|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Windows BOF (Core)  Otis Smith / Cybersecurity Professional / 11.16.23 |  |
|  |  |  |  |
| Pipette dropping liquid in a petri dish | | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
|  |  | TABLE OF CONTENTS |  |
|  |  | Summary …………………………..  Discovery……………………………  Vulnerability……………………….  Exploitation………………………..  References.………………………..  Mitigation………………………….. |  |



|  |  |  |
| --- | --- | --- |
| Summary |  | |
| This report details the process of performing a Windows Privilege Escalation (Core) lab exercise, focusing on escalating privileges on a Windows Server machine in Metasploitable 3 using the Juicy Potato technique. The steps involve setting up the environment, discovering vulnerabilities, exploiting them, and finally gaining elevated privileges.  A hand holding a glowing city  Description automatically generated | |  |
| Discovery   * Setting Up the Environment: * Downloaded and imported the virtual machine using Oracle VM VirtualBox.   Use this “**https://drive.google.com/u/0/uc?id=1Jf5vd83XKwsfDjS7lVT-HrMxc12zIPfA&export=download**”  Click on “**Download**” to download the file.  A screenshot of a computer  Description automatically generated  The file has been downloaded.  A screenshot of a computer  Description automatically generated  Open “**Oracle VM VirtualBox Manager**” and click on “**File**” and “**Import Appliance….**” and follow all the defaults.  A screenshot of a computer  Description automatically generated  Review all the data and click “**Finish**”  A screenshot of a computer  Description automatically generated  The new “**Coding Dojo BOF**” has now been created.  A screenshot of a computer  Description automatically generated  Select “**Coding Dojo BOF**” and click on “**Settings**”  A black and blue rectangle  Description automatically generated  Selected “**Network**”, selected “**NAT Network**” and then “**Ok**”  A computer screen shot of a computer  Description automatically generated   * Ensured both the Kali Linux and Windows VM were on the same NAT Network. * Initial System Configuration: * Launched SLmail and ran "slmgr /rearm" on the Windows VM.   After logging into theWindows VM, **right-click** on the “**start button**” in the lower left-hand corner of the screen. Type in the command “**cmd**” in the search box. **Right-click** on “**cmd.exe**” and select “**Run as administrator**”  A screenshot of a computer  Description automatically generated  The cmd box **opened**. Ran the cmd "**slmgr /rearm**"  A screenshot of a computer  Description automatically generated   * Installed and configured the SLmail application. * Double click “**12f1ab027e5374587e7e998c00682c5d-SLMail55\_4433.exe”** file on the desktop and just click through everything as is to finish the setup.   A computer screen shot of a blue screen  Description automatically generated  Click “**Yes**”  A screenshot of a computer error  Description automatically generated  Click “**Finish**”  A computer screen with a message  Description automatically generated  Open “**File Explorer**”, select “**Computer**” and then put in “**SL Mail.cp**” in the search box. Once file is found, “**right click**”, “**Send to**”, and “**Desktop (create shortcut)**  A computer screen shot of a computer  Description automatically generated   1. Identifying the Target:  * Used nmap to identify the Windows VM's IP address (10.0.2.12) and the target port (POP3 110).   Did the nmap command **nmap 10.0.2.1-254** and found the **Win BOF ip** of **10.0.2.12** and the target port “ **POP3 110**”A screenshot of a computer  Description automatically generated   1. Fuzzing:  * Used a Python script to perform fuzzing on the target, starting with a buffer of "A" and increasing the payload size gradually.   Made the changes and saved the file as “**classFuzzy.py**” . A bunch of letter “**A**” will be sent to crash the buffer when this file is running.  A screenshot of a computer  Description automatically generated   * Observed a crash at **2900 bytes**, indicating a potential buffer overflow.   The file has crashed at “**Fuzzing PASS with 2900 bytes**”. It shows that the “**EIP**” has been replaced with “**41414141**” which is equal to the hex number “**A**”. A whole bunch of “**A’s**” were sent to crash the buffer.  A screenshot of a computer  Description automatically generated   1. Determining Offset:  * Conducted further analysis to determine the offset by creating a pattern with **3000 bytes** and identifying the crash point.   Run the command “**msf-pattern\_create -l 3000**” to create another file/tool with 3000 bytes which will exceed the bytes of “**2900**”  A screenshot of a computer screen  Description automatically generated   * Found the offset to be 2606.   Did the command “**msf-pattern\_offset – l 3000 -q 39694438**” to query this number “**39694438**” that over ran the “**EIP**” and receive message “**Exact match at offset 2606**”  A screen shot of a computer program  Description automatically generated   1. Exploitation:  * Developed a new Python script with the payload containing a JMP ESP instruction.   A screenshot of a computer program  Description automatically generated   * Obtained the **JMP ESP address** from the **mona modules** and mona find commands in Immunity Debugger.   Search to find the “**JMP ESP**” by using this “ **!mona find -s \xff\xe4 -m SLMFC.DLL** “Scroll down and find this one to use “**SLMFC.DLL**” on the top of the list with **19 total found**.  A screen shot of a computer  Description automatically generated  Used **MSFvenom** to generate a **reverse shell payload**, avoiding bad characters.    Modified the Python script to include the crafted payload. | |  |
|  | |  |

|  |  |  |
| --- | --- | --- |
| Vulnerability |  | |
| The vulnerability exploited is a buffer overflow in the SLmail application. The flaw occurs due to insufficient bounds checking, allowing an attacker to overwrite the EIP (Extended Instruction Pointer) and gain control over the program's execution flow.  A screenshot of a computer  Description automatically generated | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exploitation | | |  | |
| 1. Identifying Bad Characters:  * Utilized a script to identify bad characters by comparing the expected values with the ones present in the payload.   Ran the command “**python2 classBaddies.py**” and it returned kind of the same result.  A screenshot of a computer  Description automatically generated  Select the “**ESP 024BA128**” then right click and then select also follow in dump.  A computer screen shot of a computer screen  Description automatically generated  A screenshot of a computer  Description automatically generated  Move down to the “**Hex dump**” section to view the file to compare with the baddies to find the bad characters.  A screenshot of a computer  Description automatically generated   * Found '\x00', '\x0a', and '\x0d' to be bad characters.   These **“\x00\x0a\x0d**” were the only bad characters found.  A screenshot of a computer  Description automatically generated   1. Finding the JMP ESP Instruction:  * Used Immunity Debugger and mona modules to identify the SLMFC.DLL module.   Looked for the ones with the field “**False, False, False, False, True**” and decided to use this one with name “**SLMFC.DLL**”  A computer screen shot of a computer screen  Description automatically generated   * Employed mona find to locate the JMP ESP instruction within the module.   A computer screen with a black background  Description automatically generated   1. Crafting the Payload:  * Developed a new Python script incorporating the jump to ESP instruction and avoiding bad characters.   A screenshot of a computer  Description automatically generated   * Generated a reverse shell payload using MSFvenom.   Ran the command “**python2 classShell.py**” and get “**Fuzzing PASS with 2977 bytes**”  A screen shot of a computer  Description automatically generated   1. Exploiting the Vulnerability:  * Executed the Python script, leading to a successful exploitation. * Established a reverse shell connection to the attacker's machine.   A screenshot of a computer program  Description automatically generated | | | |  |
| References |  |  | |  |

1. Tools Used:

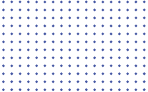
* Oracle VM VirtualBox
* Kali Linux
* Immunity Debugger
* MSFvenom
* nmap
* Netcat (nc)3

1. Documentation:

* Mona Script Documentation
* MSFvenom Documentation
* Immunity Debugger Manual

1. Online Resources:

Various online forums, tutorials, and documentation related to buffer overflow exploitation in Windows environments.

Mitigation: 

1. Update and Patch Applications:
   * Regularly update and patch applications, including SLmail, to address known vulnerabilities and improve security.
2. Input Validation:
   * Implement proper input validation in applications to ensure that user inputs are within expected ranges, preventing buffer overflow vulnerabilities.
3. Code Review and Testing:
   * Conduct thorough code reviews and testing, including penetration testing, to identify and address potential security flaws in the application code.
4. Use of Modern Development Practices:
   * Adopt secure coding practices and use modern development frameworks that include built-in security features to mitigate common vulnerabilities.
5. Network Segmentation:
   * Implement network segmentation to isolate critical systems from potential attackers, reducing the impact of a successful exploitation.
6. Intrusion Detection and Monitoring:
   * Deploy intrusion detection systems (IDS) and continuously monitor network traffic for unusual or suspicious activities that may indicate a buffer overflow attempt.
7. Regular Security Audits:
   * Conduct regular security audits, including vulnerability assessments and penetration testing, to proactively identify and address security weaknesses.
8. Least Privilege Principle:
   * Follow the principle of least privilege, ensuring that applications and users have the minimum level of access necessary to perform their tasks.
9. Memory Protection Mechanisms:
   * Enable and utilize memory protection mechanisms, such as Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR), to make it more difficult for attackers to exploit vulnerabilities.
10. Use of Safe Functions:
    * Use safe and bounded functions in the application code to handle string and memory operations, reducing the risk of buffer overflow vulnerabilities.
11. Security Training for Developers:
    * Provide security training for developers to raise awareness of secure coding practices and common vulnerabilities, including buffer overflow risks.
12. Incident Response Plan:
    * Develop and regularly update an incident response plan to ensure a swift and coordinated response in case of a security incident.
13. User Education:
    * Educate users and administrators about the risks associated with opening unexpected or suspicious emails, as buffer overflow vulnerabilities may be exploited through phishing attacks.
14. Network Traffic Encryption:
    * Encrypt network traffic, especially communication involving sensitive information or critical systems, to protect against man-in-the-middle attacks.
15. Implementing Firewall Rules:
    * Configure firewalls to restrict unnecessary incoming and outgoing traffic, limiting the attack surface and preventing unauthorized access.
16. Regular Backup and Recovery:
    * Implement regular backup procedures for critical data and systems to facilitate quick recovery in case of a security incident.
17. Collaboration with the Community:
    * Engage with the cybersecurity community, participate in forums, and share information to stay updated on emerging threats and mitigation strategies.

By implementing these mitigation measures, organizations can enhance the security of their systems, reduce the risk of buffer overflow vulnerabilities, and better protect against potential exploitation attempts.

Report:

This report provides a comprehensive overview of the entire process, from setting up the environment to successfully exploiting the buffer overflow vulnerability in the SLmail application.