the end of a buffer when sending a malicious Content-Type response header. This can lead to memory corruption and potentially enable remote code execution or other malicious activities.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can manipulate the Content-Type response header to trigger a buffer overflow or other memory corruption issues. This can result in the execution of arbitrary code, unauthorized access, or compromise of the affected Apache httpd server.

The severity is also influenced by the fact that this vulnerability is critical in nature and can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by sophisticated threat actors.

It is worth noting that the vulnerability affects multiple versions of Apache httpd, indicating a potentially wide impact across various deployments.

In summary, the critical severity of this vulnerability highlights the urgent need for the organization to address and remediate the issue. Failure to do so can result in severe consequences, including unauthorized access, remote code execution, data breaches, financial losses, reputational damage, and potential legal liabilities. It is crucial for the organization to update the affected versions of Apache httpd to a patched version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, mod_mime can read one byte past the end of a buffer when sending a malicious Content-Type response header.

Mitigation steps

Vulnerability: Apache HTTP Server Buffer Overread in mod_mime

Description:
In Apache HTTP Server versions 2.2.x before 2.2.33 and 2.4.x before 2.4.26, the mod_mime module can read one byte past the end of a buffer when sending a malicious Content-Type response header. This vulnerability can potentially lead to information disclosure or remote code execution.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server buffer overread in mod_mime, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.33 or 2.4.26, depending on the version you are currently using.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Web Application Firewalls (WAF): Deploy a WAF in front of the Apache HTTP Server to detect and block malicious requests that exploit this vulnerability. WAFs can provide an additional layer of protection by inspecting and filtering incoming traffic.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

```
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 11

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2017-3167**
- Severity score: **7.5** (**CRITICAL** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26 has been classified as "Critical" severity due to its significant impact on the organization's security. A critical severity vulnerability indicates a severe risk to the organization's systems and data.

The vulnerability occurs when third-party modules use the ap_get_basic_auth_pw() function outside of the authentication phase. This can lead to the bypassing of authentication requirements, allowing unauthorized access to protected resources.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can bypass authentication requirements and gain unauthorized access to protected resources. This can result in the exposure of sensitive data, unauthorized modifications, or other malicious activities.

The severity is also influenced by the fact that this vulnerability is critical in nature and can be exploited by third-party modules. This means that even if the core Apache httpd server is properly configured and secured, the vulnerability can still be present if third-party modules are used improperly.

It is worth noting that the vulnerability affects multiple versions of Apache httpd, indicating a potentially wide impact across various deployments.

In summary, the critical severity of this vulnerability highlights the urgent need for the organization to address and remediate the issue. Failure to do so can result in severe consequences, including unauthorized access, data breaches, financial losses, reputational damage, and potential legal liabilities. It is crucial for the organization to review and ensure the proper usage of the ap_get_basic_auth_pw() function by third-party modules and update the affected versions of Apache httpd to a patched version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, use of the ap_get_basic_auth_pw() by third-party modules outside of the authentication phase may lead to authentication requirements being bypassed.

Mitigation steps

Vulnerability: Apache HTTP Server Authentication Bypass via ap_get_basic_auth_pw()

Description:
In Apache HTTP Server versions 2.2.x before 2.2.33 and 2.4.x before 2.4.26, the use of the ap_get_basic_auth_pw() function by third-party modules outside of the authentication phase may lead to authentication requirements being bypassed. This vulnerability can potentially allow unauthorized access to protected resources.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server authentication bypass via ap_get_basic_auth_pw(), the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.33 or 2.4.26, depending on the version you are currently using.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Third-Party Module Usage: Evaluate and limit the use of third-party modules in your Apache HTTP Server configuration. Only use trusted and well-maintained modules from reputable sources. Minimizing the use of third-party modules reduces the attack surface and potential vulnerabilities.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 12

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-1891**
- Severity score: **7.1** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_deflate module in Apache httpd 2.2.11 and earlier has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability occurs when the mod_deflate module compresses large files until completion, even after the associated network connection is closed. This can be exploited by remote attackers to cause a denial of service by consuming excessive CPU resources.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a denial of service by consuming significant CPU resources on the affected Apache httpd server. This can result in degraded performance, unresponsiveness, or even complete unavailability of the server, impacting the organization's online presence and services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_deflate module, which is responsible for compressing content. This means that organizations utilizing this module are particularly vulnerable to this issue.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in a significant disruption of services, financial losses, and reputational damage. It is crucial for the organization to update the affected versions of Apache httpd to a patched version or disable the mod_deflate module to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

The mod_deflate module in Apache httpd 2.2.11 and earlier compresses large files until completion even after the associated network connection is closed, which allows remote attackers to cause a denial of service (CPU consumption).

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_deflate

Description:
The mod_deflate module in Apache HTTP Server versions 2.2.11 and earlier compresses large files until completion, even after the associated network connection is closed. This behavior allows remote attackers to cause a denial of service (CPU consumption) by sending specially crafted requests that trigger the compression of large files. This vulnerability can lead to server resource exhaustion and impact the availability of the service.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_deflate, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.11.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Disable mod_deflate: If mod_deflate is not required for your environment, consider disabling it. This can be done by modifying the Apache HTTP Server configuration file (e.g., httpd.conf) and commenting out or removing the line that enables mod_deflate. The line to be modified is usually "LoadModule deflate_module modules/mod_deflate.so".

4. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which requests can be processed by the Apache HTTP Server. This can help prevent overwhelming the server with a large number of requests.

5. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

Example Command:
To disable mod_deflate in the Apache HTTP Server configuration, follow these steps:

1. Open the Apache HTTP Server configuration file using a text editor. The location and name of the file may vary depending on your operating system and installation. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the line that enables mod_deflate. It may look like this:

```
LoadModule deflate_module modules/mod_deflate.so
```

3. Comment out the line by adding a "#" at the beginning:

```
# LoadModule deflate_module modules/mod_deflate.so
```

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 13

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-1890**
- Severity score: **7.1** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability occurs when a reverse proxy is configured and the stream_reqbody_cl function does not properly handle an amount of streamed data that exceeds the Content-Length value. This can be exploited by remote attackers to cause a denial of service by consuming excessive CPU resources.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a denial of service by consuming significant CPU resources on the affected Apache HTTP Server. This can result in degraded performance, unresponsiveness, or even complete unavailability of the server, impacting the organization's online presence and services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy module, which is responsible for proxying requests. This means that organizations utilizing this module as a reverse proxy are particularly vulnerable to this issue.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in a significant disruption of services, financial losses, and reputational damage. It is crucial for the organization to update the affected versions of Apache HTTP Server to a patched version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

The stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server before 2.3.3, when a reverse proxy is configured, does not properly handle an amount of streamed data that exceeds the Content-Length value, which allows remote attackers to cause a denial of service (CPU consumption) via crafted requests.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_proxy

Description:
The stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server before version 2.3.3, when a reverse proxy is configured, does not properly handle an amount of streamed data that exceeds the Content-Length value. This vulnerability allows remote attackers to cause a denial of service (CPU consumption) by sending crafted requests that exceed the specified Content-Length value. This can lead to server resource exhaustion and impact the availability of the service.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_proxy, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.3.3.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which requests can be processed by the Apache HTTP Server. This can help prevent overwhelming the server with a large number of requests.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 14

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-0883**
- Severity score: **6.9** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the envvars (aka envvars-std) script in the Apache HTTP Server before 2.4.2 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the envvars script, specifically in the LD_LIBRARY_PATH environment variable, where a zero-length directory name is placed. This can be exploited by local users to gain privileges by placing a Trojan horse DSO (Dynamic Shared Object) in the current working directory during the execution of apachectl.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability, they can gain elevated privileges by tricking the Apache HTTP Server into loading a malicious DSO from the current working directory. This can result in unauthorized access, data compromise, or other malicious activities.

The severity is also influenced by the requirement for local access. The vulnerability can only be exploited by users who already have access to the system, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the envvars script and the execution of apachectl. This means that organizations utilizing Apache HTTP Server and running apachectl are particularly vulnerable to this issue.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for privilege escalation and unauthorized access exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential privilege escalation by local users.

Public exploits related to this finding

Description of the finding

envvars (aka envvars-std) in the Apache HTTP Server before 2.4.2 places a zero-length directory name in the LD_LIBRARY_PATH, which allows local users to gain privileges via a Trojan horse DSO in the current working directory during execution of apachectl.

Mitigation steps

Vulnerability: Apache HTTP Server Privilege Escalation via LD_LIBRARY_PATH

Description:
The envvars script (aka envvars-std) in the Apache HTTP Server before version 2.4.2 places a zero-length directory name in the LD_LIBRARY_PATH environment variable. This allows local users to gain privileges by placing a Trojan horse Dynamic Shared Object (DSO) in the current working directory during the execution of apachectl. This vulnerability can lead to privilege escalation and unauthorized access to sensitive resources.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server privilege escalation via LD_LIBRARY_PATH, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.2.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Remove Zero-Length Directory from LD_LIBRARY_PATH: Modify the envvars script to remove the zero-length directory from the LD_LIBRARY_PATH environment variable. The envvars script is typically located in the Apache HTTP Server installation directory. Look for the line that sets the LD_LIBRARY_PATH variable and remove the zero-length directory.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Limit Access to Apache HTTP Server Files: Restrict access to Apache HTTP Server files and directories to trusted users only. Ensure that appropriate file permissions and ownership are set to prevent unauthorized modifications.

Example Command:
To remove the zero-length directory from the LD_LIBRARY_PATH in the envvars script, follow these steps:

1. Locate the envvars script in your Apache HTTP Server installation directory. The exact location may vary depending on your operating system and installation method.

2. Open the envvars script using a text editor. For example:

```
sudo nano /usr/local/apache2/bin/envvars
```

3. Look for the line that sets the LD_LIBRARY_PATH variable. It may look like this:

```
LD_LIBRARY_PATH=""
```

4. Remove the zero-length directory from the line, so it looks like this:

```
LD_LIBRARY_PATH="/path/to/other/directories"
```

Replace "/path/to/other/directories" with the appropriate directories for your environment.

5. Save the changes and exit the text editor.

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 15

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2016-5387**
- Severity score: **6.8** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server through version 2.4.23 has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability occurs because the Apache HTTP Server follows RFC 3875 section 4.1.18 and does not protect applications from untrusted client data in the HTTP_PROXY environment variable. This can be exploited by remote attackers to redirect an application's outbound HTTP traffic to an arbitrary proxy server via a crafted Proxy header in an HTTP request, leading to potential data leakage or manipulation.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can redirect an application's outbound HTTP traffic to an arbitrary proxy server. This can result in unauthorized access to sensitive data, data exfiltration, or manipulation of the organization's outbound communication.

The severity is also influenced by the fact that this vulnerability is well-known as "httpoxy" and has been assigned a mitigation identifier (CVE-2016-5387) by the vendor. While it may not be a traditional CVE ID for a vulnerability, the recognition of this issue as a security concern by the vendor further emphasizes its severity.

It is worth noting that the vulnerability affects multiple versions of the Apache HTTP Server, indicating a potentially wide impact across various deployments.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in a significant compromise of data confidentiality, integrity, and availability. It is crucial for the organization to update the affected versions of Apache HTTP Server to a patched version or implement the necessary mitigations to prevent potential redirection of outbound HTTP traffic via the HTTP_PROXY environment variable.

Public exploits related to this finding

Description of the finding

The Apache HTTP Server through 2.4.23 follows RFC 3875 section 4.1.18 and therefore does not protect applications from the presence of untrusted client data in the HTTP_PROXY environment variable, which might allow remote attackers to redirect an application's outbound HTTP traffic to an arbitrary proxy server via a crafted Proxy header in an HTTP request, aka an "httpoxy" issue. NOTE: the vendor states "This mitigation has been assigned the identifier CVE-2016-5387"; in other words, this is not a CVE ID for a vulnerability.

Mitigation steps

Vulnerability: Apache HTTP Server HTTP_PROXY Environment Variable Manipulation (httpoxy)

Description:
The Apache HTTP Server through version 2.4.23 follows RFC 3875 section 4.1.18 and does not protect applications from the presence of untrusted client data in the HTTP_PROXY environment variable. This vulnerability, known as "httpoxy," allows remote attackers to redirect an application's outbound HTTP traffic to an arbitrary proxy server by sending a crafted Proxy header in an HTTP request. This can lead to unauthorized access or interception of sensitive data.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server HTTP_PROXY environment variable manipulation (httpoxy), the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.23.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Protect Applications: Modify your applications to explicitly unset or ignore the HTTP_PROXY environment variable. This can be done by adding code to your application to remove or ignore the variable before processing any requests.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To unset or ignore the HTTP_PROXY environment variable in your application, follow these steps:

1. Identify the programming language and framework used by your application. The steps may vary depending on the specific technologies being used.

2. Locate the code that handles incoming HTTP requests in your application.

3. Add code to unset or ignore the HTTP_PROXY environment variable before processing any requests. The exact code will depend on the programming language and framework being used. Here is an example in Python:

```python
import os

# Unset the HTTP_PROXY environment variable
if 'HTTP_PROXY' in os.environ:
    del os.environ['HTTP_PROXY']

# Continue processing the request
# ...
```

Adjust the code according to the programming language and framework being used in your application.

4. Save the changes and redeploy your application.

Note: The specific steps may vary depending on the programming language, framework, and application architecture being used. It is important to consult the documentation and best practices for your specific environment.

Finding 16

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2014-0226**
- Severity score: **6.8** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_status module in the Apache HTTP Server before 2.4.10 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs due to a race condition in the mod_status module, specifically within the status_handler function in modules/generators/mod_status.c and the lua_ap_scoreboard_worker function in modules/lua/lua_request.c. This can be exploited by remote attackers to cause a denial of service (heap-based buffer overflow) or potentially obtain sensitive credential information or execute arbitrary code.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a denial of service by triggering a heap-based buffer overflow. Additionally, there is a possibility of obtaining sensitive credential information or executing arbitrary code, which can lead to unauthorized access, data compromise, or other malicious activities.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_status module and the Lua module in Apache HTTP Server. This means that organizations utilizing these modules are particularly vulnerable to this issue.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service, information disclosure, or code execution exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks and unauthorized access or execution of arbitrary code.

Public exploits related to this finding

Exploit ID 34133.

- Description: Apache 2.4.7 mod_status - Scoreboard Handling Race Condition

- Download URL: https://www.exploit-db.com/exploits/34133

Description of the finding

Race condition in the mod_status module in the Apache HTTP Server before 2.4.10 allows remote attackers to cause a denial of service (heap-based buffer overflow), or possibly obtain sensitive credential information or execute arbitrary code, via a crafted request that triggers improper scoreboard handling within the status_handler function in modules/generators/mod_status.c and the lua_ap_scoreboard_worker function in modules/lua/lua_request.c.

Mitigation steps

Vulnerability: Apache HTTP Server Race Condition in mod_status

Description:
The mod_status module in the Apache HTTP Server before version 2.4.10 is vulnerable to a race condition. This vulnerability allows remote attackers to cause a denial of service (heap-based buffer overflow) or potentially obtain sensitive credential information or execute arbitrary code. The vulnerability is triggered by a crafted request that triggers improper scoreboard handling within the status_handler function in modules/generators/mod_status.c and the lua_ap_scoreboard_worker function in modules/lua/lua_request.c.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server race condition in mod_status, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.10.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Disable mod_status: If mod_status is not required for your environment, consider disabling it. This can be done by modifying the Apache HTTP Server configuration file (e.g., httpd.conf) and commenting out or removing the line that enables mod_status. The line to be modified is usually "LoadModule status_module modules/mod_status.so".

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log

Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To disable mod_status in the Apache HTTP Server configuration, follow these steps:

1. Open the Apache HTTP Server configuration file using a text editor. The location and name of the file may vary depending on your operating system and installation. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the line that enables mod_status. It may look like this:

```
LoadModule status_module modules/mod_status.so
```

3. Comment out the line by adding a "#" at the beginning:

```
# LoadModule status_module modules/mod_status.so
```

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 17

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2017-9788**
- Severity score: **6.4** (**CRITICAL** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Apache httpd before 2.2.34 and 2.4.x before 2.4.27 has been classified as "Critical" severity due to its significant impact on the organization's security. A critical severity vulnerability indicates a severe risk to the organization's systems and data.

The vulnerability occurs in the handling of [Proxy-]Authorization headers of type 'Digest' by mod_auth_digest. The value placeholder in these headers was not properly initialized or reset before or between successive key=value assignments. This can lead to the leakage of potentially confidential information and, in some cases, result in a segmentation fault (segfault) leading to denial of service.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can potentially access and leak confidential information contained in the uninitialized pool memory. Additionally, in certain cases, a segfault can occur, leading to a denial of service.

The severity is also influenced by the fact that this vulnerability is critical in nature and can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability affects multiple versions of Apache httpd, indicating a potentially wide impact across various deployments.

In summary, the critical severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in severe consequences, including unauthorized access, data breaches, denial of service, financial losses, reputational damage, and potential legal liabilities. It is crucial for the organization to update the affected versions of Apache httpd to a patched version to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

In Apache httpd before 2.2.34 and 2.4.x before 2.4.27, the value placeholder in [Proxy-]Authorization headers of type 'Digest' was not initialized or reset before or between successive key=value assignments by mod_auth_digest. Providing an initial key with no '=' assignment could reflect the stale value of uninitialized pool memory used by the prior request, leading to leakage of potentially confidential information, and a segfault in other cases resulting in denial of service.

Mitigation steps

Vulnerability: Apache HTTP Server Information Leakage and Denial of Service (DoS) via Digest Authorization Headers

Description:
In Apache HTTP Server versions before 2.2.34 and 2.4.x before 2.4.27, the value placeholder in [Proxy-]Authorization headers of type 'Digest' was not properly initialized or reset before or between successive key=value assignments by mod_auth_digest. This vulnerability allows remote attackers to exploit uninitialized pool memory, leading to the leakage of potentially confidential information. In some cases, it can also result in a segmentation fault (segfault), causing a denial of service (DoS) condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server information leakage and DoS via Digest Authorization headers, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.34 or 2.4.27, depending on the version you are currently using.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 18

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-1956**
- Severity score: **6.4** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the apr_brigade_vprintf function in Apache APR-util before 1.3.5 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs due to an off-by-one error in the apr_brigade_vprintf function on big-endian platforms. This can be exploited by remote attackers to obtain sensitive information or cause a denial of service by crafting malicious input.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can obtain sensitive information or cause an application crash, leading to a denial of service.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the Apache Portable Runtime (APR) utility library, which is used by various applications, including Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for information disclosure or denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of APR-util that includes the fix for this vulnerability to prevent potential information disclosure or denial of service attacks.

Public exploits related to this finding

Description of the finding

Off-by-one error in the apr_brigade_vprintf function in Apache APR-util before 1.3.5 on big-endian platforms allows remote attackers to obtain sensitive information or cause a denial of service (application crash) via crafted input.

Mitigation steps

Vulnerability: Apache APR-util Off-by-One Error

Description:
The apr_brigade_vprintf function in Apache APR-util before version 1.3.5 on big-endian platforms contains an off-by-one error. This vulnerability allows remote attackers to obtain sensitive information or cause a denial of service (application crash) by providing crafted input. The vulnerability is due to improper handling of input, leading to a buffer overflow or memory corruption.

Mitigation Steps:
To mitigate the risk of the Apache APR-util off-by-one error, the following steps can be taken:

1. Update Apache APR-util: Upgrade to a version of Apache APR-util that includes a fix for this vulnerability. Ensure that you are using a version later than 1.3.5.

2. Apply Patches: If an official patch is available for your version of Apache APR-util, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache APR-util and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

4. Monitor and Log Application Activity: Implement a centralized logging system to monitor and log application activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Input Validation and Sanitization: Implement strict input validation and sanitization mechanisms in your applications. Validate and sanitize all user-supplied input to prevent buffer overflows and other types of vulnerabilities.

Example Command:
To update Apache APR-util, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

```
```

or

```
sudo yum update
```

3. Upgrade the Apache APR-util package:

```
sudo apt upgrade libaprutil1
```

or

```
sudo yum upgrade apr-util
```

Replace "libaprutil1" or "apr-util" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 19

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-3555**
- Severity score: **5.8** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the TLS and SSL protocols has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the TLS and SSL protocols, specifically in the way renegotiation handshakes are associated with existing connections. This can be exploited by man-in-the-middle attackers to insert data into HTTPS sessions and potentially other types of sessions protected by TLS or SSL.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can insert data into encrypted sessions, potentially leading to unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the fact that this vulnerability affects various implementations of TLS and SSL, including Microsoft Internet Information Services (IIS), Apache HTTP Server, OpenSSL, GnuTLS, Mozilla Network Security Services (NSS), and multiple Cisco products. This indicates a potentially wide impact across various deployments.

It is worth noting that this vulnerability is related to a "plaintext injection" attack, also known as the "Project Mogul" issue.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized data insertion exists, the vulnerability requires a man-in-the-middle position and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating the affected TLS and SSL implementations to versions that include the fix for this vulnerability to prevent potential data injection attacks.

Public exploits related to this finding

Exploit ID 10071.

- Description: Mozilla NSS - NULL Character CA SSL Certificate Validation Security Bypass
- Download URL: https://www.exploit-db.com/exploits/10071

Exploit ID 10579.

- Description: TLS - Renegotiation
- Download URL: https://www.exploit-db.com/exploits/10579

Description of the finding

The TLS protocol, and the SSL protocol 3.0 and possibly earlier, as used in Microsoft Internet Information Services (IIS) 7.0, mod_ssl in the Apache HTTP Server 2.2.14 and earlier, OpenSSL before 0.9.8l, GnuTLS 2.8.5 and earlier, Mozilla Network Security Services (NSS) 3.12.4 and earlier, multiple Cisco products, and other products, does not properly associate renegotiation handshakes with an existing connection, which allows man-in-the-middle attackers to insert data into HTTPS sessions, and possibly other types of sessions protected by TLS or SSL, by sending an unauthenticated request that is processed retroactively by a server in a post-renegotiation context, related to a "plaintext injection" attack, aka the "Project Mogul" issue.

Mitigation steps

Vulnerability: TLS/SSL Renegotiation Plaintext Injection

Description:
The TLS protocol, as well as the SSL protocol 3.0 and possibly earlier versions, have a vulnerability that allows man-in-the-middle attackers to insert data into HTTPS sessions and other sessions protected by TLS or SSL. This vulnerability occurs due to a failure to properly associate renegotiation handshakes with an existing connection. Attackers can exploit this vulnerability by sending an unauthenticated request that is processed retroactively by a server in a post-renegotiation context. This vulnerability is also known as the "Project Mogul" issue.

Mitigation Steps:
To mitigate the risk of TLS/SSL renegotiation plaintext injection, the following steps can be taken:

1. Update Software: Ensure that all software components involved in TLS/SSL connections, including web servers (such as Microsoft IIS or Apache HTTP Server), OpenSSL, GnuTLS, and Mozilla NSS, are updated to versions that include a fix for this vulnerability. Check with the respective vendors or open-source projects for the latest security patches.

2. Disable Renegotiation: Consider disabling renegotiation in the TLS/SSL configuration of your web server. Renegotiation is not always necessary and disabling it can help mitigate the risk of this vulnerability. Consult the documentation for your specific web server software to learn how to disable renegotiation.

3. Implement Strict Certificate Validation: Configure your web server and client applications to perform

strict certificate validation. This helps prevent man-in-the-middle attacks and ensures that the server's identity is verified during the TLS/SSL handshake.

4. Monitor for Anomalous Activity: Implement monitoring and logging mechanisms to detect any anomalous or suspicious activity related to TLS/SSL connections. Regularly review logs for any signs of unauthorized access or plaintext injection attempts.

5. Regularly Update and Patch Software: Keep all software, including web servers, TLS/SSL libraries, and operating systems, up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to perform various attacks, including plaintext injection.

Example Command:
To disable renegotiation in Apache HTTP Server, follow these steps:

1. Open the Apache HTTP Server configuration file using a text editor. The location and name of the file may vary depending on your operating system and installation. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the line that enables SSL/TLS renegotiation. It may look like this:

```
SSLRenegotation on
```

3. Change the value to "off" to disable renegotiation:

```
SSLRenegotation off
```

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 20

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2013-1862**
- Severity score: **5.1** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_rewrite module in the Apache HTTP Server 2.2.x before 2.2.25 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_rewrite module, specifically in the way it writes data to a log file without sanitizing non-printable characters. This can be exploited by remote attackers to execute arbitrary commands by sending an HTTP request containing an escape sequence for a terminal emulator.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can execute arbitrary commands on the affected Apache HTTP Server by injecting specially crafted escape sequences into the log file. This can result in unauthorized access, data compromise, or other malicious activities.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_rewrite module, which is responsible for URL rewriting and redirection in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for command execution exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential command execution attacks.

Public exploits related to this finding

Description of the finding

mod_rewrite.c in the mod_rewrite module in the Apache HTTP Server 2.2.x before 2.2.25 writes data to a log file without sanitizing non-printable characters, which might allow remote attackers to execute arbitrary commands via an HTTP request containing an escape sequence for a terminal emulator.

Mitigation steps

Vulnerability: Apache HTTP Server Arbitrary Command Execution via mod_rewrite

Description:
The mod_rewrite module in the Apache HTTP Server versions 2.2.x before 2.2.25 writes data to a log file without sanitizing non-printable characters. This vulnerability allows remote attackers to execute arbitrary commands by sending an HTTP request containing an escape sequence for a terminal emulator. Attackers can exploit this vulnerability to execute arbitrary commands with the privileges of the user running the Apache HTTP Server process.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server arbitrary command execution via mod_rewrite, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.25.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

4. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent the execution of arbitrary commands. Validate and sanitize all user-supplied input to prevent command injection vulnerabilities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 21

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2017-9798**
- Severity score: **5.0** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Apache httpd, known as Optionsbleed, has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability allows remote attackers to read secret data from process memory if the Limit directive can be set in a user's .htaccess file or if httpd.conf has certain misconfigurations. This affects the Apache HTTP Server through versions 2.2.34 and 2.4.x through 2.4.27.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can read secret data from the process memory of the affected Apache httpd server. The specific data that can be accessed depends on various factors, including configuration, but it can potentially include sensitive information such as passwords, API keys, or other confidential data.

The severity is also influenced by the fact that this vulnerability can be exploited remotely by sending an unauthenticated OPTIONS HTTP request. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability is a use-after-free issue, and the secret data is not always sent. The specific data that can be accessed depends on various factors, including configuration.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in the unauthorized access and exposure of sensitive data, leading to potential data breaches, financial losses, reputational damage, and potential legal liabilities. It is crucial for the organization to apply the necessary patches or updates provided by Apache to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Exploit ID 42745.

- Description: Apache < 2.2.34 / < 2.4.27 - OPTIONS Memory Leak
- Download URL: https://www.exploit-db.com/exploits/42745

Description of the finding

Apache httpd allows remote attackers to read secret data from process memory if the Limit directive can be set in a user's .htaccess file, or if httpd.conf has certain misconfigurations, aka Optionsbleed. This affects the Apache HTTP Server through 2.2.34 and 2.4.x through 2.4.27. The attacker sends an unauthenticated OPTIONS HTTP request when attempting to read secret data. This is a use-after-free issue and thus secret data is not always sent, and the specific data depends on many factors including configuration. Exploitation with .htaccess can be blocked with a patch to the ap_limit_section function in server/core.c.

Mitigation steps

Vulnerability: Apache HTTP Server Optionsbleed Information Disclosure

Description:
Apache HTTP Server versions through 2.2.34 and 2.4.x through 2.4.27 are vulnerable to an information disclosure vulnerability known as Optionsbleed. This vulnerability allows remote attackers to read secret data from process memory if the Limit directive can be set in a user's .htaccess file or if httpd.conf has certain misconfigurations. The attacker sends an unauthenticated OPTIONS HTTP request to trigger the vulnerability. This is a use-after-free issue, and the specific data that can be leaked depends on various factors, including the server's configuration.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server Optionsbleed information disclosure, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.34 or 2.4.27, depending on the version you are currently using.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Review and Secure Configuration: Review your Apache HTTP Server configuration files, including httpd.conf and .htaccess files, to ensure that the Limit directive is properly configured and restricted. Remove any unnecessary or insecure configurations.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 22

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2016-8743**
- Severity score: **5.0** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a substantial risk to the organization's systems and data.

The vulnerability occurs in all releases of Apache HTTP Server prior to versions 2.2.32 and 2.4.25. It relates to the handling of whitespace in requests, response lines, and headers. The liberal acceptance of whitespace can lead to security concerns when Apache HTTP Server participates in any chain of proxies or interacts with back-end application servers. This vulnerability can result in request smuggling, response splitting, and cache pollution.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can manipulate whitespace in requests and responses to perform request smuggling, response splitting, or cache pollution attacks. These attacks can lead to unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server when it participates in proxy chains or interacts with back-end application servers through mod_proxy or conventional CGI mechanisms.

In summary, the high severity of this vulnerability emphasizes the urgent need for the organization to address and remediate the issue. Failure to do so can result in severe consequences, including unauthorized access, data manipulation, cache pollution, financial losses, reputational damage, and potential legal liabilities. It is crucial for the organization to update the affected versions of Apache HTTP Server to versions 2.2.32 or 2.4.25 or apply the necessary patches provided by Apache to mitigate the risk posed by this vulnerability.

Public exploits related to this finding

Description of the finding

Apache HTTP Server, in all releases prior to 2.2.32 and 2.4.25, was liberal in the whitespace accepted from requests and sent in response lines and headers. Accepting these different behaviors represented a security concern when httpd participates in any chain of proxies or interacts with back-end application servers, either through mod_proxy or using conventional CGI mechanisms, and may result in request smuggling, response splitting and cache pollution.

Mitigation steps

Vulnerability: Apache HTTP Server Request Smuggling and Response Splitting

Description:
Apache HTTP Server versions prior to 2.2.32 and 2.4.25 were liberal in accepting whitespace from requests and sending it in response lines and headers. This behavior represented a security concern when Apache HTTP Server participates in any chain of proxies or interacts with back-end application servers. It may result in request smuggling, response splitting, and cache pollution. Attackers can exploit this vulnerability to manipulate the behavior of the server and potentially bypass security controls or inject malicious content.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server request smuggling and response splitting, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.32 or 2.4.25, depending on the version you are currently using.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Secure Proxy and Backend Interactions: Review and secure the configuration of any proxy or backend interactions involving Apache HTTP Server. Ensure that proper input validation and sanitization mechanisms are in place to prevent request smuggling and response splitting attacks.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 23

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2014-0231**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_cgid module in the Apache HTTP Server before 2.4.10 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_cgid module, which does not have a timeout mechanism. This can be exploited by remote attackers to cause a denial of service by sending a request to a CGI script that does not read from its stdin file descriptor.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a process hang by sending a request to a CGI script that does not read from its stdin file descriptor. This can result in resource exhaustion and unresponsiveness of the affected Apache HTTP Server.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_cgid module, which is responsible for executing CGI scripts in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks.

Public exploits related to this finding

Description of the finding

The mod_cgid module in the Apache HTTP Server before 2.4.10 does not have a timeout mechanism, which allows remote attackers to cause a denial of service (process hang) via a request to a CGI script that does not read from its stdin file descriptor.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_cgid

Description:
The mod_cgid module in the Apache HTTP Server versions before 2.4.10 does not have a timeout mechanism. This allows remote attackers to cause a denial of service (process hang) by sending a request to a CGI script that does not read from its stdin file descriptor. The lack of a timeout mechanism can result in the CGI script hanging indefinitely, consuming server resources and impacting the availability of the service.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_cgid, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.10.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Timeout Mechanism: Configure a timeout mechanism for the mod_cgid module in the Apache HTTP Server configuration. This can be done by adding or modifying the following line in the httpd.conf file:

```
CGIDScriptTimeout <timeout>
```

Replace `<timeout>` with the desired timeout value in seconds. This ensures that CGI scripts that do not read from stdin will not hang indefinitely.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To configure a timeout mechanism for the mod_cgid module in the Apache HTTP Server configuration,

follow these steps:

1. Open the Apache HTTP Server configuration file using a text editor. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the line that controls the CGIDScriptTimeout setting. It may look like this:

```
#CGIDScriptTimeout 30
```

3. Uncomment the line by removing the "#" at the beginning, and set the desired timeout value:

```
CGIDScriptTimeout 60
```

In this example, the timeout value is set to 60 seconds.

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 24

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2014-0098**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_log_config module in the Apache HTTP Server before 2.4.8 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the log_cookie function in mod_log_config.c, where a crafted cookie is not properly handled during truncation. This can be exploited by remote attackers to cause a denial of service by triggering a segmentation fault and crashing the Apache HTTP Server daemon.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a segmentation fault and crash the Apache HTTP Server daemon, resulting in a denial of service.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_log_config module, which is responsible for logging configuration in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks.

Public exploits related to this finding

Description of the finding

The log_cookie function in mod_log_config.c in the mod_log_config module in the Apache HTTP Server before 2.4.8 allows remote attackers to cause a denial of service (segmentation fault and daemon crash) via a crafted cookie that is not properly handled during truncation.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_log_config

Description:
The log_cookie function in the mod_log_config module in the Apache HTTP Server versions before 2.4.8 is vulnerable to a denial of service (DoS) attack. Remote attackers can cause a segmentation fault and crash the server by sending a crafted cookie that is not properly handled during truncation. This vulnerability can impact the availability of the server and result in a DoS condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_log_config, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.8.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

```
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 25

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2013-6438**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_dav module in the Apache HTTP Server before 2.4.8 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the dav_xml_get_cdata function in main/util.c, where whitespace characters are not properly removed from CDATA sections. This can be exploited by remote attackers to cause a denial of service by sending a crafted DAV WRITE request that triggers a daemon crash.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a daemon crash in the Apache HTTP Server, resulting in a denial of service.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_dav module, which is responsible for providing WebDAV functionality in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks.

Public exploits related to this finding

Description of the finding

The dav_xml_get_cdata function in main/util.c in the mod_dav module in the Apache HTTP Server before 2.4.8 does not properly remove whitespace characters from CDATA sections, which allows remote attackers to cause a denial of service (daemon crash) via a crafted DAV WRITE request.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_dav

Description:
The dav_xml_get_cdata function in the mod_dav module in the Apache HTTP Server versions before 2.4.8 does not properly remove whitespace characters from CDATA sections. This vulnerability allows remote attackers to cause a denial of service (daemon crash) by sending a crafted DAV WRITE request. The improper handling of whitespace characters can lead to a crash of the server, impacting its availability.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_dav, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.8.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

```
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 26

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2013-5704**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_headers module in the Apache HTTP Server 2.2.22 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability allows remote attackers to bypass "RequestHeader unset" directives by placing a header in the trailer portion of data sent with chunked transfer coding.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can bypass "RequestHeader unset" directives, potentially allowing them to manipulate or inject headers into the HTTP response.

However, it is worth noting that the vendor states that this is not a security issue in Apache HTTP Server itself. This suggests that the impact of this vulnerability may be limited and may not directly lead to unauthorized access, data compromise, or disruption of critical systems.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for bypassing "RequestHeader unset" directives exists, the vendor does not consider it a security issue in Apache HTTP Server itself. It is still important for the organization to be aware of this behavior and consider appropriate mitigations or workarounds if necessary.

Public exploits related to this finding

Description of the finding

The mod_headers module in the Apache HTTP Server 2.2.22 allows remote attackers to bypass "RequestHeader unset" directives by placing a header in the trailer portion of data sent with chunked transfer coding. NOTE: the vendor states "this is not a security issue in httpd as such."

Mitigation steps

Vulnerability: Apache HTTP Server Bypass of "RequestHeader unset" Directives

Description:

The mod_headers module in the Apache HTTP Server version 2.2.22 allows remote attackers to bypass "RequestHeader unset" directives. This can be achieved by placing a header in the trailer portion of data sent with chunked transfer coding. It is important to note that the vendor states that "this is not a security issue in httpd as such."

Mitigation Steps:
To mitigate the risk of bypassing "RequestHeader unset" directives in Apache HTTP Server, the following steps can be taken:

1. Review and Update Configuration: Review the Apache HTTP Server configuration and ensure that the "RequestHeader unset" directives are properly configured. Verify that the directives are correctly applied and that the desired headers are being unset.

2. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

3. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

4. Implement Additional Security Measures: Consider implementing additional security measures, such as a Web Application Firewall (WAF), to provide an extra layer of protection against potential vulnerabilities and attacks.

Example Command:
To review and update the Apache HTTP Server configuration, follow these steps:

1. Open the Apache HTTP Server configuration file using a text editor. The location and name of the file may vary depending on your operating system and installation. For example:

```
sudo nano /etc/httpd/conf/httpd.conf
```

2. Locate the "RequestHeader unset" directives in the configuration file. These directives are used to unset specific headers. Verify that the directives are correctly applied and that the desired headers are being unset.

3. Make any necessary changes to the configuration file to update the "RequestHeader unset" directives.

4. Save the changes and exit the text editor.

5. Restart the Apache HTTP Server for the changes to take effect:

```
sudo service httpd restart
```

Note: The specific steps may vary depending on the operating system and version of Apache HTTP Server being used. It is important to consult the documentation and best practices for your specific environment.

Finding 27

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-3368**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy module in the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_proxy module in Apache HTTP Server versions 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21. It relates to the interaction between the module and the use of RewriteRule and ProxyPassMatch pattern matches for configuring a reverse proxy. This vulnerability allows remote attackers to send requests to intranet servers via a malformed URI containing an initial @ (at sign) character.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can send requests to intranet servers by crafting a URI with a malformed initial @ character. This can potentially lead to unauthorized access to intranet resources and data exposure.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy module, which is responsible for proxying requests in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized access and data exposure exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential unauthorized access to intranet resources.

Public exploits related to this finding

Exploit ID 17969.

- Description: Apache mod_proxy - Reverse Proxy Exposure

- Download URL: https://www.exploit-db.com/exploits/17969

Description of the finding

The mod_proxy module in the Apache HTTP Server 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21 does not properly interact with use of (1) RewriteRule and (2) ProxyPassMatch pattern matches for configuration of a reverse proxy, which allows remote attackers to send requests to intranet servers via a malformed URI containing an initial @ (at sign) character.

Mitigation steps

Vulnerability: Apache HTTP Server Reverse Proxy Exposure

Description:
The mod_proxy module in the Apache HTTP Server versions 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21 has a vulnerability that allows remote attackers to send requests to intranet servers via a malformed URI containing an initial "@" (at sign) character. This vulnerability occurs when using RewriteRule and ProxyPassMatch pattern matches for the configuration of a reverse proxy. Attackers can exploit this vulnerability to bypass security restrictions and access intranet servers.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server reverse proxy exposure, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Review and Secure Configuration: Review your Apache HTTP Server configuration files, including httpd.conf and any other relevant configuration files, to ensure that reverse proxy configurations are properly secured. Verify that access controls and restrictions are in place to prevent unauthorized access to intranet servers.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to gain unauthorized access or perform other malicious activities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access

attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 28

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2010-1452**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_cache and mod_dav modules in the Apache HTTP Server 2.2.x before 2.2.16 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_cache and mod_dav modules, where a request that lacks a path can cause a denial of service by crashing the Apache HTTP Server process.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a process crash in the Apache HTTP Server, resulting in a denial of service.

The severity is also influenced by the requirement for remote access. The vulnerability can only be exploited by remote attackers, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects the mod_cache and mod_dav modules, which are responsible for caching and WebDAV functionality in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks.

Public exploits related to this finding

Description of the finding

The (1) mod_cache and (2) mod_dav modules in the Apache HTTP Server 2.2.x before 2.2.16 allow remote attackers to cause a denial of service (process crash) via a request that lacks a path.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_cache and mod_dav

Description:
The mod_cache and mod_dav modules in the Apache HTTP Server versions 2.2.x before 2.2.16 have a vulnerability that allows remote attackers to cause a denial of service (process crash). This vulnerability occurs when a request lacks a path. Attackers can exploit this vulnerability to crash the server, resulting in a denial of service condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_cache and mod_dav, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.16.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
```

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 29

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2010-0408**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy_ajp module in the Apache HTTP Server 2.2.x before 2.2.15 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the ap_proxy_ajp_request function in mod_proxy_ajp.c, where certain situations in which a client sends no request body are not properly handled. This can be exploited by remote attackers to cause a denial of service by crafting a request that triggers a backend server outage.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a backend server outage by sending a crafted request that does not include a request body. This can result in the unavailability of the backend server and potentially impact the organization's services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy_ajp module, which is responsible for proxying AJP (Apache JServ Protocol) requests in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential backend server outages.

Public exploits related to this finding

Description of the finding

The ap_proxy_ajp_request function in mod_proxy_ajp.c in mod_proxy_ajp in the Apache HTTP Server 2.2.x before 2.2.15 does not properly handle certain situations in which a client sends no request body, which

allows remote attackers to cause a denial of service (backend server outage) via a crafted request, related to use of a 500 error code instead of the appropriate 400 error code.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_proxy_ajp

Description:
The ap_proxy_ajp_request function in mod_proxy_ajp.c in mod_proxy_ajp in the Apache HTTP Server versions 2.2.x before 2.2.15 has a vulnerability that allows remote attackers to cause a denial of service (backend server outage). This vulnerability occurs in certain situations where a client sends no request body. Attackers can exploit this vulnerability by sending a crafted request, causing a backend server outage. The vulnerability is related to the use of a 500 error code instead of the appropriate 400 error code.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_proxy_ajp, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.15.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Vulnerabilities in software can be exploited by attackers to cause DoS conditions.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Implement Additional Security Measures: Consider implementing additional security measures, such as rate limiting or access controls, to protect against potential DoS attacks and unauthorized access.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 30

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-3095**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy_ftp module in the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability allows remote attackers to bypass intended access restrictions and send arbitrary commands to an FTP server. This is achieved through vectors related to the embedding of these commands in the Authorization HTTP header.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can bypass access restrictions and send arbitrary commands to an FTP server. This can result in unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy_ftp module, which is responsible for proxying FTP requests in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized access and command execution exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential unauthorized access and command execution on FTP servers.

Public exploits related to this finding

Description of the finding

The mod_proxy_ftp module in the Apache HTTP Server allows remote attackers to bypass intended access restrictions and send arbitrary commands to an FTP server via vectors related to the embedding of

these commands in the Authorization HTTP header, as demonstrated by a certain module in VulnDisco Pack Professional 8.11.

Mitigation steps

Vulnerability: Apache HTTP Server FTP Proxy Command Injection

Description:
The mod_proxy_ftp module in the Apache HTTP Server has a vulnerability that allows remote attackers to bypass intended access restrictions and send arbitrary commands to an FTP server. This vulnerability is related to the embedding of these commands in the Authorization HTTP header. Attackers can exploit this vulnerability to execute arbitrary commands on the FTP server, bypassing intended access restrictions.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server FTP proxy command injection, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive resources. This can help prevent unauthorized access and limit the impact of potential vulnerabilities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 31

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-2699**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Solaris pollset feature in the Event Port backend in poll/unix/port.c in the Apache Portable Runtime (APR) library has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the APR library, specifically in the Solaris pollset feature used in the Event Port backend. It relates to the improper handling of errors, which can be exploited by remote attackers to cause a denial of service by sending unspecified HTTP requests that result in a daemon hang.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a daemon hang in the affected Apache HTTP Server or other products that use the APR library. This can result in unresponsiveness and potential disruption of services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the prefork and event MPMs (Multi-Processing Modules) used in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of the APR library that includes the fix for this vulnerability to prevent potential daemon hangs and denial of service attacks.

Public exploits related to this finding

Description of the finding

The Solaris pollset feature in the Event Port backend in poll/unix/port.c in the Apache Portable Runtime (APR) library before 1.3.9, as used in the Apache HTTP Server before 2.2.14 and other products, does not properly handle errors, which allows remote attackers to cause a denial of service (daemon hang) via unspecified HTTP requests, related to the prefork and event MPMs.

Mitigation steps

Vulnerability: Apache Portable Runtime (APR) Denial of Service (DoS) via Solaris pollset

Description:
The Solaris pollset feature in the Event Port backend in poll/unix/port.c in the Apache Portable Runtime (APR) library before version 1.3.9 has a vulnerability that allows remote attackers to cause a denial of service (daemon hang). This vulnerability occurs due to improper handling of errors in the Solaris pollset feature. Attackers can exploit this vulnerability by sending unspecified HTTP requests, which can cause a daemon hang and result in a denial of service condition. This vulnerability is related to the prefork and event MPMs.

Mitigation Steps:
To mitigate the risk of Apache Portable Runtime (APR) DoS via Solaris pollset, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Implement Additional Security Measures: Consider implementing additional security measures, such as rate limiting or access controls, to protect against potential DoS attacks and unauthorized access.

Example Command:

To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 32

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2008-2364**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the ap_proxy_http_process_response function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server 2.0.63 and 2.2.8 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_proxy module, specifically in the ap_proxy_http_process_response function. It relates to the lack of limitation on the number of forwarded interim responses, which can be exploited by remote HTTP servers to cause a denial of service by consuming excessive memory through a large number of interim responses.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can cause a denial of service by overwhelming the Apache HTTP Server with a large number of interim responses, leading to excessive memory consumption and potential unresponsiveness.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy module, which is responsible for proxying requests in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks through excessive memory consumption.

Public exploits related to this finding

Description of the finding

The ap_proxy_http_process_response function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server 2.0.63 and 2.2.8 does not limit the number of forwarded interim responses, which allows remote HTTP servers to cause a denial of service (memory consumption) via a large number of interim responses.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_proxy

Description:
The ap_proxy_http_process_response function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server versions 2.0.63 and 2.2.8 has a vulnerability that allows remote HTTP servers to cause a denial of service (memory consumption). This vulnerability occurs due to the lack of limitation on the number of forwarded interim responses. Attackers can exploit this vulnerability by sending a large number of interim responses, leading to excessive memory consumption and causing a denial of service condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_proxy, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Implement Additional Security Measures: Consider implementing additional security measures, such as rate limiting or access controls, to protect against potential DoS attacks and unauthorized access.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 33

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2007-6750**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server 1.x and 2.x has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability allows remote attackers to cause a denial of service by sending partial HTTP requests, as demonstrated by the Slowloris attack. This vulnerability is related to the lack of the mod_reqtimeout module in versions before 2.2.15.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability using the Slowloris attack technique, they can cause a denial of service by overwhelming the Apache HTTP Server with partial HTTP requests, leading to a daemon outage and unavailability of services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects versions of Apache HTTP Server before 2.2.15 and does not have the mod_reqtimeout module.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the mod_reqtimeout module or implementing other mitigation measures to prevent potential denial of service attacks through partial HTTP requests.

Public exploits related to this finding

Description of the finding

The Apache HTTP Server 1.x and 2.x allows remote attackers to cause a denial of service (daemon outage) via partial HTTP requests, as demonstrated by Slowloris, related to the lack of the mod_reqtimeout module in versions before 2.2.15.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via Partial HTTP Requests

Description:
The Apache HTTP Server versions 1.x and 2.x have a vulnerability that allows remote attackers to cause a denial of service (daemon outage) via partial HTTP requests. This vulnerability is demonstrated by Slowloris and is related to the lack of the mod_reqtimeout module in versions before 2.2.15. Attackers can exploit this vulnerability by sending partial HTTP requests, which can cause the server to become overwhelmed and result in a denial of service condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via partial HTTP requests, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which requests can be processed by the Apache HTTP Server. This can help prevent overwhelming the server with a large number of partial requests.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 34

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-1195**
- Severity score: **4.9** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server 2.2.11 and earlier 2.2 versions has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the handling of the Options=IncludesNOEXEC directive in the AllowOverride directive. This allows local users to gain privileges by configuring certain Options directives in a .htaccess file and then inserting an exec element in a .shtml file.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability, they can gain elevated privileges by configuring specific Options directives in a .htaccess file and inserting an exec element in a .shtml file. This can result in unauthorized access, privilege escalation, or other malicious activities.

The severity is also influenced by the requirement for local access. The vulnerability can only be exploited by local users, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.2.11 and earlier in the 2.2 series.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for privilege escalation exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential privilege escalation by local users.

Public exploits related to this finding

Description of the finding

The Apache HTTP Server 2.2.11 and earlier 2.2 versions does not properly handle Options=IncludesNOEXEC in the AllowOverride directive, which allows local users to gain privileges by

configuring (1) Options Includes, (2) Options +Includes, or (3) Options +IncludesNOEXEC in a .htaccess file, and then inserting an exec element in a .shtml file.

Mitigation steps

Vulnerability: Apache HTTP Server Privilege Escalation via Options Directive

Description:
The Apache HTTP Server versions 2.2.11 and earlier 2.2 versions have a vulnerability that allows local users to gain privileges. This vulnerability occurs due to improper handling of the Options=IncludesNOEXEC directive in the AllowOverride directive. Local users can exploit this vulnerability by configuring certain Options directives in a .htaccess file and then inserting an exec element in a .shtml file. By doing so, they can gain elevated privileges.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server privilege escalation via the Options directive, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Review and Secure Configuration: Review your Apache HTTP Server configuration files, including httpd.conf and any other relevant configuration files, to ensure that the AllowOverride directive is properly configured. Limit the use of Options directives and ensure that appropriate access controls are in place to prevent unauthorized access and privilege escalation.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on

CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 35

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-0031**
- Severity score: **4.6** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the scoreboard.c module in the Apache HTTP Server 2.2.21 and earlier has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the scoreboard.c module, where local users might be able to cause a denial of service or potentially have other unspecified impacts by modifying a certain type field within a shared memory segment. This can lead to an invalid call to the free function, resulting in a daemon crash during shutdown or other potential impacts.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability by modifying the type field within a shared memory segment, they can cause a denial of service by crashing the Apache HTTP Server daemon during shutdown. Additionally, there may be other unspecified impacts that could potentially lead to unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the requirement for local access. The vulnerability can only be exploited by local users, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.2.21 and earlier.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service and other impacts exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential daemon crashes and other impacts caused by modifying the shared memory segment.

Public exploits related to this finding

Exploit ID 41768.

- Description: Apache 2.2 - Scoreboard Invalid Free On Shutdown

- Download URL: https://www.exploit-db.com/exploits/41768

Description of the finding

scoreboard.c in the Apache HTTP Server 2.2.21 and earlier might allow local users to cause a denial of service (daemon crash during shutdown) or possibly have unspecified other impact by modifying a certain type field within a scoreboard shared memory segment, leading to an invalid call to the free function.

Mitigation steps

Vulnerability: Apache HTTP Server Scoreboard Invalid Free on Shutdown

Description:
The Apache HTTP Server versions 2.2.21 and earlier have a vulnerability that might allow local users to cause a denial of service (daemon crash during shutdown) or potentially have other unspecified impacts. This vulnerability occurs in the scoreboard.c file and is related to modifying a certain type field within a scoreboard shared memory segment. This can lead to an invalid call to the free function, resulting in a crash or other impacts during the shutdown process.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server scoreboard invalid free on shutdown, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 36

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-3607**
- Severity score: **4.4** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the ap_pregsub function in server/util.c in the Apache HTTP Server 2.0.x through 2.0.64 and 2.2.x through 2.2.21 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the ap_pregsub function when the mod_setenvif module is enabled. It allows local users to gain privileges by exploiting a heap-based buffer overflow through a crafted .htaccess file with a SetEnvIf directive, in conjunction with a crafted HTTP request header.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability, they can gain elevated privileges by leveraging the heap-based buffer overflow. This can result in unauthorized access, privilege escalation, or other malicious activities.

The severity is also influenced by the requirement for local access. The vulnerability can only be exploited by local users, limiting the potential scope of exploitation.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x through 2.2.21 when the mod_setenvif module is enabled.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for privilege escalation exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential privilege escalation through the crafted .htaccess file and HTTP request header.

Public exploits related to this finding

Exploit ID 41769.

- Description: Apache < 2.0.64 / < 2.2.21 mod_setenvif - Integer Overflow
- Download URL: https://www.exploit-db.com/exploits/41769

Description of the finding

Integer overflow in the ap_pregsub function in server/util.c in the Apache HTTP Server 2.0.x through 2.0.64 and 2.2.x through 2.2.21, when the mod_setenvif module is enabled, allows local users to gain privileges via a .htaccess file with a crafted SetEnvIf directive, in conjunction with a crafted HTTP request header, leading to a heap-based buffer overflow.

Mitigation steps

Vulnerability: Apache HTTP Server Integer Overflow in mod_setenvif

Description:
The Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x through 2.2.21 have a vulnerability that allows local users to gain privileges. This vulnerability occurs due to an integer overflow in the ap_pregsub function in server/util.c when the mod_setenvif module is enabled. Local users can exploit this vulnerability by crafting a .htaccess file with a SetEnvIf directive and a crafted HTTP request header. This can lead to a heap-based buffer overflow, potentially allowing the attacker to gain elevated privileges.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server integer overflow in mod_setenvif, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:

To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 37

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2016-4975**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server, related to possible CRLF injection allowing HTTP response splitting attacks for sites using mod_userdir, has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability allows for CRLF injection, which can lead to HTTP response splitting attacks. This vulnerability specifically affects sites that use mod_userdir.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can manipulate CRLF characters to inject malicious content into HTTP responses, potentially leading to various attacks such as cross-site scripting (XSS) or session hijacking.

The severity is also influenced by the fact that this vulnerability has been mitigated in Apache HTTP Server versions 2.4.25 and 2.2.32. These versions prohibit CR or LF injection into the "Location" or other outbound header key or value.

It is worth noting that the vulnerability affects Apache HTTP Server versions 2.4.1 to 2.4.23 and 2.2.0 to 2.2.31.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for HTTP response splitting attacks exists, the vulnerability has been mitigated in newer versions of Apache HTTP Server. It is important for the organization to update to a version that includes the fix for this vulnerability to prevent potential CRLF injection attacks and protect against HTTP response splitting vulnerabilities.

Public exploits related to this finding

Description of the finding

Possible CRLF injection allowing HTTP response splitting attacks for sites which use mod_userdir. This issue was mitigated by changes made in 2.4.25 and 2.2.32 which prohibit CR or LF injection into the

"Location" or other outbound header key or value. Fixed in Apache HTTP Server 2.4.25 (Affected 2.4.1-2.4.23). Fixed in Apache HTTP Server 2.2.32 (Affected 2.2.0-2.2.31).

Mitigation steps

Vulnerability: Apache HTTP Server CRLF Injection and HTTP Response Splitting

Description:
The Apache HTTP Server versions 2.4.1 through 2.4.23 and 2.2.0 through 2.2.31, when mod_userdir is enabled, have a vulnerability that allows possible CRLF (Carriage Return Line Feed) injection. This vulnerability can be exploited for HTTP response splitting attacks. However, this issue was mitigated by changes made in Apache HTTP Server versions 2.4.25 and 2.2.32. These changes prohibit CR or LF injection into the "Location" or other outbound header key or value.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server CRLF injection and HTTP response splitting, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to Apache HTTP Server version 2.4.25 or later if you are using version 2.4.x, or version 2.2.32 or later if you are using version 2.2.x. These versions include the necessary fixes to prohibit CR or LF injection into outbound headers.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent CRLF injection and other types of vulnerabilities. Validate and sanitize all user-supplied input to prevent malicious manipulation of headers.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 38

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2014-0118**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_deflate module in the Apache HTTP Server before 2.4.10 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the deflate_in_filter function in mod_deflate.c, specifically when request body decompression is enabled. It allows remote attackers to cause a denial of service by sending crafted request data that decompresses to a much larger size, leading to resource consumption.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by sending crafted request data, they can cause a denial of service by consuming excessive resources through the decompression process. This can result in unresponsiveness and potential disruption of services.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions before 2.4.10 when request body decompression is enabled.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential resource consumption and denial of service attacks.

Public exploits related to this finding

Description of the finding

The deflate_in_filter function in mod_deflate.c in the mod_deflate module in the Apache HTTP Server before 2.4.10, when request body decompression is enabled, allows remote attackers to cause a denial of service (resource consumption) via crafted request data that decompresses to a much larger size.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_deflate

Description:
The mod_deflate module in the Apache HTTP Server versions before 2.4.10 has a vulnerability that allows remote attackers to cause a denial of service (resource consumption). This vulnerability occurs when request body decompression is enabled. Attackers can exploit this vulnerability by sending crafted request data that decompresses to a much larger size. This can lead to excessive resource consumption and result in a denial of service condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_deflate, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.10.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Rate Limiting: Configure rate limiting mechanisms, such as firewall rules or network-level tools, to limit the rate at which requests can be processed by the Apache HTTP Server. This can help prevent overwhelming the server with excessive decompression requests.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on

CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 39

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2013-1896**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_dav module in the Apache HTTP Server before 2.2.25 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_dav module, specifically in the mod_dav_svn module, when determining whether DAV (WebDAV) is enabled for a URI. It allows remote attackers to cause a denial of service by sending a MERGE request in which the URI is configured for handling by mod_dav_svn, but a certain href attribute in XML data refers to a non-DAV URI. This can result in a segmentation fault and potential disruption of services.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by sending a crafted MERGE request, they can cause a segmentation fault in the Apache HTTP Server, leading to a denial of service.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions before 2.2.25.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential segmentation faults and denial of service attacks.

Public exploits related to this finding

Description of the finding

mod_dav.c in the Apache HTTP Server before 2.2.25 does not properly determine whether DAV is enabled for a URI, which allows remote attackers to cause a denial of service (segmentation fault) via a MERGE

request in which the URI is configured for handling by the mod_dav_svn module, but a certain href attribute in XML data refers to a non-DAV URI.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_dav

Description:
The Apache HTTP Server versions before 2.2.25 have a vulnerability in mod_dav.c that allows remote attackers to cause a denial of service (segmentation fault). This vulnerability occurs due to improper determination of whether DAV (WebDAV) is enabled for a URI. Attackers can exploit this vulnerability by sending a MERGE request in which the URI is configured for handling by the mod_dav_svn module, but a certain href attribute in XML data refers to a non-DAV URI. This can lead to a segmentation fault and result in a denial of service condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_dav, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.25.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

5. Limit Access to Sensitive Information: Review and restrict access to sensitive information that could potentially be leaked through this vulnerability. Ensure that appropriate access controls are in place to protect confidential data.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 40

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-4558**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerabilities in the mod_proxy_balancer module in the Apache HTTP Server 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4 have been classified as "Medium" severity due to their moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerabilities occur in the balancer_handler function in the manager interface of the mod_proxy_balancer module. They allow remote attackers to inject arbitrary web script or HTML by exploiting cross-site scripting (XSS) vulnerabilities. This can be achieved by providing a crafted string to the affected function.

The severity of these vulnerabilities is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits these vulnerabilities, they can inject arbitrary web script or HTML into the manager interface of the mod_proxy_balancer module. This can lead to various attacks, including session hijacking, defacement, or the theft of sensitive information.

The severity is also influenced by the fact that these vulnerabilities can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerabilities specifically affect Apache HTTP Server versions 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4.

In summary, the medium severity of these vulnerabilities indicates a moderate risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerabilities require remote access and do not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for these vulnerabilities to prevent potential XSS attacks and protect against unauthorized script or HTML injection.

Public exploits related to this finding

Description of the finding

Multiple cross-site scripting (XSS) vulnerabilities in the balancer_handler function in the manager interface in mod_proxy_balancer.c in the mod_proxy_balancer module in the Apache HTTP Server 2.2.x before

2.2.24-dev and 2.4.x before 2.4.4 allow remote attackers to inject arbitrary web script or HTML via a crafted string.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in mod_proxy_balancer

Description:
The Apache HTTP Server versions 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4 have multiple cross-site scripting (XSS) vulnerabilities in the balancer_handler function in the manager interface. These vulnerabilities occur in the mod_proxy_balancer module. Remote attackers can exploit these vulnerabilities by injecting arbitrary web script or HTML via a crafted string. This can lead to the execution of malicious code in the context of the affected user's browser.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in mod_proxy_balancer, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.24-dev or 2.4.4.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 41

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-3499**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerabilities in the Apache HTTP Server 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4 have been classified as "Medium" severity due to their moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerabilities occur in multiple modules of the Apache HTTP Server, including mod_imagemap, mod_info, mod_ldap, mod_proxy_ftp, and mod_status. They allow remote attackers to inject arbitrary web script or HTML by exploiting cross-site scripting (XSS) vulnerabilities. These vulnerabilities are related to vectors involving hostnames and URIs.

The severity of these vulnerabilities is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits these vulnerabilities, they can inject arbitrary web script or HTML into the affected modules. This can lead to various attacks, including session hijacking, defacement, or the theft of sensitive information.

The severity is also influenced by the fact that these vulnerabilities can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerabilities specifically affect Apache HTTP Server versions 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4.

In summary, the medium severity of these vulnerabilities indicates a moderate risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerabilities require remote access and do not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for these vulnerabilities to prevent potential XSS attacks and protect against unauthorized script or HTML injection in the affected modules.

Public exploits related to this finding

Description of the finding

Multiple cross-site scripting (XSS) vulnerabilities in the Apache HTTP Server 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4 allow remote attackers to inject arbitrary web script or HTML via vectors involving hostnames and URIs in the (1) mod_imagemap, (2) mod_info, (3) mod_ldap, (4) mod_proxy_ftp, and (5) mod_status modules.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in Multiple Modules

Description:
The Apache HTTP Server versions 2.2.x before 2.2.24-dev and 2.4.x before 2.4.4 have multiple cross-site scripting (XSS) vulnerabilities. These vulnerabilities occur in various modules, including mod_imagemap, mod_info, mod_ldap, mod_proxy_ftp, and mod_status. Remote attackers can exploit these vulnerabilities by injecting arbitrary web script or HTML via vectors involving hostnames and URIs. This can lead to the execution of malicious code in the context of the affected user's browser.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in multiple modules, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.24-dev or 2.4.4.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 42

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-0053**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server 2.2.x through 2.2.21 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the protocol.c file of the Apache HTTP Server, specifically in the construction of Bad Request (400) error documents. It does not properly restrict header information, allowing remote attackers to obtain the values of HTTPOnly cookies. This can be exploited through vectors involving a long or malformed header in conjunction with crafted web script.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability, they can obtain the values of HTTPOnly cookies, which are intended to be inaccessible to client-side scripts. This can lead to unauthorized access, session hijacking, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.2.x through 2.2.21.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for obtaining HTTPOnly cookie values exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential unauthorized access to HTTPOnly cookies.

Public exploits related to this finding

Exploit ID 18442.

- Description: Apache - httpOnly Cookie Disclosure
- Download URL: https://www.exploit-db.com/exploits/18442

Description of the finding

protocol.c in the Apache HTTP Server 2.2.x through 2.2.21 does not properly restrict header information during construction of Bad Request (aka 400) error documents, which allows remote attackers to obtain the values of HTTPOnly cookies via vectors involving a (1) long or (2) malformed header in conjunction with crafted web script.

Mitigation steps

Vulnerability: Apache HTTP Server HTTPOnly Cookie Disclosure

Description:
The Apache HTTP Server versions 2.2.x through 2.2.21 have a vulnerability that allows remote attackers to obtain the values of HTTPOnly cookies. This vulnerability occurs due to improper restriction of header information during the construction of Bad Request (400) error documents. Attackers can exploit this vulnerability by using long or malformed headers in conjunction with crafted web scripts. This can lead to the disclosure of HTTPOnly cookies, which are intended to be inaccessible to client-side scripts.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server HTTPOnly cookie disclosure, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.21.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Secure Cookie Flags: Set the "Secure" flag on cookies to ensure that they are only transmitted over secure (HTTPS) connections. This can help protect sensitive information, including session cookies, from being disclosed over insecure channels.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 43

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-4317**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy module in the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_proxy module when interacting with the use of RewriteRule and ProxyPassMatch pattern matches for configuring a reverse proxy. It allows remote attackers to send requests to intranet servers via a malformed URI containing an @ (at sign) character and a : (colon) character in invalid positions. This vulnerability exists due to an incomplete fix for CVE-2011-3368.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by sending a crafted URI, they can bypass security measures and send requests to intranet servers. This can potentially lead to unauthorized access to intranet resources and data exposure.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21 when the Revision 1179239 patch is in place.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized access and data exposure exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the complete fix for this vulnerability to prevent potential unauthorized access to intranet resources.

Public exploits related to this finding

Exploit ID 36352.

- Description: Apache 7.0.x mod_proxy - Reverse Proxy Security Bypass

- Download URL: https://www.exploit-db.com/exploits/36352

Description of the finding

The mod_proxy module in the Apache HTTP Server 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21, when the Revision 1179239 patch is in place, does not properly interact with use of (1) RewriteRule and (2) ProxyPassMatch pattern matches for configuration of a reverse proxy, which allows remote attackers to send requests to intranet servers via a malformed URI containing an @ (at sign) character and a : (colon) character in invalid positions. NOTE: this vulnerability exists because of an incomplete fix for CVE-2011-3368.

Mitigation steps

Vulnerability: Apache HTTP Server Reverse Proxy Security Bypass

Description:
The Apache HTTP Server versions 1.3.x through 1.3.42, 2.0.x through 2.0.64, and 2.2.x through 2.2.21 have a vulnerability that allows remote attackers to send requests to intranet servers via a malformed URI. This vulnerability occurs when using RewriteRule and ProxyPassMatch pattern matches for the configuration of a reverse proxy. Attackers can exploit this vulnerability by including an "@" (at sign) character and a ":" (colon) character in invalid positions in the URI. This can bypass security restrictions and allow requests to be sent to intranet servers.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server reverse proxy security bypass, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive resources. This can help prevent unauthorized access to intranet servers and limit the impact of potential vulnerabilities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 44

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-3639**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy module in the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_proxy module when interacting with the use of RewriteRule and ProxyPassMatch pattern matches for configuring a reverse proxy. It allows remote attackers to send requests to intranet servers by using the HTTP/0.9 protocol with a malformed URI containing an initial @ (at sign) character. This vulnerability exists due to an incomplete fix for CVE-2011-3368.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by sending a request using the HTTP/0.9 protocol with a malformed URI, they can bypass security measures and send requests to intranet servers. This can potentially lead to unauthorized access to intranet resources and data exposure.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x before 2.2.18 when the Revision 1179239 patch is in place.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized access and data exposure exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the complete fix for this vulnerability to prevent potential unauthorized access to intranet resources.

Public exploits related to this finding

Exploit ID 36663.

- Description: Apache 2.2.15 mod_proxy - Reverse Proxy Security Bypass

- Download URL: https://www.exploit-db.com/exploits/36663

Description of the finding

The mod_proxy module in the Apache HTTP Server 2.0.x through 2.0.64 and 2.2.x before 2.2.18, when the Revision 1179239 patch is in place, does not properly interact with use of (1) RewriteRule and (2) ProxyPassMatch pattern matches for configuration of a reverse proxy, which allows remote attackers to send requests to intranet servers by using the HTTP/0.9 protocol with a malformed URI containing an initial @ (at sign) character. NOTE: this vulnerability exists because of an incomplete fix for CVE-2011-3368.

Mitigation steps

Vulnerability: Apache HTTP Server Reverse Proxy Security Bypass

Description:
The Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x before 2.2.18 have a vulnerability that allows remote attackers to send requests to intranet servers. This vulnerability occurs when using RewriteRule and ProxyPassMatch pattern matches for the configuration of a reverse proxy. Attackers can exploit this vulnerability by using the HTTP/0.9 protocol with a malformed URI containing an initial "@" (at sign) character. This can bypass security restrictions and allow requests to be sent to intranet servers.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server reverse proxy security bypass, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.18.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive resources. This can help prevent unauthorized access to intranet servers and limit the impact of potential vulnerabilities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access

attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 45

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-0419**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified stack consumption vulnerability in the fnmatch implementation in the Apache Portable Runtime (APR) library and the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the fnmatch implementation in the APR library before version 1.4.3 and the Apache HTTP Server before version 2.2.18. It also affects libc in various operating systems such as NetBSD, OpenBSD, FreeBSD, Apple Mac OS X, Oracle Solaris, and Android. The vulnerability allows context-dependent attackers to cause a denial of service by consuming excessive CPU and memory resources through the use of *? sequences in the first argument.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by providing a specially crafted argument with *? sequences, they can cause the affected application to consume excessive CPU and memory resources. This can result in a denial of service, leading to unresponsiveness and potential disruption of services.

The severity is also influenced by the fact that this vulnerability can be exploited by context-dependent attackers. This means that the impact may vary depending on the specific context in which the vulnerable function is used.

It is worth noting that the vulnerability specifically affects the fnmatch implementation in the APR library and the Apache HTTP Server, as well as libc in various operating systems.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires specific context and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of the affected software that includes the fix for this vulnerability to prevent potential denial of service attacks through excessive CPU and memory consumption.

Public exploits related to this finding

Exploit ID 35738.

- Description: Apache 1.4/2.2.x - APR 'apr_fnmatch()' Denial of Service
- Download URL: https://www.exploit-db.com/exploits/35738

Description of the finding

Stack consumption vulnerability in the fnmatch implementation in apr_fnmatch.c in the Apache Portable Runtime (APR) library before 1.4.3 and the Apache HTTP Server before 2.2.18, and in fnmatch.c in libc in NetBSD 5.1, OpenBSD 4.8, FreeBSD, Apple Mac OS X 10.6, Oracle Solaris 10, and Android, allows context-dependent attackers to cause a denial of service (CPU and memory consumption) via *? sequences in the first argument, as demonstrated by attacks against mod_autoindex in httpd.

Mitigation steps

Vulnerability: Apache Portable Runtime (APR) Denial of Service (DoS) via fnmatch

Description:
The Apache Portable Runtime (APR) library before version 1.4.3 and the Apache HTTP Server before version 2.2.18 have a stack consumption vulnerability in the fnmatch implementation. This vulnerability also affects other systems such as NetBSD, OpenBSD, FreeBSD, Apple Mac OS X, Oracle Solaris, and Android. Context-dependent attackers can exploit this vulnerability by using *? sequences in the first argument, leading to a denial of service (CPU and memory consumption). This vulnerability can be demonstrated by attacks against mod_autoindex in the Apache HTTP Server.

Mitigation Steps:
To mitigate the risk of Apache Portable Runtime (APR) DoS via fnmatch, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.18.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Resource Limitations: Implement resource limitations, such as CPU and memory limits, to prevent excessive consumption by individual processes. This can help mitigate the impact of DoS attacks and prevent resource exhaustion.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 46

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2010-0434**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the ap_read_request function in the Apache HTTP Server 2.2.x before 2.2.15 has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the ap_read_request function in the server/protocol.c file of the Apache HTTP Server. It affects situations where a multithreaded MPM (Multi-Processing Module) is used and involves headers in subrequests when a parent request has a body. In certain circumstances, this vulnerability can allow remote attackers to obtain sensitive information by triggering access to memory locations associated with an earlier request.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by sending a crafted request, they can access memory locations associated with an earlier request, potentially leading to the disclosure of sensitive information.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects Apache HTTP Server versions 2.2.x before 2.2.15.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for sensitive information disclosure exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential disclosure of sensitive information through memory access.

Public exploits related to this finding

Description of the finding

The ap_read_request function in server/protocol.c in the Apache HTTP Server 2.2.x before 2.2.15, when a multithreaded MPM is used, does not properly handle headers in subrequests in certain circumstances

involving a parent request that has a body, which might allow remote attackers to obtain sensitive information via a crafted request that triggers access to memory locations associated with an earlier request.

Mitigation steps

Vulnerability: Apache HTTP Server Information Disclosure via Subrequests

Description:
The Apache HTTP Server versions 2.2.x before 2.2.15 have a vulnerability that allows remote attackers to obtain sensitive information. This vulnerability occurs in the ap_read_request function in server/protocol.c when a multithreaded MPM (Multi-Processing Module) is used. It does not properly handle headers in subrequests in certain circumstances involving a parent request that has a body. Attackers can exploit this vulnerability by sending a crafted request that triggers access to memory locations associated with an earlier request. This can lead to the disclosure of sensitive information.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server information disclosure via subrequests, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.15.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive resources. This can help prevent unauthorized access to sensitive information and limit the impact of potential vulnerabilities.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on

CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 47

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2008-2939**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified cross-site scripting (XSS) vulnerability in the mod_proxy_ftp module in Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the proxy_ftp.c file in the mod_proxy_ftp module in Apache HTTP Server versions 2.0.63 and earlier, as well as mod_proxy_ftp.c in Apache HTTP Server versions 2.2.9 and earlier 2.2 versions. It allows remote attackers to inject arbitrary web script or HTML by using a wildcard in the last directory component in the pathname of an FTP URI.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by crafting an FTP URI with a wildcard in the last directory component, they can inject arbitrary web script or HTML. This can lead to various attacks, including session hijacking, defacement, or the theft of sensitive information.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy_ftp module in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential XSS attacks and protect against unauthorized script or HTML injection in FTP URIs.

Public exploits related to this finding

Description of the finding

Cross-site scripting (XSS) vulnerability in proxy_ftp.c in the mod_proxy_ftp module in Apache 2.0.63 and earlier, and mod_proxy_ftp.c in the mod_proxy_ftp module in Apache 2.2.9 and earlier 2.2 versions, allows

remote attackers to inject arbitrary web script or HTML via a wildcard in the last directory component in the pathname in an FTP URI.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in mod_proxy_ftp

Description:
The Apache HTTP Server versions 2.0.63 and earlier, and 2.2.9 and earlier 2.2 versions, have a cross-site scripting (XSS) vulnerability in the mod_proxy_ftp module. This vulnerability occurs when a wildcard is used in the last directory component in the pathname of an FTP URI. Remote attackers can exploit this vulnerability to inject arbitrary web script or HTML, potentially leading to XSS attacks.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in mod_proxy_ftp, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 48

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2008-0455**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified cross-site scripting (XSS) vulnerability in the mod_negotiation module in the Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_negotiation module in Apache HTTP Server versions 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series. It allows remote authenticated users to inject arbitrary web script or HTML by uploading a file with a name containing XSS sequences and a file extension. This leads to injection within a "406 Not Acceptable" or "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an authenticated user successfully uploads a file with a specially crafted name, they can inject arbitrary web script or HTML. This can lead to various attacks, including session hijacking, defacement, or the theft of sensitive information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects the mod_negotiation module in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential XSS attacks and protect against unauthorized script or HTML injection in file uploads.

Public exploits related to this finding

Exploit ID 31052.

- Description: Apache 2.2.6 mod_negotiation - HTML Injection / HTTP Response Splitting

- Download URL: https://www.exploit-db.com/exploits/31052

Description of the finding

Cross-site scripting (XSS) vulnerability in the mod_negotiation module in the Apache HTTP Server 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series allows remote authenticated users to inject arbitrary web script or HTML by uploading a file with a name containing XSS sequences and a file extension, which leads to injection within a (1) "406 Not Acceptable" or (2) "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in mod_negotiation

Description:
The Apache HTTP Server versions 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series have a cross-site scripting (XSS) vulnerability in the mod_negotiation module. This vulnerability allows remote authenticated users to inject arbitrary web script or HTML by uploading a file with a name containing XSS sequences and a file extension. This leads to injection within a "406 Not Acceptable" or "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in mod_negotiation, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log

Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 49

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2008-0005**
- Severity score: **4.3** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the mod_proxy_ftp module in Apache HTTP Server has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in the mod_proxy_ftp module in Apache HTTP Server versions 2.2.x before 2.2.7-dev, 2.0.x before 2.0.62-dev, and 1.3.x before 1.3.40-dev. It is related to the module not defining a charset, which allows remote attackers to conduct cross-site scripting (XSS) attacks using UTF-7 encoding.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits this vulnerability by using UTF-7 encoding, they can conduct cross-site scripting attacks. This can lead to various attacks, including session hijacking, defacement, or the theft of sensitive information.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy_ftp module in Apache HTTP Server.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential XSS attacks and protect against unauthorized script or HTML injection using UTF-7 encoding.

Public exploits related to this finding

Description of the finding

mod_proxy_ftp in Apache 2.2.x before 2.2.7-dev, 2.0.x before 2.0.62-dev, and 1.3.x before 1.3.40-dev does not define a charset, which allows remote attackers to conduct cross-site scripting (XSS) attacks using UTF-7 encoding.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in mod_proxy_ftp

Description:
The Apache HTTP Server versions 2.2.x before 2.2.7-dev, 2.0.x before 2.0.62-dev, and 1.3.x before 1.3.40-dev have a cross-site scripting (XSS) vulnerability in the mod_proxy_ftp module. This vulnerability occurs because mod_proxy_ftp does not define a charset. Remote attackers can exploit this vulnerability by using UTF-7 encoding to conduct XSS attacks.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in mod_proxy_ftp, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
```

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 50

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2012-2687**
- Severity score: **2.6** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified multiple cross-site scripting (XSS) vulnerabilities in the mod_negotiation module in the Apache HTTP Server have been classified as "Low" severity due to their low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerabilities occur in the make_variant_list function in mod_negotiation.c in the mod_negotiation module in Apache HTTP Server versions 2.4.x before 2.4.3 when the MultiViews option is enabled. They allow remote attackers to inject arbitrary web script or HTML by using a crafted filename that is not properly handled during the construction of a variant list.

The severity of these vulnerabilities is determined by the potential consequences of successful exploitation. In this case, if an attacker successfully exploits these vulnerabilities by providing a specially crafted filename, they can inject arbitrary web script or HTML. However, the impact of these vulnerabilities is limited due to the low severity rating.

The severity is also influenced by the fact that these vulnerabilities can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerabilities specifically affect the mod_negotiation module in Apache HTTP Server.

In summary, the low severity of these vulnerabilities indicates a low risk to the organization's systems and data. While the potential for XSS attacks exists, the vulnerabilities have a limited impact and do not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for these vulnerabilities to prevent potential XSS attacks and protect against unauthorized script or HTML injection in filenames.

Public exploits related to this finding

Description of the finding

Multiple cross-site scripting (XSS) vulnerabilities in the make_variant_list function in mod_negotiation.c in the mod_negotiation module in the Apache HTTP Server 2.4.x before 2.4.3, when the MultiViews option is

enabled, allow remote attackers to inject arbitrary web script or HTML via a crafted filename that is not properly handled during construction of a variant list.

Mitigation steps

Vulnerability: Apache HTTP Server Cross-Site Scripting (XSS) in mod_negotiation

Description:
The Apache HTTP Server versions 2.4.x before 2.4.3 have multiple cross-site scripting (XSS) vulnerabilities in the mod_negotiation module. These vulnerabilities occur when the MultiViews option is enabled. Remote attackers can exploit these vulnerabilities by injecting arbitrary web script or HTML via a crafted filename that is not properly handled during the construction of a variant list.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server XSS in mod_negotiation, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.4.3.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent XSS vulnerabilities. Validate and sanitize all user-supplied input to prevent the execution of malicious scripts or HTML.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 51

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2009-3094**
- Severity score: **2.6** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the ap_proxy_ftp_handler function in the mod_proxy_ftp module in the Apache HTTP Server has been classified as "Low" severity due to its low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability occurs in the ap_proxy_ftp_handler function in the mod_proxy_ftp module in Apache HTTP Server versions 2.0.63 and 2.2.13. It allows remote FTP servers to cause a denial of service by triggering a NULL pointer dereference and crashing the child process. This can be achieved by sending a malformed reply to an EPSV command.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote FTP server sends a specially crafted reply to an EPSV command, it can trigger a NULL pointer dereference and crash the child process. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects the mod_proxy_ftp module in Apache HTTP Server.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. While the potential for a denial of service exists, the vulnerability has a limited impact and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential crashes and denial of service caused by malformed FTP replies.

Public exploits related to this finding

Description of the finding

The ap_proxy_ftp_handler function in modules/proxy/proxy_ftp.c in the mod_proxy_ftp module in the Apache HTTP Server 2.0.63 and 2.2.13 allows remote FTP servers to cause a denial of service (NULL pointer dereference and child process crash) via a malformed reply to an EPSV command.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) in mod_proxy_ftp

Description:
The Apache HTTP Server versions 2.0.63 and 2.2.13 have a vulnerability in the mod_proxy_ftp module. This vulnerability occurs in the ap_proxy_ftp_handler function in modules/proxy/proxy_ftp.c. Remote FTP servers can exploit this vulnerability by sending a malformed reply to an EPSV command. This can cause a NULL pointer dereference and crash the child process, resulting in a denial of service (DoS) condition.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS in mod_proxy_ftp, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.13.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
```

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 52

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2008-0456**
- Severity score: **2.6** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified CRLF injection vulnerability in the mod_negotiation module in the Apache HTTP Server has been classified as "Low" severity due to its low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability occurs in the mod_negotiation module in Apache HTTP Server versions 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series. It allows remote authenticated users to inject arbitrary HTTP headers and conduct HTTP response splitting attacks by uploading a file with a multi-line name containing HTTP header sequences and a file extension. This leads to injection within a "406 Not Acceptable" or "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully uploads a file with a specially crafted name, they can inject arbitrary HTTP headers and conduct HTTP response splitting attacks. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects the mod_negotiation module in Apache HTTP Server.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. While the potential for HTTP response splitting attacks exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential HTTP response splitting attacks and protect against unauthorized HTTP header injection.

Public exploits related to this finding

Description of the finding

CRLF injection vulnerability in the mod_negotiation module in the Apache HTTP Server 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series allows remote authenticated users to inject arbitrary HTTP headers and conduct HTTP response splitting attacks by uploading a file with a multi-line name containing HTTP header sequences and a file extension, which leads to injection within a (1) "406 Not Acceptable" or (2) "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

Mitigation steps

Vulnerability: Apache HTTP Server CRLF Injection and HTTP Response Splitting in mod_negotiation

Description:
The Apache HTTP Server versions 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series have a CRLF injection vulnerability in the mod_negotiation module. This vulnerability allows remote authenticated users to inject arbitrary HTTP headers and conduct HTTP response splitting attacks. Attackers can exploit this vulnerability by uploading a file with a multi-line name containing HTTP header sequences and a file extension. This leads to injection within a "406 Not Acceptable" or "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server CRLF injection and HTTP response splitting in mod_negotiation, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent CRLF injection and HTTP response splitting vulnerabilities. Validate and sanitize all user-supplied input to prevent the injection of malicious headers.

4. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log

Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 53

- Affected IP: **192.168.203.129**
- Affected port and service: **80** (service **http**)
- Related CVE: **CVE-2011-4415**
- Severity score: **1.2** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the Apache HTTP Server has been classified as "Low" severity due to its low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability occurs in the ap_pregsub function in server/util.c in Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x through 2.2.21 when the mod_setenvif module is enabled. It relates to the unrestricted size of values of environment variables, allowing local users to cause a denial of service through memory consumption or a NULL pointer dereference. This can be achieved by using a crafted .htaccess file with a SetEnvIf directive in conjunction with a crafted HTTP request header.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability by using a crafted .htaccess file and HTTP request header, they can cause a denial of service by consuming excessive memory or triggering a NULL pointer dereference. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires local access. This means that an attacker needs to have direct access to the organization's systems, reducing the risk of opportunistic attacks. However, it still poses a threat if a local user with malicious intent is present.

It is worth noting that the vulnerability specifically affects the Apache HTTP Server when the mod_setenvif module is enabled.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Apache HTTP Server that includes the fix for this vulnerability to prevent potential denial of service attacks through excessive memory consumption or NULL pointer dereference.

Public exploits related to this finding

Exploit ID 41769.

- Description: Apache < 2.0.64 / < 2.2.21 mod_setenvif - Integer Overflow

- Download URL: https://www.exploit-db.com/exploits/41769

Description of the finding

The ap_pregsub function in server/util.c in the Apache HTTP Server 2.0.x through 2.0.64 and 2.2.x through 2.2.21, when the mod_setenvif module is enabled, does not restrict the size of values of environment variables, which allows local users to cause a denial of service (memory consumption or NULL pointer dereference) via a .htaccess file with a crafted SetEnvIf directive, in conjunction with a crafted HTTP request header, related to (1) the "len +=" statement and (2) the apr_pcalloc function call, a different vulnerability than CVE-2011-3607.

Mitigation steps

Vulnerability: Apache HTTP Server Denial of Service (DoS) via mod_setenvif

Description:
The Apache HTTP Server versions 2.0.x through 2.0.64 and 2.2.x through 2.2.21 have a vulnerability in the ap_pregsub function in server/util.c when the mod_setenvif module is enabled. This vulnerability allows local users to cause a denial of service (memory consumption or NULL pointer dereference). Attackers can exploit this vulnerability by using a crafted .htaccess file with a SetEnvIf directive, in conjunction with a crafted HTTP request header. This vulnerability is related to the "len +=" statement and the apr_pcalloc function call.

Mitigation Steps:
To mitigate the risk of Apache HTTP Server DoS via mod_setenvif, the following steps can be taken:

1. Update Apache HTTP Server: Upgrade to a version of Apache HTTP Server that includes a fix for this vulnerability. Ensure that you are using a version later than 2.2.21.

2. Apply Patches: If an official patch is available for your version of Apache HTTP Server, apply it to fix the vulnerability. Patches are typically provided by the Apache Software Foundation or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep Apache HTTP Server and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Input Validation: Implement strict input validation mechanisms in your web application to prevent DoS vulnerabilities. Validate and sanitize all user-supplied input to prevent excessive memory consumption or NULL pointer dereference.

5. Monitor and Log HTTP Server Activity: Implement a centralized logging system to monitor and log

Apache HTTP Server activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Example Command:
To update Apache HTTP Server, follow these steps:

1. Identify the package manager used by your operating system. For example, on Ubuntu, it is apt, and on CentOS, it is yum.

2. Update the package manager's repository information:

```
sudo apt update
```

or

```
sudo yum update
```

3. Upgrade the Apache HTTP Server package:

```
sudo apt upgrade apache2
```

or

```
sudo yum upgrade httpd
```

Replace "apache2" or "httpd" with the appropriate package name for your operating system.

4. Follow the prompts to complete the upgrade process.

Note: The specific steps may vary depending on the operating system and package manager being used. It is important to consult the documentation and best practices for your specific environment.

Finding 54

- Affected IP: **192.168.203.129**
- Affected port and service: **139** (service **netbios-ssn**)
- Related CVE: **CVE-2017-0144**
- Severity score: **8.1** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the SMBv1 server in Microsoft Windows has been classified as "High" severity due to its significant impact on the organization's security. A high severity vulnerability indicates a high risk to the organization's systems and data.

The vulnerability affects various versions of Microsoft Windows, including Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8.1, Windows Server 2012 Gold and R2, Windows RT 8.1, Windows 10 Gold, 1511, and 1607, and Windows Server 2016. It allows remote attackers to execute arbitrary code by sending crafted packets to the SMBv1 server.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully exploits this vulnerability by sending crafted packets, they can execute arbitrary code on the affected Windows system. This can lead to complete compromise of the system, unauthorized access, data theft, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that this vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.

In summary, the high severity of this vulnerability indicates a significant risk to the organization's systems and data. The potential for remote code execution by an attacker poses a serious threat to the security and integrity of the affected Windows systems. It is crucial for the organization to address and remediate the issue by applying the necessary security updates or patches provided by Microsoft to mitigate the risk of exploitation and potential unauthorized code execution.

Public exploits related to this finding

Exploit ID 41891.

- Description: Microsoft Windows - SMB Remote Code Execution Scanner (MS17-010) (Metasploit)

- Download URL: https://www.exploit-db.com/exploits/41891

Exploit ID 41987.

- Description: Microsoft Windows Server 2008 R2 (x64) - 'SrvOs2FeaToNt' SMB Remote Code Execution (MS17-010)
- Download URL: https://www.exploit-db.com/exploits/41987

Exploit ID 42030.

- Description: Microsoft Windows 8/8.1/2012 R2 (x64) - 'EternalBlue' SMB Remote Code Execution (MS17-010)
- Download URL: https://www.exploit-db.com/exploits/42030

Exploit ID 42031.

- Description: Microsoft Windows 7/2008 R2 - 'EternalBlue' SMB Remote Code Execution (MS17-010)
- Download URL: https://www.exploit-db.com/exploits/42031

Exploit ID 42315.

- Description: Microsoft Windows 7/8.1/2008 R2/2012 R2/2016 R2 - 'EternalBlue' SMB Remote Code Execution (MS17-010)
- Download URL: https://www.exploit-db.com/exploits/42315

Exploit ID 47456.

- Description: DOUBLEPULSAR - Payload Execution and Neutralization (Metasploit)
- Download URL: https://www.exploit-db.com/exploits/47456

Description of the finding

The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted

packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.

Mitigation steps

Vulnerability: Microsoft Windows SMB Remote Code Execution

Description:
The SMBv1 server in various versions of Microsoft Windows, including Windows Vista, Windows Server 2008, Windows 7, Windows 8.1, Windows Server 2012, Windows RT 8.1, Windows 10, and Windows Server 2016, is vulnerable to remote code execution. This vulnerability allows remote attackers to execute arbitrary code by sending crafted packets to the SMBv1 server. The vulnerability is known as "Windows SMB Remote Code Execution Vulnerability" and is assigned CVE-2017-0144. It is different from other vulnerabilities described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.

Mitigation Steps:
To mitigate the risk of Microsoft Windows SMB Remote Code Execution, the following steps can be taken:

1. Apply Security Updates: Install the security updates provided by Microsoft for the affected versions of Windows. These updates address the vulnerability and prevent remote code execution.

2. Disable SMBv1: Consider disabling the SMBv1 protocol on Windows systems, as it is known to have multiple vulnerabilities. Disabling SMBv1 can help protect against future vulnerabilities and attacks.

3. Use Network Segmentation: Implement network segmentation to isolate critical systems and limit the exposure of vulnerable systems to potential attacks.

4. Implement Intrusion Detection and Prevention Systems: Deploy intrusion detection and prevention systems (IDPS) to monitor network traffic and detect any attempts to exploit the vulnerability.

5. Educate Users: Educate users about the risks of opening suspicious email attachments or clicking on links from unknown sources. Social engineering attacks can be used to deliver malware that exploits this vulnerability.

Note: It is important to consult the official documentation and follow best practices provided by Microsoft for applying security updates and securing Windows systems.

Finding 55

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2009-2446**
- Severity score: **8.5** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified multiple format string vulnerabilities in the dispatch_command function in libmysqld/sql_parse.cc in mysqld in MySQL have been classified as "High" severity due to their significant impact on the organization's security. A high severity vulnerability indicates a high risk to the organization's systems and data.

The vulnerabilities occur in MySQL versions 4.0.0 through 5.0.83 and allow remote authenticated users to cause a denial of service by crashing the MySQL daemon. These vulnerabilities are caused by format string specifiers in a database name in COM_CREATE_DB or COM_DROP_DB requests.

The severity of these vulnerabilities is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits these vulnerabilities by using format string specifiers in a database name, they can cause a denial of service by crashing the MySQL daemon. Additionally, there may be other unspecified impacts that could potentially lead to further compromise or disruption.

The severity is also influenced by the fact that these vulnerabilities can be exploited by remote authenticated users. This means that an attacker needs valid credentials to exploit the vulnerabilities, reducing the risk of opportunistic attacks. However, it still poses a significant threat if an authenticated user with malicious intent is present.

It is worth noting that some of the details of these vulnerabilities may be obtained from third-party information.

In summary, the high severity of these vulnerabilities indicates a significant risk to the organization's systems and data. The potential for denial of service and other impacts caused by format string vulnerabilities poses a serious threat to the security and availability of the MySQL database. It is crucial for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for these vulnerabilities to prevent potential crashes and unauthorized access or manipulation of the database.

Public exploits related to this finding

Exploit ID 33077.

- Description: MySQL 5.0.75 - 'sql_parse.cc' Multiple Format String Vulnerabilities
- Download URL: https://www.exploit-db.com/exploits/33077

Description of the finding

Multiple format string vulnerabilities in the dispatch_command function in libmysqld/sql_parse.cc in mysqld in MySQL 4.0.0 through 5.0.83 allow remote authenticated users to cause a denial of service (daemon crash) and possibly have unspecified other impact via format string specifiers in a database name in a (1) COM_CREATE_DB or (2) COM_DROP_DB request. NOTE: some of these details are obtained from third party information.

Mitigation steps

Vulnerability: MySQL Multiple Format String Vulnerabilities

Description:
MySQL versions 4.0.0 through 5.0.83 have multiple format string vulnerabilities in the dispatch_command function in libmysqld/sql_parse.cc. These vulnerabilities allow remote authenticated users to cause a denial of service (daemon crash) and potentially have other unspecified impacts. The vulnerabilities occur when format string specifiers are used in a database name in a COM_CREATE_DB or COM_DROP_DB request.

Mitigation Steps:
To mitigate the risk of MySQL multiple format string vulnerabilities, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.0.83.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 56

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2008-0226**
- Severity score: **7.5** (**HIGH** severity)

Finding severity rationale

Severity Rationale:

The identified multiple buffer overflows in yaSSL have been classified as "High" severity due to their significant impact on the organization's security. A high severity vulnerability indicates a high risk to the organization's systems and data.

The buffer overflows occur in yaSSL versions 1.7.5 and earlier, which is used in MySQL and potentially other products. These vulnerabilities allow remote attackers to execute arbitrary code by exploiting the ProcessOldClientHello function in handshake.cpp or the "input_buffer& operator>>" function in yassl_imp.cpp.

The severity of these vulnerabilities is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully exploits these buffer overflows, they can execute arbitrary code on the affected system. This can lead to complete compromise of the system, unauthorized access, data theft, or other malicious activities.

The severity is also influenced by the fact that these vulnerabilities can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that these vulnerabilities specifically affect yaSSL, which is used in MySQL and potentially other products.

In summary, the high severity of these vulnerabilities indicates a significant risk to the organization's systems and data. The potential for remote code execution by an attacker poses a serious threat to the security and integrity of the affected systems. It is crucial for the organization to address and remediate the issue by updating to a version of yaSSL that includes the fix for these vulnerabilities or by applying any patches or updates provided by the vendor to mitigate the risk of exploitation and potential unauthorized code execution.

Public exploits related to this finding

Exploit ID 16701.

- Description: MySQL yaSSL (Windows) - SSL Hello Message Buffer Overflow (Metasploit)
- Download URL: https://www.exploit-db.com/exploits/16701

Exploit ID 16849.

- Description: MySQL yaSSL (Linux) - SSL Hello Message Buffer Overflow (Metasploit)
- Download URL: https://www.exploit-db.com/exploits/16849

Exploit ID 9953.

- Description: MySQL 6.0 yaSSL 1.7.5 - Hello Message Buffer Overflow (Metasploit)
- Download URL: https://www.exploit-db.com/exploits/9953

Description of the finding

Multiple buffer overflows in yaSSL 1.7.5 and earlier, as used in MySQL and possibly other products, allow remote attackers to execute arbitrary code via (1) the ProcessOldClientHello function in handshake.cpp or (2) "input_buffer& operator>>" in yassl_imp.cpp.

Mitigation steps

Vulnerability: yaSSL Buffer Overflow in MySQL

Description:
yaSSL versions 1.7.5 and earlier, as used in MySQL and possibly other products, have multiple buffer overflow vulnerabilities. These vulnerabilities allow remote attackers to execute arbitrary code. The vulnerabilities exist in the ProcessOldClientHello function in handshake.cpp and the "input_buffer& operator>>" in yassl_imp.cpp.

Mitigation Steps:
To mitigate the risk of yaSSL buffer overflow vulnerabilities in MySQL, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to

sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 57

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2009-5026**
- Severity score: **6.8** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0.x before 5.0.93 and 5.1.x before 5.1.50 and involves the executable comment feature. In certain slave configurations where the slave is running a newer version than the master, remote attackers can execute arbitrary SQL commands by using custom comments.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully exploits this vulnerability by using custom comments, they can execute arbitrary SQL commands. This can lead to unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0.x before 5.0.93 and 5.1.x before 5.1.50.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for arbitrary SQL command execution exists, the vulnerability requires specific slave configurations and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential unauthorized SQL command execution and protect against malicious activities.

Public exploits related to this finding

Exploit ID 34796.

- Description: Oracle MySQL < 5.1.50 - Privilege Escalation
- Download URL: https://www.exploit-db.com/exploits/34796

Description of the finding

The executable comment feature in MySQL 5.0.x before 5.0.93 and 5.1.x before 5.1.50, when running in certain slave configurations in which the slave is running a newer version than the master, allows remote attackers to execute arbitrary SQL commands via custom comments.

Mitigation steps

Vulnerability: MySQL Executable Comment Arbitrary SQL Execution

Description:
MySQL versions 5.0.x before 5.0.93 and 5.1.x before 5.1.50 have a vulnerability in the executable comment feature. In certain slave configurations where the slave is running a newer version than the master, remote attackers can execute arbitrary SQL commands via custom comments. This can lead to privilege escalation and unauthorized access to the database.

Mitigation Steps:
To mitigate the risk of MySQL executable comment arbitrary SQL execution, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.50 or 5.0.93, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 58

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2009-4028**
- Severity score: **6.8** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41 when OpenSSL is used. It involves the vio_verify_callback function in viosslfactories.c and the acceptance of a value of zero for the depth of X.509 certificates. This allows man-in-the-middle attackers to spoof SSL-based MySQL servers by presenting a crafted certificate, particularly when the server is linked against the yaSSL library.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a man-in-the-middle attacker successfully exploits this vulnerability by presenting a crafted certificate, they can spoof SSL-based MySQL servers. This can lead to unauthorized access, data manipulation, or other malicious activities.

The severity is also influenced by the fact that this vulnerability can be exploited by man-in-the-middle attackers. This means that an attacker can intercept and modify the communication between the client and the MySQL server, increasing the risk of unauthorized access or data manipulation.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41 when OpenSSL is used.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for man-in-the-middle attacks exists, the vulnerability requires specific conditions and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential SSL spoofing and protect against unauthorized access or data manipulation.

Public exploits related to this finding

Description of the finding

The vio_verify_callback function in viosslfactories.c in MySQL 5.0.x before 5.0.88 and 5.1.x before 5.1.41, when OpenSSL is used, accepts a value of zero for the depth of X.509 certificates, which allows man-in-the-middle attackers to spoof arbitrary SSL-based MySQL servers via a crafted certificate, as demonstrated by a certificate presented by a server linked against the yaSSL library.

Mitigation steps

Vulnerability: MySQL SSL Certificate Spoofing

Description:
MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41, when OpenSSL is used, have a vulnerability in the vio_verify_callback function in viosslfactories.c. This vulnerability allows man-in-the-middle attackers to spoof arbitrary SSL-based MySQL servers by presenting a crafted certificate. The issue arises from the acceptance of a value of zero for the depth of X.509 certificates, which should not be allowed.

Mitigation Steps:
To mitigate the risk of MySQL SSL certificate spoofing, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.41 or 5.0.88, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Secure Certificate Practices: Follow best practices for SSL certificate management. Ensure that certificates are obtained from trusted sources and properly validated.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 59

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-1848**
- Severity score: **6.5** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified directory traversal vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 and involves a directory traversal issue. It allows remote authenticated users to bypass intended table grants and read field definitions of arbitrary tables. In MySQL 5.1, it also allows reading or deleting the content of arbitrary tables. This can be achieved by using a ".." (dot dot) in a table name.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this vulnerability by using a directory traversal technique in a table name, they can bypass table grants and read field definitions or, in the case of MySQL 5.1, read or delete the content of arbitrary tables. This can lead to unauthorized access, data exposure, or data manipulation.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for unauthorized access and data exposure or manipulation exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential unauthorized access to table definitions and protect against unauthorized data exposure or manipulation.

Public exploits related to this finding

Description of the finding

Directory traversal vulnerability in MySQL 5.0 through 5.0.91 and 5.1 before 5.1.47 allows remote authenticated users to bypass intended table grants to read field definitions of arbitrary tables, and on 5.1 to read or delete content of arbitrary tables, via a .. (dot dot) in a table name.

Mitigation steps

Vulnerability: MySQL Directory Traversal

Description:
MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 have a directory traversal vulnerability. This vulnerability allows remote authenticated users to bypass intended table grants and read field definitions of arbitrary tables. In MySQL 5.1, it also allows reading or deleting the content of arbitrary tables. The vulnerability is exploited by using ".." (dot dot) in a table name.

Mitigation Steps:
To mitigate the risk of MySQL directory traversal, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.47 or 5.0.91, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 60

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-1850**
- Severity score: **6.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified buffer overflow vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 and involves a buffer overflow issue. It allows remote authenticated users to execute arbitrary code by sending a COM_FIELD_LIST command with a long table name.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this buffer overflow vulnerability by sending a specially crafted COM_FIELD_LIST command, they can execute arbitrary code on the affected MySQL server. This can lead to complete compromise of the system, unauthorized access, data theft, or other malicious activities.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for arbitrary code execution exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential arbitrary code execution and protect against unauthorized access or manipulation of the database.

Public exploits related to this finding

Description of the finding

Buffer overflow in MySQL 5.0 through 5.0.91 and 5.1 before 5.1.47 allows remote authenticated users to execute arbitrary code via a COM_FIELD_LIST command with a long table name.

Mitigation steps

Vulnerability: MySQL Buffer Overflow

Description:
MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 have a buffer overflow vulnerability. This vulnerability allows remote authenticated users to execute arbitrary code by sending a COM_FIELD_LIST command with a long table name. By exploiting this vulnerability, an attacker can execute arbitrary code within the context of the MySQL server.

Mitigation Steps:
To mitigate the risk of MySQL buffer overflow, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.47 or 5.0.91, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 61

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2008-7247**
- Severity score: **6.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0.x through 5.0.88, 5.1.x through 5.1.41, and 6.0 before 6.0.9-alpha and involves a symlink following issue. When the data home directory contains a symlink to a different filesystem, remote authenticated users can bypass intended access restrictions by calling CREATE TABLE with a DATA DIRECTORY or INDEX DIRECTORY argument that refers to a subdirectory requiring following this symlink.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this vulnerability by using a symlink to a different filesystem and calling CREATE TABLE with a specific argument, they can bypass intended access restrictions. This can lead to unauthorized access to data or other unintended consequences.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0.x through 5.0.88, 5.1.x through 5.1.41, and 6.0 before 6.0.9-alpha.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for bypassing access restrictions exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential unauthorized access and protect against unintended consequences caused by symlink following.

Public exploits related to this finding

Description of the finding

sql/sql_table.cc in MySQL 5.0.x through 5.0.88, 5.1.x through 5.1.41, and 6.0 before 6.0.9-alpha, when the data home directory contains a symlink to a different filesystem, allows remote authenticated users to bypass intended access restrictions by calling CREATE TABLE with a (1) DATA DIRECTORY or (2) INDEX DIRECTORY argument referring to a subdirectory that requires following this symlink.

Mitigation steps

Vulnerability: MySQL Symlink Bypass

Description:
MySQL versions 5.0.x through 5.0.88, 5.1.x through 5.1.41, and 6.0 before 6.0.9-alpha have a vulnerability that allows remote authenticated users to bypass intended access restrictions. This vulnerability occurs when the data home directory contains a symlink to a different filesystem. By calling CREATE TABLE with a DATA DIRECTORY or INDEX DIRECTORY argument that refers to a subdirectory requiring following the symlink, an attacker can bypass access restrictions and gain unauthorized access to data.

Mitigation Steps:
To mitigate the risk of MySQL symlink bypass, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 6.0.9-alpha, 5.1.41, or 5.0.88, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 62

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3833**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 and involves the propagation of type errors. It allows remote attackers to cause a denial of service by crashing the server through crafted arguments to extreme-value functions such as LEAST and GREATEST. This vulnerability is related to KILL_BAD_DATA and a "CREATE TABLE ... SELECT" operation.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully exploits this vulnerability by providing crafted arguments to extreme-value functions, they can cause a denial of service by crashing the MySQL server. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires specific conditions and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential server crashes caused by crafted arguments to extreme-value functions.

Public exploits related to this finding

Description of the finding

MySQL 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 does not properly propagate type errors, which allows remote attackers to cause a denial of service (server crash) via crafted arguments to extreme-value functions such as (1) LEAST and (2) GREATEST, related to KILL_BAD_DATA and a "CREATE TABLE ... SELECT."

Mitigation steps

Vulnerability: MySQL Denial of Service via Type Errors

Description:
MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 have a vulnerability that allows remote attackers to cause a denial of service (server crash). This vulnerability occurs due to the improper propagation of type errors. Attackers can exploit this vulnerability by providing crafted arguments to extreme-value functions such as LEAST and GREATEST. This vulnerability is related to the KILL_BAD_DATA feature and the "CREATE TABLE ... SELECT" statement.

Mitigation Steps:
To mitigate the risk of MySQL denial of service via type errors, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.5.6, 5.1.51, or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 63

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-1849**
- Severity score: **5.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 and involves the my_net_skip_rest function in sql/net_serv.cc. It allows remote attackers to cause a denial of service by consuming CPU and bandwidth resources. This can be achieved by sending a large number of packets that exceed the maximum length.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully exploits this vulnerability by sending a large number of packets that exceed the maximum length, they can cause a denial of service by consuming CPU and bandwidth resources. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires remote access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential denial of service attacks through excessive CPU and bandwidth consumption.

Public exploits related to this finding

Description of the finding

The my_net_skip_rest function in sql/net_serv.cc in MySQL 5.0 through 5.0.91 and 5.1 before 5.1.47 allows remote attackers to cause a denial of service (CPU and bandwidth consumption) by sending a large number of packets that exceed the maximum length.

Mitigation steps

Vulnerability: MySQL Denial of Service via Packet Overflow

Description:
MySQL versions 5.0 through 5.0.91 and 5.1 before 5.1.47 have a vulnerability that allows remote attackers to cause a denial of service (CPU and bandwidth consumption). This vulnerability occurs in the my_net_skip_rest function in sql/net_serv.cc. Attackers can exploit this vulnerability by sending a large number of packets that exceed the maximum length, leading to excessive CPU and bandwidth consumption.

Mitigation Steps:
To mitigate the risk of MySQL denial of service via packet overflow, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.47 or 5.0.91, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 64

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2021-2356**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I'm sorry, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2021-2356". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Can you please double-check the identifier or provide more details about the vulnerability?

Finding 65

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2023-21980**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I apologize, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2023-21980". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Please ensure the correctness of the identifier or provide additional details if available.

Finding 66

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2008-4098**
- Severity score: **4.6** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions before 5.0.67 and involves privilege checks bypass. It allows local users to bypass certain privilege checks by calling CREATE TABLE on a MyISAM table with modified DATA DIRECTORY or INDEX DIRECTORY arguments. These arguments are originally associated with pathnames without symlinks but can be modified to contain a symlink to a subdirectory of the MySQL home data directory at a future time.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability by calling CREATE TABLE with modified arguments, they can bypass certain privilege checks. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires local access. This means that an attacker needs to have direct access to the organization's systems, reducing the risk of opportunistic attacks. However, it still poses a threat if a local user with malicious intent is present.

It is worth noting that the vulnerability exists due to an incomplete fix for CVE-2008-4097.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for privilege checks bypass exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the complete fix for this vulnerability to prevent potential privilege escalation and protect against unauthorized access or manipulation of the database.

Public exploits related to this finding

Description of the finding

MySQL before 5.0.67 allows local users to bypass certain privilege checks by calling CREATE TABLE on a MyISAM table with modified (1) DATA DIRECTORY or (2) INDEX DIRECTORY arguments that are originally associated with pathnames without symlinks, and that can point to tables created at a future time at which a pathname is modified to contain a symlink to a subdirectory of the MySQL home data directory. NOTE: this vulnerability exists because of an incomplete fix for CVE-2008-4097.

Mitigation steps

Vulnerability: MySQL Privilege Bypass via CREATE TABLE

Description:
MySQL versions before 5.0.67 have a vulnerability that allows local users to bypass certain privilege checks. This vulnerability occurs when a local user calls CREATE TABLE on a MyISAM table with modified DATA DIRECTORY or INDEX DIRECTORY arguments. These arguments are originally associated with pathnames without symlinks but can be modified to contain a symlink to a subdirectory of the MySQL home data directory. By exploiting this vulnerability, a local user can bypass privilege checks and gain unauthorized access to tables created at a future time.

Mitigation Steps:
To mitigate the risk of MySQL privilege bypass via CREATE TABLE, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.0.67.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 67

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2008-2079**
- Severity score: **4.6** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 4.1.x before 4.1.24, 5.0.x before 5.0.60, 5.1.x before 5.1.24, and 6.0.x before 6.0.5. It involves privilege check bypass. Local users can bypass certain privilege checks by calling CREATE TABLE on a MyISAM table with modified DATA DIRECTORY or INDEX DIRECTORY arguments. These modified arguments are within the MySQL home data directory and can point to tables that are created in the future.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability by calling CREATE TABLE with modified arguments, they can bypass certain privilege checks. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires local access. This means that an attacker needs to have direct access to the organization's systems, reducing the risk of opportunistic attacks. However, it still poses a threat if a local user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 4.1.x before 4.1.24, 5.0.x before 5.0.60, 5.1.x before 5.1.24, and 6.0.x before 6.0.5.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for privilege check bypass exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential privilege escalation and protect against unauthorized access or manipulation of the database.

Public exploits related to this finding

Description of the finding

MySQL 4.1.x before 4.1.24, 5.0.x before 5.0.60, 5.1.x before 5.1.24, and 6.0.x before 6.0.5 allows local users to bypass certain privilege checks by calling CREATE TABLE on a MyISAM table with modified (1) DATA DIRECTORY or (2) INDEX DIRECTORY arguments that are within the MySQL home data directory, which can point to tables that are created in the future.

Mitigation steps

Vulnerability: MySQL Privilege Bypass via CREATE TABLE

Description:
MySQL versions 4.1.x before 4.1.24, 5.0.x before 5.0.60, 5.1.x before 5.1.24, and 6.0.x before 6.0.5 have a vulnerability that allows local users to bypass certain privilege checks. This vulnerability occurs when a local user calls CREATE TABLE on a MyISAM table with modified DATA DIRECTORY or INDEX DIRECTORY arguments. These arguments are within the MySQL home data directory and can point to tables that are created in the future. By exploiting this vulnerability, a local user can bypass privilege checks and gain unauthorized access to tables created in the future.

Mitigation Steps:
To mitigate the risk of MySQL privilege bypass via CREATE TABLE, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 6.0.5, 5.1.24, 5.0.60, or 4.1.24, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 68

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2022-21417**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I apologize, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2022-21417". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Please ensure the correctness of the identifier or provide additional details if available.

Finding 69

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0490**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x, 5.1.x, and 5.5.x. It allows remote authenticated users to affect availability, but the specific vectors or details of the vulnerability are not disclosed.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the availability of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the availability of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x, 5.1.x, and 5.5.x allows remote authenticated users to affect availability via unknown vectors.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect availability, although the specific vectors or details of the vulnerability are not disclosed.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 70

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0484**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x, 5.1.x, and 5.5.x. It allows remote authenticated users to affect confidentiality, but the specific vectors or details of the vulnerability are not disclosed.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the confidentiality of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the confidentiality of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x, 5.1.x, and 5.5.x allows remote authenticated users to affect confidentiality via unknown vectors.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect confidentiality, although the specific vectors or details of the vulnerability are not disclosed.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 71

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0102**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x and 5.1.x. It allows remote authenticated users to affect availability, but the specific vectors or details of the vulnerability are not disclosed. It is noted that this vulnerability is different from CVE-2012-0087 and CVE-2012-0101.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the availability of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x and 5.1.x.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the availability of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x and 5.1.x allows remote authenticated users to affect availability via unknown vectors, a different vulnerability than CVE-2012-0087 and CVE-2012-0101.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x and 5.1.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect availability, although the specific vectors or details of the vulnerability are not disclosed. This vulnerability is different from CVE-2012-0087 and CVE-2012-0101.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 72

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0101**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x and 5.1.x. It allows remote authenticated users to affect availability, but the specific vectors or details of the vulnerability are not disclosed. It is noted that this vulnerability is different from CVE-2012-0087 and CVE-2012-0102.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the availability of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x and 5.1.x.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the availability of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x and 5.1.x allows remote authenticated users to affect availability via unknown vectors, a different vulnerability than CVE-2012-0087 and CVE-2012-0102.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x and 5.1.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect availability, although the specific vectors or details of the vulnerability are not disclosed. This vulnerability is different from CVE-2012-0087 and CVE-2012-0102.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 73

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0087**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x and 5.1.x. It allows remote authenticated users to affect availability, but the specific vectors or details of the vulnerability are not disclosed. It is noted that this vulnerability is different from CVE-2012-0101 and CVE-2012-0102.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the availability of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x and 5.1.x.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the availability of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x and 5.1.x allows remote authenticated users to affect availability via unknown vectors, a different vulnerability than CVE-2012-0101 and CVE-2012-0102.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x and 5.1.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect availability, although the specific vectors or details of the vulnerability are not disclosed. This vulnerability is different from CVE-2012-0101 and CVE-2012-0102.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 74

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3838**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6. It involves a denial of service issue where remote authenticated users can cause a server crash. This can be achieved by executing a query that uses the GREATEST or LEAST function with a mixed list of numeric and LONGBLOB arguments, which is not properly handled when the function's result is processed using an intermediate temporary table.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully executes a query with the GREATEST or LEAST function and a mixed list of numeric and LONGBLOB arguments, they can cause a denial of service by crashing the MySQL server. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential server crashes caused by queries with the GREATEST or LEAST function and mixed argument types.

Public exploits related to this finding

Description of the finding

MySQL 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 allows remote authenticated users to cause a denial of service (server crash) via a query that uses the (1) GREATEST or (2) LEAST function with a mixed list of numeric and LONGBLOB arguments, which is not properly handled when the function's result is "processed using an intermediate temporary table."

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 have a vulnerability that allows remote authenticated users to cause a denial of service (server crash). This vulnerability occurs when a query uses the GREATEST or LEAST function with a mixed list of numeric and LONGBLOB arguments. The server does not properly handle the function's result when it is processed using an intermediate temporary table, leading to a server crash.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.5.6, 5.1.51, or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 75

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3837**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6. It involves a denial of service issue where remote authenticated users can cause a server crash. This can be achieved by using a prepared statement that utilizes the GROUP_CONCAT function with the WITH ROLLUP modifier. This action can potentially trigger a use-after-free error when a copied object is modified in a way that also affects the original object.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully executes a prepared statement with the GROUP_CONCAT function and the WITH ROLLUP modifier, they can cause a denial of service by crashing the MySQL server. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential server crashes caused by prepared statements with the GROUP_CONCAT function and the WITH ROLLUP modifier.

Public exploits related to this finding

Description of the finding

MySQL 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 allows remote authenticated users to cause a denial of service (server crash) via a prepared statement that uses GROUP_CONCAT with the WITH ROLLUP modifier, probably triggering a use-after-free error when a copied object is modified in a way that also affects the original object.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 have a vulnerability that allows remote authenticated users to cause a denial of service (server crash). This vulnerability occurs when a prepared statement uses GROUP_CONCAT with the WITH ROLLUP modifier. It triggers a use-after-free error when a copied object is modified in a way that also affects the original object, leading to a server crash.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.5.6, 5.1.51, or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 76

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3836**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6. It involves a denial of service issue where remote authenticated users can cause a server crash and assertion failure. This can be achieved through vectors related to view preparation, pre-evaluation of LIKE predicates, and IN Optimizers.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully triggers the vulnerability through the mentioned vectors, they can cause a denial of service by crashing the MySQL server and triggering an assertion failure. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential server crashes and assertion failures caused by the mentioned vectors.

Public exploits related to this finding

Description of the finding

MySQL 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 allows remote authenticated users to cause a denial of service (assertion failure and server crash) via vectors related to view preparation, pre-evaluation of LIKE predicates, and IN Optimizers.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 have a vulnerability that allows remote authenticated users to cause a denial of service. This vulnerability occurs due to assertion failures and server crashes related to view preparation, pre-evaluation of LIKE predicates, and IN Optimizers.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.5.6, 5.1.51, or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 77

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3834**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6. It involves a denial of service issue where remote authenticated users can cause a server crash. This can be achieved through vectors related to "materializing a derived table that required a temporary table for grouping" and "user variable assignments." The specific details of the vulnerability are not disclosed.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully triggers the vulnerability through the mentioned vectors, they can cause a denial of service by crashing the MySQL server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from MySQL and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the availability of the MySQL server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in MySQL 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 allows remote authenticated users to cause a denial of service (server crash) via vectors related to "materializing a derived table that required a temporary table for grouping" and "user variable assignments."

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0 before 5.0.92, 5.1 before 5.1.51, and 5.5 before 5.5.6 have an unspecified vulnerability that allows remote authenticated users to cause a denial of service. This vulnerability is related to the materialization of a derived table that requires a temporary table for grouping and user variable assignments. Exploiting this vulnerability can lead to a server crash.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.5.6, 5.1.51, or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 78

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3682**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92. It involves a denial of service issue where remote authenticated users can cause a crash of the mysqld daemon. This can be achieved by using the EXPLAIN command with crafted "SELECT ... UNION ... ORDER BY (SELECT ... WHERE ...)" statements. This action triggers a NULL pointer dereference in the Item_singlerow_subselect::store function.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully executes the crafted EXPLAIN command, they can cause a denial of service by crashing the mysqld daemon. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Oracle MySQL that includes the fix for this vulnerability to prevent potential crashes of the mysqld daemon caused by the crafted EXPLAIN command.

Public exploits related to this finding

Exploit ID 34506.

- Description: MySQL 5.1.48 - 'EXPLAIN' Denial of Service

- Download URL: https://www.exploit-db.com/exploits/34506

Description of the finding

Oracle MySQL 5.1 before 5.1.49 and 5.0 before 5.0.92 allows remote authenticated users to cause a denial of service (mysqld daemon crash) by using EXPLAIN with crafted "SELECT ... UNION ... ORDER BY (SELECT ... WHERE ...)" statements, which triggers a NULL pointer dereference in the Item_singlerow_subselect::store function.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92 have a vulnerability that allows remote authenticated users to cause a denial of service. This vulnerability occurs when using EXPLAIN with crafted "SELECT ... UNION ... ORDER BY (SELECT ... WHERE ...)" statements. It triggers a NULL pointer dereference in the Item_singlerow_subselect::store function, leading to a crash of the mysqld daemon.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.49 or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

6. Be cautious with EXPLAIN statements: Exercise caution when using EXPLAIN statements, especially with complex queries. Avoid using crafted statements that could trigger the vulnerability.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 79

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-3677**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in Oracle MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92. It involves a denial of service issue where remote authenticated users can cause a crash of the mysqld daemon. This can be achieved by executing a join query that involves a table with a unique SET column.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully executes a join query involving a table with a unique SET column, they can cause a denial of service by crashing the mysqld daemon. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of Oracle MySQL that includes the fix for this vulnerability to prevent potential crashes of the mysqld daemon caused by join queries involving tables with unique SET columns.

Public exploits related to this finding

Description of the finding

Oracle MySQL 5.1 before 5.1.49 and 5.0 before 5.0.92 allows remote authenticated users to cause a denial of service (mysqld daemon crash) via a join query that uses a table with a unique SET column.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
Oracle MySQL versions 5.1 before 5.1.49 and 5.0 before 5.0.92 have a vulnerability that allows remote authenticated users to cause a denial of service. This vulnerability occurs when a join query is executed using a table that has a unique SET column. The vulnerability triggers a crash of the mysqld daemon.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.49 or 5.0.92, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 80

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2009-4019**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41. It involves two separate issues:

1. Improper handling of errors during the execution of certain SELECT statements with subqueries. 2. Failure to preserve certain null_value flags during the execution of statements that use the GeomFromWKB function.

These issues allow remote authenticated users to cause a denial of service by crashing the mysqld daemon through the use of crafted statements.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully executes a crafted statement that triggers the mentioned issues, they can cause a denial of service by crashing the mysqld daemon. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires authentication and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential crashes of the mysqld daemon caused by crafted statements.

Public exploits related to this finding

Exploit ID 33397.

- Description: MySQL 6.0.9 - SELECT Statement WHERE Clause Sub-query Denial of Service
- Download URL: https://www.exploit-db.com/exploits/33397

Exploit ID 33398.

- Description: MySQL 6.0.9 - 'GeomFromWKB()' Function First Argument Geometry Value Handling Denial of Service
- Download URL: https://www.exploit-db.com/exploits/33398

Description of the finding

mysqld in MySQL 5.0.x before 5.0.88 and 5.1.x before 5.1.41 does not (1) properly handle errors during execution of certain SELECT statements with subqueries, and does not (2) preserve certain null_value flags during execution of statements that use the GeomFromWKB function, which allows remote authenticated users to cause a denial of service (daemon crash) via a crafted statement.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0.x before 5.0.88 and 5.1.x before 5.1.41 have a vulnerability that allows remote authenticated users to cause a denial of service. This vulnerability has two aspects:
1. Improper handling of errors during the execution of certain SELECT statements with subqueries.
2. Failure to preserve certain null_value flags during the execution of statements that use the GeomFromWKB function.

Exploiting this vulnerability, remote authenticated users can craft statements that cause a daemon crash, leading to a denial of service.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.41 or 5.0.88, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

6. Be cautious with subqueries and GeomFromWKB function: Exercise caution when using subqueries and the GeomFromWKB function. Validate and sanitize user input to prevent the execution of crafted statements.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 81

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2008-3963**
- Severity score: **4.0** (**MEDIUM** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Medium" severity due to its moderate impact on the organization's security. A medium severity vulnerability indicates a moderate risk to the organization's systems and data.

The vulnerability occurs in MySQL versions 5.0 before 5.0.66, 5.1 before 5.1.26, and 6.0 before 6.0.6. It involves a handling issue with the b'' (b single-quote single-quote) token, which represents an empty bit-string literal. This vulnerability allows remote attackers to cause a denial of service by using this token in a SQL statement, resulting in a crash of the MySQL daemon.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote attacker successfully constructs a SQL statement that includes the b'' token and executes it against a vulnerable MySQL server, they can cause a denial of service by crashing the MySQL daemon. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability can be exploited remotely. This means that an attacker does not require direct access to the organization's network or systems, increasing the risk of opportunistic attacks and targeted attacks by malicious actors.

It is worth noting that the vulnerability specifically affects MySQL versions 5.0 before 5.0.66, 5.1 before 5.1.26, and 6.0 before 6.0.6.

In summary, the medium severity of this vulnerability indicates a moderate risk to the organization's systems and data. While the potential for denial of service exists, the vulnerability requires specific conditions and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential crashes of the MySQL daemon caused by the use of the b'' token in SQL statements.

Public exploits related to this finding

Exploit ID 32348.

- Description: MySQL 6.0.4 - Empty Binary String Literal Remote Denial of Service

- Download URL: https://www.exploit-db.com/exploits/32348

Description of the finding

MySQL 5.0 before 5.0.66, 5.1 before 5.1.26, and 6.0 before 6.0.6 does not properly handle a b'' (b single-quote single-quote) token, aka an empty bit-string literal, which allows remote attackers to cause a denial of service (daemon crash) by using this token in a SQL statement.

Mitigation steps

Vulnerability: Denial of Service in MySQL Server

Description:
MySQL versions 5.0 before 5.0.66, 5.1 before 5.1.26, and 6.0 before 6.0.6 have a vulnerability that allows remote attackers to cause a denial of service. This vulnerability occurs due to improper handling of an empty bit-string literal (b''). Attackers can exploit this vulnerability by using the empty bit-string literal in a SQL statement, causing a daemon crash and resulting in a denial of service.

Mitigation Steps:
To mitigate the risk of the denial of service vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 6.0.6, 5.1.26, or 5.0.66, depending on the MySQL version you are using.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

6. Validate and Sanitize User Input: Implement proper input validation and sanitization techniques to prevent the execution of crafted SQL statements.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 82

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2023-22053**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I'm sorry, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2023-22053". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Please ensure the correctness of the identifier or provide additional details if available.

Finding 83

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2010-1626**
- Severity score: **3.6** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in MySQL has been classified as "Low" severity due to its relatively low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability occurs in MySQL versions before 5.1.46 and involves a symlink attack in conjunction with the DROP TABLE command. It allows local users to delete the data and index files of another user's MyISAM table. This vulnerability is different from CVE-2008-4098 and CVE-2008-7247.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this vulnerability by using a symlink attack and executing the DROP TABLE command, they can delete the data and index files of another user's MyISAM table. However, the impact of this vulnerability is limited due to the low severity rating.

The severity is also influenced by the fact that this vulnerability requires local access. This means that an attacker needs to have direct access to the organization's systems, reducing the risk of opportunistic attacks. However, it still poses a threat if a local user with malicious intent is present.

It is worth noting that the vulnerability specifically affects MySQL versions before 5.1.46.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. While the potential for data and index file deletion exists, the vulnerability requires local access and does not directly lead to widespread compromise or disruption of critical systems. It is still important for the organization to address and remediate the issue by updating to a version of MySQL that includes the fix for this vulnerability to prevent potential unauthorized deletion of data and index files through symlink attacks in conjunction with the DROP TABLE command.

Public exploits related to this finding

Description of the finding

MySQL before 5.1.46 allows local users to delete the data and index files of another user's MyISAM table via a symlink attack in conjunction with the DROP TABLE command, a different vulnerability than CVE-2008-4098 and CVE-2008-7247.

Mitigation steps

Vulnerability: Symlink Attack in MySQL Server

Description:
MySQL versions before 5.1.46 have a vulnerability that allows local users to delete the data and index files of another user's MyISAM table. This vulnerability occurs due to a symlink attack in conjunction with the DROP TABLE command. By exploiting this vulnerability, a local user can create a symlink to the data and index files of another user's MyISAM table and then execute the DROP TABLE command, resulting in the deletion of the targeted table's files.

Mitigation Steps:
To mitigate the risk of the symlink attack vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than 5.1.46.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Limit Access to MySQL Files: Restrict access to the MySQL data directory and files to authorized users only. Ensure that file permissions are properly set to prevent unauthorized access.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

6. Monitor File System Activity: Implement monitoring and auditing mechanisms to detect any suspicious file system activity, such as unauthorized changes or creation of symlinks.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 84

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2023-22007**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I'm sorry, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2023-22007". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Please ensure the correctness of the identifier or provide additional details if available.

Finding 85

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0114**
- Severity score: **3.0** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Low" severity due to its relatively low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x, 5.1.x, and 5.5.x. It allows local users to affect confidentiality and integrity, but the specific vectors or details of the vulnerability are not disclosed.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a local user successfully exploits this unspecified vulnerability, they can affect the confidentiality and integrity of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires local access. This means that an attacker needs to have direct access to the organization's systems, reducing the risk of opportunistic attacks. However, it still poses a threat if a local user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the confidentiality and integrity of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x, 5.1.x, and 5.5.x allows local users to affect confidentiality and integrity via unknown vectors.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x has an unspecified vulnerability. This vulnerability allows local users to affect confidentiality and integrity, although the specific vectors or details of the vulnerability are not disclosed.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.

Finding 86

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **PRION:CVE-2022-21444**
- Severity score: **None** (**None** severity)

Finding severity rationale

The vulnerability does not have a severity attached to.

Public exploits related to this finding

Description of the finding

None

Mitigation steps

I'm sorry, but I couldn't find any information about the vulnerability with the identifier "PRION:CVE-2022-21444". It's possible that the identifier you provided is not recognized or does not correspond to a known vulnerability. Please ensure the correctness of the identifier or provide additional details if available.

Finding 87

- Affected IP: **192.168.203.129**
- Affected port and service: **3306** (service **mysql**)
- Related CVE: **CVE-2012-0075**
- Severity score: **1.7** (**LOW** severity)

Finding severity rationale

Severity Rationale:

The identified vulnerability in the MySQL Server component in Oracle MySQL has been classified as "Low" severity due to its relatively low impact on the organization's security. A low severity vulnerability indicates a low risk to the organization's systems and data.

The vulnerability is described as an unspecified vulnerability in MySQL Server versions 5.0.x, 5.1.x, and 5.5.x. It allows remote authenticated users to affect integrity, but the specific vectors or details of the vulnerability are not disclosed.

The severity of this vulnerability is determined by the potential consequences of successful exploitation. In this case, if a remote authenticated user successfully exploits this unspecified vulnerability, they can affect the integrity of the MySQL Server. However, the impact of this vulnerability is limited due to the lack of specific information.

The severity is also influenced by the fact that this vulnerability requires authentication. This means that an attacker needs to have valid credentials to exploit the vulnerability, reducing the risk of opportunistic attacks. However, it still poses a threat if an authenticated user with malicious intent is present.

It is worth noting that the vulnerability specifically affects Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x.

In summary, the low severity of this vulnerability indicates a low risk to the organization's systems and data. The lack of specific details about the vulnerability limits the understanding of its potential impact. It is important for the organization to closely monitor updates and advisories from Oracle and apply any patches or updates provided to mitigate the risk associated with this unspecified vulnerability and maintain the integrity of the MySQL Server.

Public exploits related to this finding

Description of the finding

Unspecified vulnerability in the MySQL Server component in Oracle MySQL 5.0.x, 5.1.x, and 5.5.x allows remote authenticated users to affect integrity via unknown vectors.

Mitigation steps

Vulnerability: Unspecified Vulnerability in MySQL Server

Description:
The MySQL Server component in Oracle MySQL versions 5.0.x, 5.1.x, and 5.5.x has an unspecified vulnerability. This vulnerability allows remote authenticated users to affect integrity, although the specific vectors or details of the vulnerability are not disclosed.

Mitigation Steps:
To mitigate the risk of the unspecified vulnerability in MySQL Server, the following steps can be taken:

1. Update MySQL: Upgrade to a version of MySQL that includes a fix for this vulnerability. Ensure that you are using a version later than the affected versions mentioned in the vulnerability description.

2. Apply Patches: If an official patch is available for your version of MySQL, apply it to fix the vulnerability. Patches are typically provided by the MySQL development team or the vendor of your operating system.

3. Regularly Update and Patch Software: Keep MySQL and all other software up to date with the latest security patches. Regular updates help protect against known vulnerabilities and ensure that the server is running the latest stable version.

4. Implement Access Controls: Review and implement appropriate access controls to restrict access to sensitive databases. Limit the privileges of database users to minimize the potential impact of an attack.

5. Monitor and Log Database Activity: Implement a centralized logging system to monitor and log MySQL activity. Regularly review the logs for any suspicious or unauthorized access attempts. This can help detect and respond to potential security incidents.

Note: It is important to consult the official documentation and follow best practices provided by MySQL for applying security updates and securing MySQL databases.