Abstract – This document presents a analysis of the vulnerabilities identified in Ivanti Secure Access VPN (Pulse Secure VPN) with their potential impact on organizations that rely on this VPN. The analysis delves into various aspects of these vulnerabilities, including their exploitation methods, potential impacts, and the challenges encountered during the exploitation process.

The document provides a qualitative summary of the analyzed vulnerabilities, offering valuable insights for cybersecurity professionals, IT administrators, and other stakeholders in various industries. By understanding the technical nuances, exploitation methods, and mitigation strategies, readers can enhance their organizational security posture against similar threats.

This analysis is particularly beneficial for security professionals seeking to understand the intricacies of VPN vulnerabilities and their implications for enterprise security. It also serves as a resource for IT administrators responsible for maintaining secure VPN configurations and for industry stakeholders interested in the broader implications of such vulnerabilities on digital security and compliance.

I. Introduction

Northwave Cybersecurity has identified several vulnerabilities in Ivanti Secure Access VPN (Pulse Secure VPN). These vulnerabilities, specifically CVE-2023-38043, CVE-2023-35080, and CVE-2023-38543, have been found to affect the VPN software used by over 40,000 organizations globally. The main vulnerability discussed allows for privilege escalation due to a kernel driver installed by the VPN software that creates a device readable and writable by any user. This can potentially lead to kernel corruption or privilege escalation.

II. Vulnerabilities

CVE-2023-38043, CVE-2023-35080, CVE- 2023-38543 are identified in all versions of the Ivanti Secure Access Client below 22.6R1.1.

This security flaw of CVE-2023-38043 could allow a locally authenticated attacker to exploit a vulnerable configuration, potentially leading to a Denial of Service (DoS) condition on the user's machine. In some scenarios, this vulnerability could result in a full compromise of the system.

CVE-2023-35080 is a vulnerability identified in the Ivanti Secure Access Windows client, which could allow a locally authenticated attacker to exploit a vulnerable configuration. This could potentially lead to various security risks, including the escalation of privileges, denial of service (DoS), or information disclosure.

CVE-2023-38543 is a vulnerability that exists in all versions of the Ivanti Secure Access Client (ISAC) below 22.6R1.1. This security flaw could allow a locally authenticated attacker to exploit a vulnerable configuration, potentially leading to a denial of service (DoS) condition on the user's machine. In some scenarios, this vulnerability could result in a full compromise of the system.

The vulnerability arises when a specific component is loaded, and a local attacker sends a specially crafted request to this component. Successful exploitation of this vulnerability could enable the attacker to gain elevated privileges on the affected system. The severity of this vulnerability is rated as high, with a CVSS 3.x base score of 7.8 by NIST and an 8.8 score by HackerOne, indicating a significant impact on confidentiality, integrity, and availability.

Mitigation strategies for CVEs include updating the Ivanti Secure Access Client to version 22.6R1.1 or later, as this version addresses the vulnerability. Users are advised to apply the update as soon as possible to protect their systems from potential exploitation.

A. *Attack flow*

- **Initial Access**: The attacker must first obtain the ability to execute low-privileged code on the target system. This could be achieved through various means, such as phishing, exploiting another vulnerability, or having legitimate access to a user account on the system.
- **Exploitation**: Once the attacker has the ability to execute code on the target system, they would exploit the vulnerable configuration in the Ivanti Secure Access Client. The specific details of the vulnerable configuration and how it is exploited are not provided in the search results, but it would involve sending a specially crafted request to a component of the Ivanti Secure Access Client.
- **Denial of Service**: The successful exploitation of the vulnerability could lead to a DoS condition, where the affected machine becomes unresponsive or crashes.
- **System Compromise**: In some scenarios, the vulnerability could be leveraged to gain elevated privileges or execute arbitrary code, leading to a full compromise of the system.

B. *Affected industries*

CVEs affect various industries that utilize the Ivanti Secure Access Client (ISAC), previously known as Pulse Secure Desktop Client, for secure remote access to their networks.

- **Healthcare**: Hospitals and healthcare providers use VPN clients for secure remote access to patient records and internal systems, making them potential targets.
- **Financial Services**: Banks, insurance companies, and other financial institutions rely on secure VPN access for remote employees and to protect sensitive financial data.
- **Government and Public Sector**: Government agencies use VPN clients to ensure secure communication and access to confidential government resources remotely.
- **Education**: Universities and educational institutions utilize VPN clients for secure access to academic resources and to enable remote learning and administration.
- **Technology and IT Services**: Companies in the technology sector, including IT service providers, use VPN clients for secure remote access to network resources and client environments.
- **Manufacturing and Critical Infrastructure**: Manufacturing firms and critical infrastructure providers use VPN clients to securely connect to industrial control systems and operational technology networks.
- **Retail and Consumer Goods**: Retailers use VPN clients for secure remote access to inventory management, point of sale systems, and other critical business applications.

1) *Healthcare*

In the healthcare industry, the consequences of such a vulnerability being exploited could include:

- **Disruption of Healthcare Services**: A denial-of-service attack could disrupt access to critical healthcare systems and patient data, impacting patient care and potentially leading to delays in treatment or diagnosis.
- **Compromise of Sensitive Data**: Elevated privileges could allow attackers to access, modify, or delete sensitive patient data, violating patient privacy and potentially leading to identity theft or fraud.
- **Regulatory and Compliance Violations**: Healthcare organizations are subject to strict regulatory requirements for protecting patient data. A security breach resulting from this vulnerability could lead to regulatory fines and legal consequences.
- **Damage to Reputation**: A security incident could damage the reputation of the affected healthcare organization, leading to a loss of trust among patients and partners.
- **Financial Costs**: Responding to and recovering from a security breach can be costly, including the expenses related to investigation, remediation, legal fees, and potential settlements or fines.

2) *Financial Services industry*

In the Financial Services industry, the exploitation of CVEs could have the following consequences:

- **Disruption of Financial Operations**: A denial-of-service attack could disrupt access to critical financial systems, affecting transactions, trading, and other time-sensitive operations, potentially leading to financial losses.
- **Theft of Sensitive Financial Data**: Elevated privileges could enable attackers to access, modify, or exfiltrate sensitive financial data, including client accounts, transaction histories, and proprietary trading algorithms, leading to financial fraud and competitive disadvantage.
- **Regulatory and Compliance Breaches**: Financial institutions are subject to stringent regulatory requirements for data protection and cybersecurity. A security breach resulting from this vulnerability could result in regulatory fines, sanctions, and increased scrutiny.
- **Reputational Damage**: Security incidents can severely damage the reputation of financial institutions, eroding client trust and potentially leading to a loss of business as clients move their assets to perceived safer institutions.
- **Financial Costs**: The costs associated with responding to and recovering from a security breach can be substantial, including forensic investigations, system remediations, legal fees, and potential compensation for affected clients.

3) *Government and Public Sector*

Impact on Government and Public Sector are:

- **Disruption of Essential Services**: Government agencies provide essential services to the public, including emergency services, social services, and infrastructure management. A DoS attack exploiting this vulnerability could disrupt these critical services, affecting public safety and welfare.
- **Exposure of Sensitive Information**: Government agencies handle highly sensitive information, including personal data of citizens, classified national security information, and critical infrastructure data. A full system compromise could lead to the exposure of such information, with severe implications for national security and individual privacy.
- **Loss of Public Trust**: Any breach or disruption in government services due to a cybersecurity incident can lead to a significant loss of public trust in government institutions. Restoring this trust can be a long and challenging process.
- **Regulatory and Legal Consequences**: Government agencies are subject to strict regulatory and legal frameworks regarding data protection and cybersecurity. A breach resulting from this vulnerability could lead to legal challenges, inquiries, and the imposition of penalties.
- **Financial Implications**: Responding to and recovering from a cybersecurity incident can be costly. This includes the costs associated with forensic

investigations, system remediations, potential legal liabilities, and measures to prevent future incidents.

4) *Education industry*

Here are some potential impacts and consequences of CVEs in the Education industry:

- **Disruption of Educational Services**: A denial-of-service attack could disrupt access to learning management systems, virtual classrooms, and other online educational resources, affecting both teaching and learning activities.
- **Exposure of Sensitive Data**: If the vulnerability leads to a system compromise, sensitive data such as student records, research data, and personal information of faculty and students could be accessed or leaked.
- **Regulatory and Compliance Issues**: Educational institutions are often subject to regulations regarding the protection of student data. A security breach could result in non-compliance with these regulations, leading to legal and financial repercussions.
- **Reputational Damage**: A security incident could damage the institution's reputation, potentially affecting student enrollment and partnerships with other organizations.
- **Financial Costs**: The costs associated with responding to a security breach, including investigations, system remediation, and potential legal liabilities, can be significant for educational institutions.

5) *Technology and IT Services industry*

Potential Impacts and Consequences are:

- **Disruption of IT and Technology Services**: A Denial of Service (DoS) attack exploiting this vulnerability could disrupt access to critical IT infrastructure and services, affecting both the service providers and their clients. This could lead to downtime, loss of productivity, and breach of service level agreements (SLAs).
- **Compromise of Sensitive Data**: The vulnerability could potentially lead to a full system compromise, allowing unauthorized access to sensitive data such as intellectual property, source code, customer data, and internal communications. This could have severe implications for confidentiality and data integrity.
- **Regulatory and Compliance Risks**: Many technology and IT services firms are subject to regulatory requirements concerning data protection and cybersecurity. A security breach resulting from CVE-2023-38043 could lead to non-compliance, resulting in fines, legal actions, and increased regulatory scrutiny.
- **Reputational Damage**: The reputation of technology and IT services companies is heavily dependent on their ability to protect their own and their clients' data. A security incident could erode trust, potentially leading to loss of clients and difficulty in acquiring new business.
- **Financial Costs**: The financial implications of responding to and recovering from a security breach can be substantial. Costs may include forensic investigations, system remediations, legal fees, and compensations for affected parties.

6) *Manufacturing and Critical Infrastructure industry*

In the Manufacturing and Critical Infrastructure industry, the exploitation of CVEs could have the following consequences:

- **Disruption of Operations**: A DoS attack could disrupt access to critical systems and networks, affecting production lines, supply chain management, and operational technology (OT) environments.
- **Compromise of Sensitive Data**: Elevated privileges could enable attackers to access, modify, or exfiltrate sensitive data, including proprietary manufacturing processes, infrastructure control systems data, and employee information.
- **Safety Risks**: In critical infrastructure sectors, such as energy, water, and transportation, a system compromise could pose direct safety risks to the public and the environment.
- **Regulatory and Compliance Violations**: Many manufacturing and critical infrastructure organizations are subject to regulatory requirements for cybersecurity. A security breach could lead to non-compliance, resulting in fines and legal actions.
- **Reputational Damage**: A security incident in these industries can lead to a loss of confidence from customers, partners, and regulators, potentially affecting future business opportunities.
- **Financial Costs**: The financial impact of a security breach can be considerable, including the costs of incident response, system restoration, and potential legal liabilities.

7) *Retail and Consumer Goods industry*

Here are some potential impacts and consequences of CVEs in the Retail and Consumer Goods industry:

- **Disruption of Retail Operations**: A denial-of-service attack could disrupt access to critical retail systems, affecting sales, inventory management, and customer service. This could lead to lost revenue and dissatisfied customers.
- **Compromise of Sensitive Data**: If the vulnerability leads to a system compromise, sensitive data such as customer payment information, proprietary business data, and employee information could be accessed or leaked.
- **Regulatory and Compliance Issues**: Retailers are often subject to regulations regarding the protection of consumer data. A security breach could result in non-compliance with these regulations, leading to legal and financial repercussions.
- **Reputational Damage**: A security incident could damage the retailer's reputation, potentially affecting customer loyalty and brand value.
- **Financial Costs**: The costs associated with responding to a security breach, including investigations, system

remediation, and potential legal liabilities, can be significant for retail organizations.

III. Extra Details

The IOCTL number 0x80002018 is associated with a vulnerable function within the IRP_MJ_DEVICE_CONTROL callback of a kernel driver. This function is designed to handle specific I/O control codes (IOCTLs) that are sent from user-mode applications to the driver. The code handling this IOCTL contains a privilege escalation vulnerability due to the following sequence of operations:

- A pointer to input data passed from user-mode (systembuffer) is loaded.
- The first value inside that input is taken as a pointer to a driver-specific structure.
- A pointer at offset +28h inside that structure is loaded.
- A pointer to offset +50h inside the memory that the previous pointer is pointing to is passed to the kernel API IoCsqRemoveIrp.
- Additionally, the second argument provided to the IoCsqRemoveIrp call, which is located in the RDX register, is also under the control of the user.

The IoCsqRemoveIrp function is a kernel API that removes an IRP (I/O Request Packet) from a queue using function pointers (callbacks) contained within the first argument passed to the API. The vulnerability arises because the user has control over this first argument, which means they can manipulate the function pointers used by IoCsqRemoveIrp to execute arbitrary code with kernel privileges.

The IoCsqRemoveIrp function itself is relatively straightforward and uses the queue's dispatch routines to remove the specified IRP from the queue. However, the critical security issue here is that the user can control both the RCX and RDX registers, which are used as arguments to the function. Inside the function, there are multiple places where a pointer gets loaded from the first argument (RCX) and is then passed to _guard_dispatch_icall. This internal function is designed to call whatever function pointer is in the RAX register, but it has a significant limitation: the pointer in RAX must be at the start of a valid function that is part of the kernel image. This means that shellcode or non-kernel-image functions cannot be called directly.

In summary, the vulnerability in the IOCTL handling code allows an attacker to control the function pointers used by IoCsqRemoveIrp, potentially leading to arbitrary code execution with kernel privileges. This is a serious security flaw that can be exploited for privilege escalation, allowing an attacker with local access to the system to gain full control over it.

The constraints outlined in the scenario with the vulnerable IOCTL handling in a kernel driver illustrate the complexity and challenges in developing a reliable exploit for a kernel vulnerability. Let's break down these constraints and their implications for exploit development:

A. *Constraint 1: Guaranteed Bluescreen*

The automatic deallocation of the user-provided pointer via ExFreePoolWithTag at the end of the IOCTL handling routine presents a significant challenge. This operation requires a valid kernel pointer, which is difficult for a regular user to provide. Even if an attacker manages to supply a valid pointer, its deallocation could lead to kernel instability or corruption, likely resulting in a system crash (bluescreen). This constraint significantly complicates the development of a stable exploit, as it requires the exploit to either avoid triggering this deallocation or to ensure that the deallocation does not lead to adverse effects on system stability.

B. *Constraint 2: Heavily Limited Argument Control*

The limited control over the arguments passed to the functions called by IoCsqRemoveIrp through _guard_dispatch_icall poses another challenge. The exploit has control over the RCX register (pointing to a memory area with function pointers) and, in one instance, the RDX register (pointing to a controlled memory area). However, for the other calls, RDX points to a stack area outside the attacker's control, and the R8 register, which could potentially carry additional data, is not utilized within the context of these function calls. This limitation severely restricts the exploit's ability to manipulate the execution flow of the called functions, making it difficult to achieve arbitrary code execution without causing a system crash.

C. *Constraint 3: Guarded Calls*

The use of _guard_dispatch_icall as a defensive measure by Microsoft further complicates exploit development. This mechanism ensures that only pointers to legitimate functions within the ntoskrnl.exe image can be called, effectively preventing the execution of arbitrary shellcode or functions outside the kernel image. Finding a sequence of three functions within the kernel that can be called with the limited argument control available, without causing a crash, is a significant challenge. This constraint requires an in-depth understanding of the kernel's internals and available functions to identify a viable chain that could lead to successful exploitation.

D. *Bluescreen bypass*

To address the challenge of bypassing the guaranteed bluescreen after exploiting the vulnerability, the approach involves leveraging the last function call before the system crashes. The idea is to prevent execution from continuing after this last function call, without causing a system crash. The proposed solution involves using synchronization and locking functions, specifically targeting a kernel sync function that can lock the entire thread indefinitely, thus preventing it from reaching the ExFreePoolWithTag call that leads to a bluescreen.

The chosen function is KxWaitForSpinLockAndAcquire for this purpose. This function takes a pointer in the RCX register and checks if the value at the start of the memory it points to is non-zero. If it is, the function enters a loop, checking the value repeatedly until it becomes zero. However, by setting the first 8 bytes of the memory pointed to by RCX to a non-zero value, the thread can be locked in an infinite loop, effectively preventing the bluescreen without crashing the system.

However, locking a kernel thread in an infinite loop can significantly impact system performance, causing the computer to slow down after executing the exploit multiple times. To mitigate this, the exploit can adjust the thread's priority to the lowest possible setting using the SetThreadPriority() API with the THREAD_PRIORITY_LOWEST parameter. This ensures that the locked thread receives the least amount of CPU time, minimizing its impact on system performance.

In summary, the strategy to bypass the bluescreen involves:

- Using the KxWaitForSpinLockAndAcquire function to lock the thread in an infinite loop, preventing it from reaching the ExFreePoolWithTag call.
- Setting the locked thread's priority to the lowest possible to minimize its impact on system performance.

E. *Reaching the vulnerable code*

To reach the vulnerable code and properly set up the IOCTL's input buffer to target the IoCsqRemoveIrp call, the following steps are taken in the provided code snippet:

- A HANDLE to the device is obtained by calling CreateFile with the DEVICE_NAME.
- An input buffer is allocated and initialized to zero using calloc.
- The first 8 bytes of the input buffer are set to point to an initial_buffer.
- The initial_buffer is then set up with pointers at offsets 0x28 and 0x30 to point to buff_28h and buff_30h, respectively.
- The DeviceIoControl function is called with the VULN_IOCTL code and the prepared input buffer.

The code snippet is designed to satisfy the checks performed by the driver on the input buffer before calling IoCsqRemoveIrp. Specifically, it ensures that:

- The first value in the input buffer is a non-NULL pointer to another buffer (initial_buffer).
- The initial_buffer contains non-NULL pointers at offsets +0x28 and +0x30.
- These pointers are used to pass a pointer to offset +0x50 in the buffer that buff_28h points to as the first argument to IoCsqRemoveIrp.
- The pointer loaded from offset +0x28 (buff_28h) is passed as the second argument to the function.

By setting up the input buffer in this way and calling DeviceIoControl, the code reaches the vulnerable area of the driver code where IoCsqRemoveIrp is called, as confirmed by hitting the breakpoint in a debugger.

The IoCsqRemoveIrp function is a kernel API that removes an IRP (I/O Request Packet) from a queue using function pointers (callbacks) contained within the first argument passed to the API. The vulnerability in the IOCTL handling code allows an attacker to control the function pointers used by IoCsqRemoveIrp, potentially leading to arbitrary code execution with kernel privileges.

F. *Controlling IoCsqRemoveIrp*

To control the IoCsqRemoveIrp function and prepare the input to satisfy all checks inside of it, the following steps are taken:

- The input buffer is set up to reach the IoCsqRemoveIrp call, ensuring that the first 8 bytes of the input buffer are interpreted as a pointer to another buffer, and that this pointer is not NULL.
- The buffer pointed to by the first 8 bytes of the input buffer is then set up with pointers at offsets +0x28 and +0x30 to point to buff_28h and buff_30h, respectively.
- The buff_28h buffer is prepared with function pointers for the three function calls that IoCsqRemoveIrp will make. These pointers are placed at the appropriate offsets within buff_28h:
  - The first function call pointer is placed at offset +0x20.
  - The second function call pointer is placed at offset +0x10.
  - The third function call pointer is placed at offset +0x28.
- A separate buffer, iocsq_rsi_plus_8h, is allocated and a non-zero value is placed at offset +0x68 to satisfy a check within IoCsqRemoveIrp.
- The buff_30h buffer is set up to point to iocsq_rsi_plus_8h at offset +0x08, and a non-zero value is also placed at offset +0x68 within buff_30h.
- To prevent a bluescreen after exploiting the vulnerability, the third function call is set to KxWaitForSpinLockAndAcquire, which will lock the thread indefinitely and prevent it from reaching the ExFreePoolWithTag call that would cause a bluescreen.
- The first two function calls are set to HalMakeBeep, a harmless kernel function that does not crash and takes no arguments.
- The buff_28h buffer at offset +0x50 is set to a non-zero value to provide a locked spinlock object to KxWaitForSpinLockAndAcquire.

By setting up the input buffer in this way and calling DeviceIoControl with the VULN_IOCTL code, the exploit is able to reach the vulnerable area of the driver code where IoCsqRemoveIrp is called and control the function pointers used by IoCsqRemoveIrp, potentially leading to arbitrary code execution with kernel privileges

G. *Write What Where*

The vulnerabilities discovered in Ivanti Secure Access VPN, previously known as Pulse Secure VPN, by Northwave Cybersecurity have significant implications for cybersecurity. These vulnerabilities, specifically CVE-2023-38043, CVE-2023-35080, and CVE-2023-38543, affect the VPN software utilized by over 40,000 organizations globally. The primary vulnerability allows for privilege escalation due to a kernel driver installed by the VPN software, which creates a device readable and writable by any user. This flaw can potentially lead to kernel corruption or privilege escalation.

The exploitation process detailed by Northwave involves stopping the VPN client to avoid memory corruptions, using the command "%programfiles(x86)%\Common Files\Pulse Secure\Integration\pulselauncher.exe" -stop. The timeline of the disclosure process began with an initial notice to DIVD on March 16, 2023, followed by a first reply from Ivanti regarding their responsible disclosure policy on March 20, 2023.

Further complicating the situation, CISA has reported that attackers have found workarounds to current mitigations for vulnerabilities in Ivanti Connect Secure VPN devices, with over 2,100 devices compromised in the attacks. These vulnerabilities, including CVE-2023-46805 and CVE-2024-21887, have been given severity scores of 8.2 and 9.1 out of 10.0, respectively. CISA recommends additional steps for customers to avoid being compromised or to minimize damage.

H. Escalating privileges

To escalate privileges and gain full control over a system, an attacker can exploit vulnerabilities that allow for privilege escalation. One common method is to manipulate access tokens, which are objects that describe the security context of a process or thread, including the identity and privileges of the user account associated with the process. By obtaining a token with higher privileges, an attacker can create a new process with elevated rights or replace the token of an existing process. A write-what-where condition is a vulnerability that allows an attacker to write an arbitrary value to an arbitrary location in memory. This can be exploited to overwrite critical data structures or function pointers, leading to arbitrary code execution.

In the context of the Ivanti Secure Access VPN vulnerabilities, CVE-2023-38043, CVE-2023-35080, and CVE-2023-38543, the exploitation process involves stopping the VPN client to avoid memory corruptions and then using the vulnerabilities to escalate privileges. The vulnerabilities allow for privilege escalation due to a kernel driver installed by the VPN software that creates a device readable and writable by any user, potentially leading to kernel corruption or privilege escalation.

The exploitation process may involve finding the kernel pointer for the token object using the SystemExtendedHandleInformation class in the NtQuerySystemInformation API and then using a write primitive to overwrite the TOKEN->_SEP_TOKEN_PRIVILEGES->Enabled and TOKEN->_SEP_TOKEN_PRIVILEGES->Present fields to grant system-level privileges to the process. This can be followed by spawning a shell with elevated privileges.

I. Enabling The Vulnerable Driver

To enable the vulnerable driver in Ivanti Secure Access VPN, which is typically disabled by default, an attacker can replicate the behavior that automatically starts the driver when a user connects to a VPN server with TDI fail-over enabled. This can be done by setting up a rogue Ivanti Secure Access VPN server and configuring it to use TDI fail-over.

- **Download an Image**: Obtain an Ivanti Secure Access VPN server VM image from the official site.
- **Install the Server**: Install the downloaded VM image on a Virtual Private Server (VPS) or locally. Ensure that you can point a domain name to it, such as vpn.rogue-server.com.
- **Complete the VM Setup**: Boot the VM image and complete the setup as prompted. After finishing, you can access the admin portal via the web.
- **Configure a Valid Certificate**: Obtain a valid certificate for a rogue server domain (e.g., vpn.rogue-server.com) using a service like Let's Encrypt. Upload the fullchain.pem and privkey.pem to the admin portal under System -> Configuration -> Certificates -> Device certificate. Delete any pre-configured self-signed certificate and configure your valid certificate to be used by the internal and external ports.
- **Restrict VPN & Configure TDI-Failover**: In the admin portal, navigate to Users -> User Roles -> Users. Uncheck all Access Features except for Secure Application Manager & Windows/Mac version sub-item. Then, enable Enable fail-over to TDI for Pulse SAM connection under the SAM -> Options tab.
- **Create a VPN User**: Go to Authentication -> Auth. Servers -> System Local -> Users tab and create a new user with a static username and password. This user will be used to connect to the rogue VPN.
- **Let Victim Connect to the Rogue Server**: Have the victim connect to the rogue server by providing the URL, username/password of the user you created, and the realm which that user is in (default is Users). Use the following command to connect:

  "%programfiles(x86)%\Common Files\Pulse Secure\Integration\pulselauncher.exe" -url YOUR_DOMAIN -u YOUR_USER -p YOUR_PASS -r Users

  For example:

  "%programfiles(x86)%\Common Files\Pulse Secure\Integration\pulselauncher.exe" -url vpn.rogue-server.com -u steve -p Welcome01! -r Users
- Stop the VPN Client: Before running the privilege escalation exploit, stop the VPN client to prevent memory corruptions using the command:

  "%programfiles(x86)%\Common Files\Pulse Secure\Integration\pulselauncher.exe" -stop

By following these steps, an attacker can enable the vulnerable driver and potentially exploit the vulnerabilities CVE-2023-38043, CVE-2023-35080, and CVE-2023-38543 in Ivanti Secure Access VPN to escalate privileges

IV. PoC "main.c"

The code relates to the Ivanti Pulse VPN Client Exploit for CVE-2023-35080 is designed to exploit a vulnerability in the Ivanti Secure Access Windows client, allowing for privilege escalation, denial of service (DoS), or information disclosure.

A. How the Code Works

- **Thread Priority Adjustment**: The code starts by attempting to set the current thread's priority to background mode to minimize its impact on the system's performance.
- **Memory Allocation and Configuration**: It allocates memory for various buffers (input_buffer, initial_buffer, buff_30h, iocsq_rsi_plus_8h) and configures them to construct a malicious payload. This includes setting up a pointer (buff_28h) to hold the byte value intended to be written into a vulnerable component within the driver's memory space.
- **Kernel Base Address Retrieval**: The code retrieves the base address of the kernel (ntoskrnl_base) to calculate the addresses of specific functions or offsets within the kernel that the exploit intends to manipulate.
- **Function Pointers Setting**: It sets up function pointers within the prepared buffers to point to malicious or controlled code segments or to trigger the vulnerability within the Ivanti Secure Access Client driver.
- **Triggering the Vulnerability**: The exploit triggers the vulnerability by making a DeviceIoControl call with the prepared input_buffer, which contains the malicious payload designed to exploit the vulnerability.
- **Privilege Escalation**: If successful, the exploit modifies the current process's token privileges or performs other unauthorized actions, leading to privilege escalation, DoS, or information disclosure.

B. Incoming Data:

- **Target Device Path**: The path to the vulnerable device or driver that the exploit targets.
- **Byte Value (what)**: The specific byte value that the exploit intends to write into the target memory location.
- **Target Memory Address (where)**: The memory address within the vulnerable component or driver where the exploit intends to write the byte value.

C. Outgoing Data/Result

- **Exploit Status Messages**: The code prints status messages indicating the success or failure of various steps, such as setting thread priority, creating threads, and executing the exploit.
- **Privileged Access**: If the exploit is successful, it achieves elevated privileges for the current process, allowing it to perform actions that were previously restricted.
- **Potential System Modification**: Depending on the exploit's intent, it could modify system settings, disable security measures, or perform other unauthorized actions as a result of the privilege escalation.

V. PoC “kernel.c”

The code targets a vulnerability in a system driver, likely related to the Ivanti Pulse VPN Client Exploit for CVE-2023-35080. The code is written in C and includes several functions that interact with the Windows operating system at a low level to manipulate device handles and memory.

A. How the Code Works

- **BuildDevicePath**: Constructs the device path string for the vulnerable driver.
- **OpenDevice**: Opens a handle to the device using the CreateFileW function, which allows for reading and writing to the device.
- **CloseDevice**: Closes the handle to the device and frees associated memory.
- **GetFunctionOffset**: Retrieves the offset of a function within the ntoskrnl.exe file, which is the Windows NT kernel.
- **GetKernelBase**: Determines the base address of the kernel by querying system information.
- **GetObjectPointedByHandle**: Retrieves the kernel object pointed to by a given handle, which could be used to manipulate or read information from that object.

B. Incoming Data

- DevicePath: A string representing the path to the vulnerable device or driver.
- DEVICE_NAME_W: The name of the device, which is used to construct the device path.
- hDevice: A pointer to a handle that will be used to interact with the device.
- fnName: The name of the function whose offset is being retrieved.
- h: a handle whose pointed object is being retrieved.

C. Outgoing Data/Result

- **DevicePath**: The full device path string that is constructed and used to open a handle to the device.
- **hDevice**: The handle obtained by opening the device, which can be used for further interaction with the device.
- **FnOffset**: The offset of the specified function within the kernel's executable image.
- **KernelBase**: The base address of the kernel obtained from the system information.
- **Object**: The kernel object pointed to by the specified handle, which can be manipulated or read.

The code is designed to perform low-level operations that are typically part of an exploit chain. These operations include opening a handle to a vulnerable driver, determining the location of certain functions or data within the kernel, and potentially using this information to manipulate the system in a way that exploits the vulnerability.