

HTB Sherlock -MisCloud Writeup

Description:

My name is John. I am a student who started an e-commerce startup business named "DummyExample" with my partner, James. Initially, I was using WordPress and shared hosting. After experiencing good traffic, I decided to migrate from **WordPress** to a customized website on **Google Cloud Platform (GCP)**. Currently, my partner and I are working on the website, contributing to a **Gitea server** hosted on GCP. I migrated all customer data to cloud storage. Recently, my **data was breached**, and I have no clue how it happened or what was vulnerable. My GCP infrastructure consists of **five VM instances** and a **single Cloud Storage**. There is **one Windows machine** for my partner to use, with very restricted permissions over GCP, only allowing access to his Gitea account. I have **two Linux machines** for my work, one for hosting the **Gitea server** and another for **packet mirroring**. All the machines have public IPs but very restricted access due to firewalls in place. Due to budget constraints, I can't use the Google Security Command Center service, so I am providing you with the **VPC network traffic capture** and the **Google Cloud logs**.

Solution:

I am given a zip file to download and after downloading it, I unzip the file with the password given: **hacktheblue**. Inside the folder are two files named **GCloud_Logs.json** and **VPCNetwork_Traffic.pcap**. To analyse the files, I use Visual Studio Code to view the JSON file and Wireshark to view the PCAP files.

Task 1: What's the private IP address of the Windows machine?

After researching, I found that private IP addresses for Windows machines are usually within the ranges:

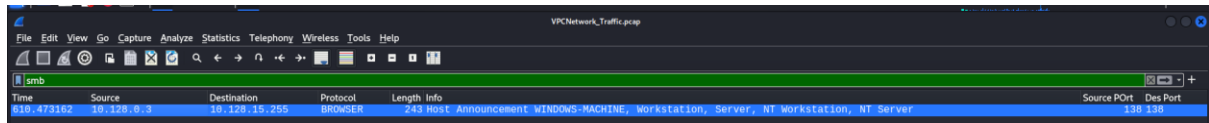
- 10.x.x.x
- 172.16.x.x - 172.31.x.x
- 192.168.x.x

I started searching in the JSON file for the keyword "Windows" and found two public addresses related to it, 169.150.196.101 and 154.198.108.200. However, no private key can be found. Therefore, I move to analyse the PCAP files.

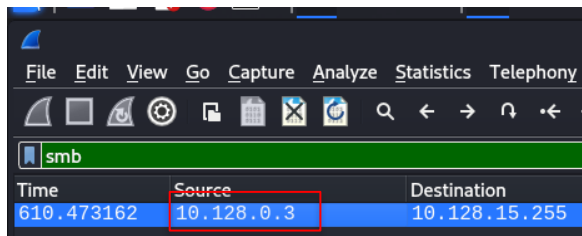
For the PCAP files, I tried to filtering for **SMB** protocols.

Why SMB?

- SMB (Server Message Block) is a protocol primarily used by Windows systems for file sharing and network services.
- Windows machines often broadcast Host Announcements using the BROWSER protocol within SMB traffic, revealing key details like hostname and private IP.



Time	Source	Destination	Protocol	Length	Info	Source Port	Des Port
610.473162	10.128.0.3	10.128.15.255	BROWSER	243	Host Announcement WINDOWS-MACHINE, WORKSTATION, Server, NT Workstation, NT Server	138	138



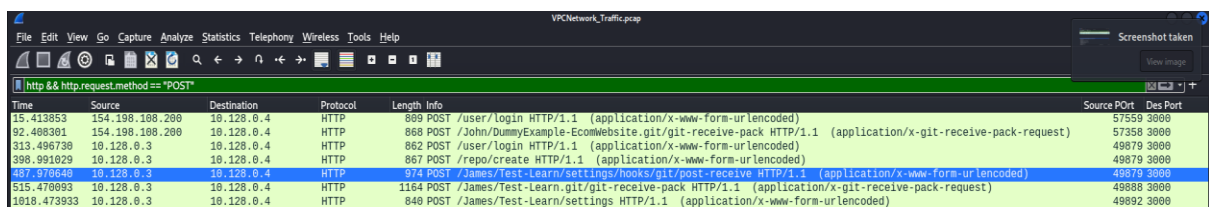
Time	Source	Destination
610.473162	10.128.0.3	10.128.15.255

After filtering, a single SMB packet was found. We can see that the source IP, 10.128.0.3, belongs to the **private IP range** and matches the description of the **Windows** machine.

The answer for task 1 is: **10.128.0.3**

Task 2: Which CVE was exploited by the threat actor?

For this task, I continue to explore in Wireshark. I first filter **http && http.request.method == "POST"** to see if there is any attempt of exploitation by uploading any suspicious file. After filtering, I got 7 packets:



Time	Source	Destination	Protocol	Length	Info	Source Port	Des Port
15.413653	154.198.108.200	10.128.0.4	HTTP	809	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)	57559	3000
92.488301	154.198.108.200	10.128.0.4	HTTP	868	POST /John/DummyExample-EcomWebsite.git/git-receive-pack HTTP/1.1 (application/x-git-receive-pack-request)	57358	3000
313.496730	10.128.0.3	10.128.0.4	HTTP	862	POST /user/login HTTP/1.1 (application/x-www-form-urlencoded)	49879	3000
398.991029	10.128.0.3	10.128.0.4	HTTP	867	POST /repo/create HTTP/1.1 (application/x-www-form-urlencoded)	49879	3000
487.978640	10.128.0.3	10.128.0.4	HTTP	974	POST /James/Test-Learn/settings/hooks/git/post-receive HTTP/1.1 (application/x-www-form-urlencoded)	49879	3000
515.478093	10.128.0.3	10.128.0.4	HTTP	1164	POST /James/Test-Learn.git/git-receive-pack HTTP/1.1 (application/x-git-receive-pack-request)	49888	3000
1018.473933	10.128.0.3	10.128.0.4	HTTP	840	POST /James/Test-Learn/settings HTTP/1.1 (application/x-www-form-urlencoded)	49892	3000

I feed this data to ChatGPT to analyse and help me understand what activity is happening here. I will do a breakdown of the activity here:

1. According to Packet 1, there is a suspicious login attempt from an external ip 154.198.108.200 to the web application hosted on 10.128.0.4.
2. From Packet 2, we can see that the attacker is interacting with a Git repository. This could involve pushing malicious code or exploiting vulnerabilities in Git functionality.

The answer for task 2 is: **CVE-2020-14144**

A little info about CVE-2020-14144

Cause: The vulnerability exists due to **improper input validation** in GiTea, specifically in the way the application handles certain user inputs. It enables an **authenticated attacker** (one who has logged in or has access to the GiTea system) to inject malicious code that can be executed on the server, leading to remote code execution (RCE).

Exploitation: An attacker can exploit this vulnerability by providing crafted inputs, which are then improperly processed by GiTea, allowing them to run arbitrary commands on the server with the same privileges as the GiTea application.

Task 3: What is the hostname and port number to which the reverse shell was connecting?

First, I familiarised myself with the CVE and the reverse shell mechanism that it exploits. A key point here was the use of a reverse shell payload triggered by certain actions on the target system.

Upon studying the CVE and its associated reverse shell code, which was available on GitHub, I recognised that the Python code provided in the repository contained a function named **repo_set_githook_post_receive**. This function was responsible for setting up a post-receive hook in the Git repository, which would trigger the reverse shell.

```
def repo_set_githook_post_receive(self, repository_name, content):
    if self.verbose:
        print("  [>] repo_set_githook_post_receive('%s')" % repository_name)
    csrf = self._get_csrf('%s/%s/%s/settings/hooks/git/post-receive' % (self.host, self.username, repository_name))
    # Set post receive git hook
    r = self.session.post(
        '%s/%s/%s/settings/hooks/git/post-receive' % (self.host, self.username, repository_name),
        data={
            '_csrf': csrf,
            'content': content
        }
    )
    return
```

From the packets being filtered earlier, one packet stood out because it referred to the directory **/James/Test-Learn/settings/hooks/git/post-receive**. This directory was directly related to the post-receive hook as described in the reverse shell Python code.

From the decoded reverse shell command, I was able to identify two crucial pieces of information:

- The **host** is 0.tcp.ngrok.io
- The **port** is 14509

Therefore, the answer for task 3 is: **0.tcp.eu.ngrok.io:14509**

Task 4: From which IP address was the CVE exploited, and is this threat an insider or outsider attack?

According to the analysis we did earlier in task 2, the answer for task 4 is:
10.128.0.3:insider

Task 5: Which account helped the threat actor to pivot?

I initially tried to find email addresses in the packet capture (PCAP) file, but only found incorrect information. After searching the JSON logs, I discovered the service account email in an audit log. This log shows that the service account was used to list storage buckets in Google Cloud Storage, which is typically part of a reconnaissance step where an attacker gathers information about available resources.

```
{
  "protoPayload": {
    "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
    "status": {},
    "authenticationInfo": {
      "principalEmail": "257145238219-compute@developer.gserviceaccount.com",
      "serviceAccountDelegationInfo": [
        {
          "firstPartyPrincipal": {
            "principalEmail": "service-257145238219@compute-system.iam.gserviceaccount.com"
```

The service account had delegated access to another account, **service-257145238219@compute-system.iam.gserviceaccount.com**, which suggests that the attacker might have escalated their privileges or moved laterally within the environment. The attacker used legitimate tools like **gsutil** (part of the Google Cloud SDK) to interact with the system, making the activity harder to detect.

This service account played a key role in the attacker's ability to explore the environment and potentially access sensitive data, marking it as the account that helped the attacker pivot.

Therefore, the answer for task 5 is: 257145238219-compute@developer.gserviceaccount.com

Task 6: Which machines did the threat actor log into? (sorted alphabetically)

For this task, I filter the keyword "LOGIN" and I found three different instances that the threat actor has log into: **packet-mirror-instance**, **linux-machine2** & **linux-machine1**.

```
{
  "resourceName": "projects/qwiklabs-gcp-00-848c1b920007/zones/us-central1-a/instances/packet-mirror-instance",
  "request": {
    "@type": "type.googleapis.com/google.cloud.oslogin.v1.CheckPolicyRequest",
    "zone": "us-central1-a",
    "instance": "packet-mirror-instance",
    "numericProjectId": "257145238219",
    "email": "11560335204330737157",
    "projectId": "qwiklabs-gcp-00-848c1b920007",
    "policy": "LOGIN"
  }
},
```

```
"resourceName": "projects/qwiklabs-gcp-00-848c1b920007/zones/us-central1-a/instances/linux-machine2",
"request": {
  "instance": "linux-machine2",
  "policy": "LOGIN",
  "zone": "us-central1-a",
  "@type": "type.googleapis.com/google.cloud.oslogin.v1.CheckPolicyRequest",
  "email": "109960989495683138790",
  "projectId": "qwiklabs-gcp-00-848c1b920007",
  "numericProjectId": "257145238219",
  "serviceAccount": "257145238219-compute@developer.gserviceaccount.com"
},
```

```

"resourceName": "projects/qwiklabs-gcp-00-848c1b920007/zones/us-central1-a/instances/linux-machine1",
"request": {
  "email": "109960989495683138790",
  "instance": "linux-machine1",
  "zone": "us-central1-a",
  "numericProjectId": "257145238219",
  "policy": "LOGIN",
  "projectId": "qwiklabs-gcp-00-848c1b920007",
  "serviceAccount": "storage-svc-acc@qwiklabs-gcp-00-848c1b920007.iam.gserviceaccount.com",
  "@type": "type.googleapis.com/google.cloud.oslogin.v1.CheckPolicyRequest"
},

```

For the answer, we just have to put it in alphabetical order without space in between the machine names: **linux-machine1,linux-machine2,packet-mirror-instance**

Task 7: What's the original name of the sensitive file?

While initially looking through the logs, I found a resource name pointing to a Google Cloud Storage bucket: **projects/_/buckets/sensitive-ecomuser-data**.

This bucket name, **sensitive-ecomuser-data**, suggested that it might contain sensitive files.

```

"authorizationInfo": [
  {
    "resource": "projects/_/buckets/sensitive-ecomuser-data",
    "permission": "storage.objects.list",
    "granted": true,
    "resourceAttributes": {}
  }
],

```

I performed keyword search again for the directory and put a slash behind the directory (**projects/_/buckets/sensitive-ecomuser-data/**) to find the sensitive file name.

```

"serviceName": "storage.googleapis.com",
"methodName": "storage.objects.get",
"authorizationInfo": [
  {
    "resource": "projects/_/buckets/sensitive-ecomuser-data/objects/Custom-Data-e7b9e806c08435793e310d7137b068fa.xlsx",
    "permission": "storage.objects.getIamPolicy",
    "granted": true,
    "resourceAttributes": {}
  }
],

```

The answer for task 7 is: **Customer-Data-e7b9e806c08435793e310d7137b068fa.xlsx**

Task 8: Which gcloud role did the threat actor try to assign to the storage bucket to make it publicly accessible?

I tried filtering for “role” and found the name of the role the threat actor try to assign:

```
"policyDelta": {
  "bindingDeltas": [
    {
      "action": "ADD",
      "role": "roles/storage.legacyObjectReader",
      "member": "allUsers"
    }
  ]
}
```

The answer for task 8 is: **roles/storage.legacyObjectReader**

Task 9: Which account led to the cloud storage data breach?

During my investigation, I reviewed the logs and identified an email associated with the access to the sensitive file (Customer-Data-e7b9e806c08435793e310d7137b068fa.xlsx). The email **storage-svc-acc@qwiklabs-gcp-00-848c1b920007.iam.gserviceaccount.com** was found to be linked to the initial access to the file.

```
4469     "authenticationInfo": {
4470       "principalEmail": "storage-svc-acc@qwiklabs-gcp-00-848c1b920007.iam.gserviceaccount.com",
4471       "serviceAccountDelegationInfo": [
4472         {
4473           "firstPartyPrincipal": {
4474             "principalEmail": "service-257145238219@compute-system.iam.gserviceaccount.com"
4475           }
4476         }
4477       ]
4478     },
4479     "requestMetadata": {
4480       "callerIp": "34.42.164.212",
4481       "callerSuppliedUserAgent": "apitools Python/3.11.8 gsutil/5.29 (linux) analytics/disabled interactive/True command/acl google-clou",
4482       "callerNetwork": "//compute.googleapis.com/projects/qwiklabs-gcp-00-848c1b920007/global/networks/__unknown__",
4483       "requestAttributes": {
4484         "time": "2024-06-16T09:53:53.208257944Z",
4485         "auth": {}
4486       },
4487       "destinationAttributes": {}
4488     },
4489     "serviceName": "storage.googleapis.com",
4490     "methodName": "storage.objects.get",
4491     "authorizationInfo": [
4492       {
4493         "resource": "projects/_/buckets/sensitive-ecomuser-data/objects/Customer-Data-e7b9e806c08435793e310d7137b068fa.xlsx",
4494         "permission": "storage.objects.getIamPolicy",
```

The answer for task 9 is: storage-svc-acc@gwiklabs-gcp-00-848c1b920007.iam.gserviceaccount.com

Task 10: Which port number was exploited by the attacker to exfiltrate data that is allowed by default ingress traffic rules in the default VPC network?

I focused on HTTP traffic, specifically the **GET** requests, as this is often a method attackers use to request files or data. Upon reviewing the GET requests, I identified a file name **cusdata.xlsx.enc**, which appears to be a file of interest to the attacker. The **.enc** extension indicates that the file is encrypted, possibly to hide its content during transfer, which is a common tactic in data exfiltration.

[http.request.method == "GET"]					
Time	Source	Destination	Protocol	Length	Info
2.727623	154.198.108.200	10.128.0.4	HTTP	538	GET /js/index.js?v=62abc69fd3fad42d4dadca97c5d57105 HTTP/1.1
1024.303330	10.128.0.3	10.128.0.4	HTTP	535	GET / HTTP/1.1
0.827132	154.198.108.200	10.128.0.4	HTTP	535	GET /vendor/plugins/cssrelpreload/loadCSS.min.js HTTP/1.1
2.204453	154.198.108.200	10.128.0.4	HTTP	533	GET /vendor/plugins/clipboard/clipboard.min.js HTTP/1.1
2.478652	154.198.108.200	10.128.0.4	HTTP	531	GET /vendor/plugins/semantic/semantic.min.js HTTP/1.1
15.689391	154.198.108.200	10.128.0.4	HTTP	530	GET / HTTP/1.1
2.149280	154.198.108.200	10.128.0.4	HTTP	529	GET /vendor/plugins/emojify/emojify.min.js HTTP/1.1
1.941385	154.198.108.200	10.128.0.4	HTTP	527	GET /vendor/plugins/autolink/autolink.js HTTP/1.1
1.121034	154.198.108.200	10.128.0.4	HTTP	527	GET /vendor/plugins/jquery/jquery.min.js HTTP/1.1
2.416188	154.198.108.200	10.128.0.4	HTTP	521	GET /vendor/plugins/vue/vue.min.js HTTP/1.1
963.870182	169.150.196.101	10.128.0.7	HTTP	519	GET /cusdata.xlsx.enc HTTP/1.1
221.742979	10.128.0.3	10.128.0.4	HTTP	518	GET /ima/aitea-la.bna HTTP/1.1

To determine which port was used to transfer the file, I checked the **destination port** of the GET request. The destination port in this case was **3389**, a port traditionally used by the **Remote Desktop Protocol (RDP)**.

Frame 20270: 519 bytes on wire (4152 bits), 519 bytes captured (4152 bits) on interface unknown, id 0
Ethernet II, Src: 42:01:0a:80:00:01 (42:01:0a:80:00:01), Dst: 42:01:0a:80:00:05 (42:01:0a:80:00:05)
Internet Protocol Version 4, Src: 169.150.196.101, Dst: 10.128.0.7
Transmission Control Protocol, Src Port: 34880, Dst Port: 3389, Seq: 1, Ack: 1, Len: 453
Hypertext Transfer Protocol

The answer for task 10 is: **3389**

Interesting Fact:

Port 3389 is typically used for RDP, which allows remote access to a machine. While RDP is typically used for administrative purposes, its use in this context suggests that the attacker may have exploited a remote access session to transfer the sensitive file. Port 3389 is often allowed by default ingress traffic rules in many VPC configurations, making it an attractive target for attackers, as it may bypass security controls.

Task 11: What is the key to decrypt the encrypted file?

After finding the file being exfiltrated, I continued to follow the TCP stream of the request and I found this:

[illegible]

I can see that there is a pattern of l33t code words that looks like the key.

The answer for task 11 is: **J@m37 h@Rd3st k3Y enCrypt Exf!!7r@73**

Task 12: What are the SSN and credit card numbers of "Founder John"?

To get the SSN and credit card numbers, I will have to decrypt the **cusdata.xlsx.enc** file first. By inspecting back the **GET** requests, I found a file name **file-xor.py** and thought it seemed suspicious.

830.689073	10.128.0.3	10.128.0.4	HTTP	619 GET /James/Test-Learn HTTP/1.1
495.613011	10.128.0.3	10.128.0.4	HTTP	257 GET /James/Test-Learn.git/info/refs?service=git-receive-pack HTTP/1.1
514.940916	10.128.0.3	10.128.0.4	HTTP	340 GET /James/Test-Learn.git/info/refs?service=git-receive-pack HTTP/1.1
835.929790	10.128.0.3	10.128.0.4	HTTP	642 GET /James/Test-Learn/raw/master/file-xor.py HTTP/1.1
850.751075	10.128.0.7	10.128.0.4	HTTP	184 GET /James/Test-Learn/raw/master/file-xor.py HTTP/1.1
493.105737	10.128.0.3	10.128.0.4	HTTP	628 GET /James/Test-Learn/settings HTTP/1.1
986.356730	10.128.0.3	10.128.0.4	HTTP	620 GET /James/Test-Learn/settings HTTP/1.1

I proceed to export out the python file and take a look at the code.

```

import sys

def xor_file_with_password(filename, password):
    password_bytes = password.encode()
    password_len = len(password_bytes)

    with open(filename, 'rb') as f:
        file_data = f.read()

    xored_data = bytearray(file_data)

    for i in range(len(file_data)):
        xored_data[i] ^= password_bytes[i % password_len]

    return xored_data

def write_xored_file(output_filename, xored_data):
    with open(output_filename, 'wb') as f:
        f.write(xored_data)

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print(f"Usage: {sys.argv[0]} <filename> <password> <output_filename>")
        sys.exit(1)

    filename = sys.argv[1]
    password = sys.argv[2]
    output_filename = sys.argv[3]

    xored_data = xor_file_with_password(filename, password)
    write_xored_file(output_filename, xored_data)

    print(f"File '{filename}' has been XORed with the password and saved as '{output_filename}'")

```

From the code, we can see that it performs **XOR encryption** on a file using a password. XOR encryption works by XOR-ing the file's bytes with a repeating pattern derived from the password. Hence, this python script can be used to encrypt and decrypt files with the same password.

To decrypt **cusdata.xlsx.enc** file, we can use the command:

Python3 file-xor.py <encrypted_file> <password> <decrypted_file>

Python3 file-xor.py cusdata.xlsx.enc

J@m37_h@Rd3st_k3Y_enCrypt_Exf!l7r@73 decrypted_file.xlsx

```

PS C:\misccloud> python3 .\file-xor.py .\cusdata.xlsx.enc J@m37_h@Rd3st_k3Y_enCrypt_Exf!l7r@73 decrypted_file.xlsx
File '.\cusdata.xlsx.enc' has been XORed with the password and saved as 'decrypted_file.xlsx'
PS C:\misccloud> |

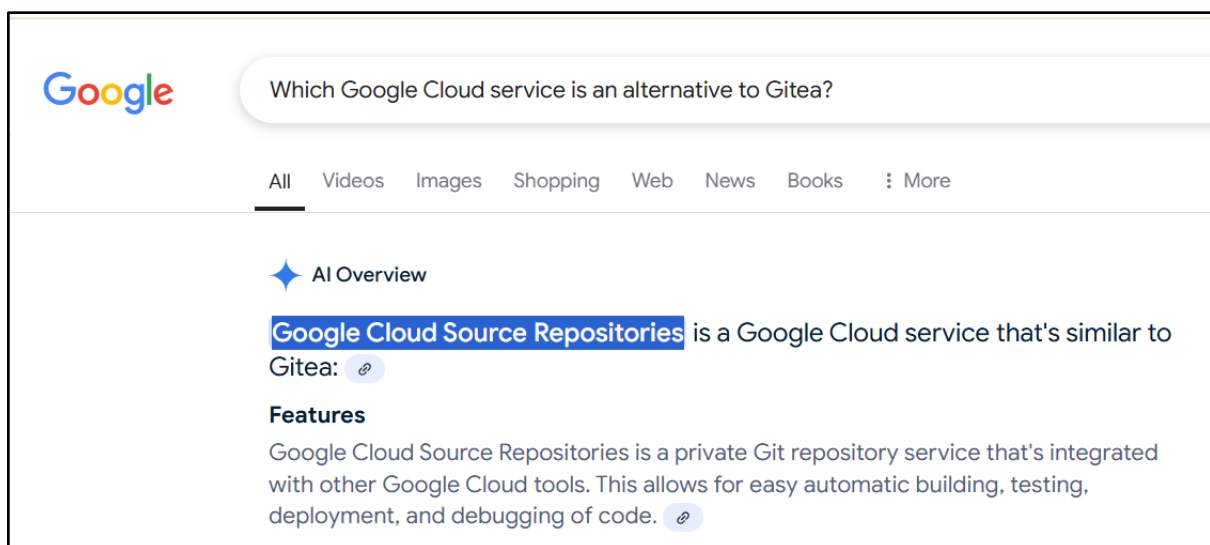
```

Inside **decrypted_file.xlsx** we can see all SSN and Credit Card Numbers for users. For optimal compatibility, it is recommended to open the file on a **Windows operating system**, as it natively supports **.xlsx** files. Alternatively, if you are using a **Linux terminal**, you can use **OpenOffice** to access the contents.

	A	B	C
1	First and Last Name	SSN	Credit Card Number
2			Visa MC AMEX
3	Robert Aragon	489-36-8350	4929-3813-3266-4295
4	Ashley Borden	514-14-8905	5370-4638-8881-3020
5	Founder John	HTB-FR-SRLK	1111-3345-1234-5123
6	Thomas Conley	690-05-5315	4916-4811-5814-8111
7	Susan Davis	421-37-1396	4916-4034-9269-8783
8	Christopher Diaz	458-02-6124	5299-1561-5689-1938
9	Rick Edwards	612-20-6832	5293-8502-0071-3058
10	Victor Faulkner	300-62-3266	5548-0246-6336-5664
11	Lisa Garrison	660-03-8360	4539-5385-7425-5825
12	Marjorie Green	213-46-8915	4916-9766-5240-6147
13	Mark Hall	449-48-3135	4556-0072-1294-7415

The answer for task 12 is: **HTB-FR-SRLK:1111-3345-1234-5123**

Task 13: Which Google Cloud service is an alternative to Gitea?



Google

Which Google Cloud service is an alternative to Gitea?

All Videos Images Shopping Web News Books : More

AI Overview

Google Cloud Source Repositories is a Google Cloud service that's similar to Gitea: [🔗](#)

Features

Google Cloud Source Repositories is a private Git repository service that's integrated with other Google Cloud tools. This allows for easy automatic building, testing, deployment, and debugging of code. [🔗](#)

The answer for task 13 is: **Cloud Source Repositories**

Task 14: Is it safe to use the Default Compute Engine Service Account on VM instances?

Risks of Using the Default Account:

- **Excessive Permissions:** It often has more permissions than needed, increasing the attack surface.
- **Security Best Practices:** It violates the **principle of least privilege**, making it easier for attackers to misuse excessive access.

- Audit Challenges: It's harder to track resource usage and permissions with a default account.
- Lack of Isolation: Multiple VMs using the same default account may share unnecessary access to resources.

The answer for task 14 is: **No**

Task 15: Which Google Cloud service restricts data exfiltration from Cloud Storage?

The answer for task 15 is: **VPC Service Controls**

A little info about VPC Service Controls

VPC-SC creates isolation perimeters around Google Cloud resources and networks. You can configure these perimeters to control data exfiltration across the perimeter boundary.