Global Information Assurance Certification Paper

Lateral traffic movement in Virtual Private Clouds

Author: Andy Huang (tokidoki2@gmail.com)
Advisor: *Clay Risenhoover*

Accepted: December 5th, 2019

Abstract

Cloud vendors have introduced virtual private cloud (VPC) structures to bring the benefits of private cloud into the public cloud. These structures provide vertical segmentation and isolation for application projects implemented within them. However, the security context needs to be considered as applications communicate with one another between VPCs using technologies such as peering and privatelinks. Applications are usually highly dependent on each other for data and functionality, leading to cross-connections between VPC structures. The implications between different connection setups need to be vetted to ensure that access is not overly permissive, thus leading to possible lateral movement of traffic.

1. Introduction

On-premise network structures communication zones by grouping similar functionalities within network segments. These subnets encompass application functionality from different projects and include applications providing front-end web interfaces, middle-tier application logic, and persistent data sets. This type of horizontal structure groups application components into functionality layers. An application subnet can contain many different applications from different projects. Additional segmentation within the layer is usually implemented based on compliance or regulatory needs. Since all projects ultimately belong to the organization, centralized protection is usually deployed at the edge to protect all resources within the network. The organization thus controls all assets related to such a design, from server hardware to network routers.

The advent of virtualization enabled the capability to create more virtual systems on virtualization clusters, however, the clusters tend to fit within the horizontal layers as well. The concept of the private cloud relates to this definition of exclusive use of network and server structure by the organization because the private cloud is owned, managed, and operated by the organization (Mell, Grance, 2011).

Cloud vendors implement software-defined networks, to allow for multi-tenancy and scalability since they maintain the hardware and software assets for both systems and networking. The public cloud defines the infrastructure for open use by the general public since it is managed by the business organization but exists on the premises of a cloud provider. (Mell, Grance, 2011). Virtual Private Cloud (VPC) is a concept from cloud providers to create a network structure in the public cloud area that allows organizations to treat it as a private cloud.

Traditional public cloud share network traffic between customers and, when restricted, still shares traffic between projects for one customer. This design creates noisy neighbor conditions that can be controlled by deploying client-owned equipment in an on-premise or co-located fashion. This remediation has inherent drawbacks in that the organization is still expending resources for capital equipment to maintain isolation, and it also creates the same horizontal application structure. A VPC allows the same concepts and controls to be deployed on the cloud service provider level, with control given to the

customer. Therefore, VPC uses a combination of controls to address the vulnerabilities and weaknesses that are present when clients operate their own equipment (Jackson, 2018).

The VPC design has led to deployment approaches that implement each application into its Virtual Private cloud to provide isolation and independence. The horizontal segmentation based on functionality is no longer applicable in this structure as project development teams take advantage of the flexibility offered by the cloud. Development teams favor the concept of implementing applications into dedicated VPCs since it increases isolation and changes related to scaling. In addition, changes are implementated without impact to other teams, minimizing noisy neighbor issues. Cloud providers also offer tools which allow application teams to operate the VPC without the need for a centralized team. The resulting pattern creates a vertical stack application network structure, where each application defines its layers, and projects do not share common network layers.

1.1. Horizonal stack and Vertical stack

Traditional client-server and multi-tier applications create network structures that are segmented via functional layers. The standard three-tier design focuses on a web tier, application tier, and data persistence tier shown in Figure 1. Multiple projects share the overall tier structure, resulting in a horizontal approach. The network team may opt to separate individual tiers into multiple zones to separate applications, databases, and web systems due to compliance reasons. This additional separation does not necessarily isolate down to the system level; instead, this creates a compliance zone that has three layers within it, and all compliance related systems are allocated to this zone. Such horizontal structuring is common with on-premise datacenter and networks and forms traditional network design.

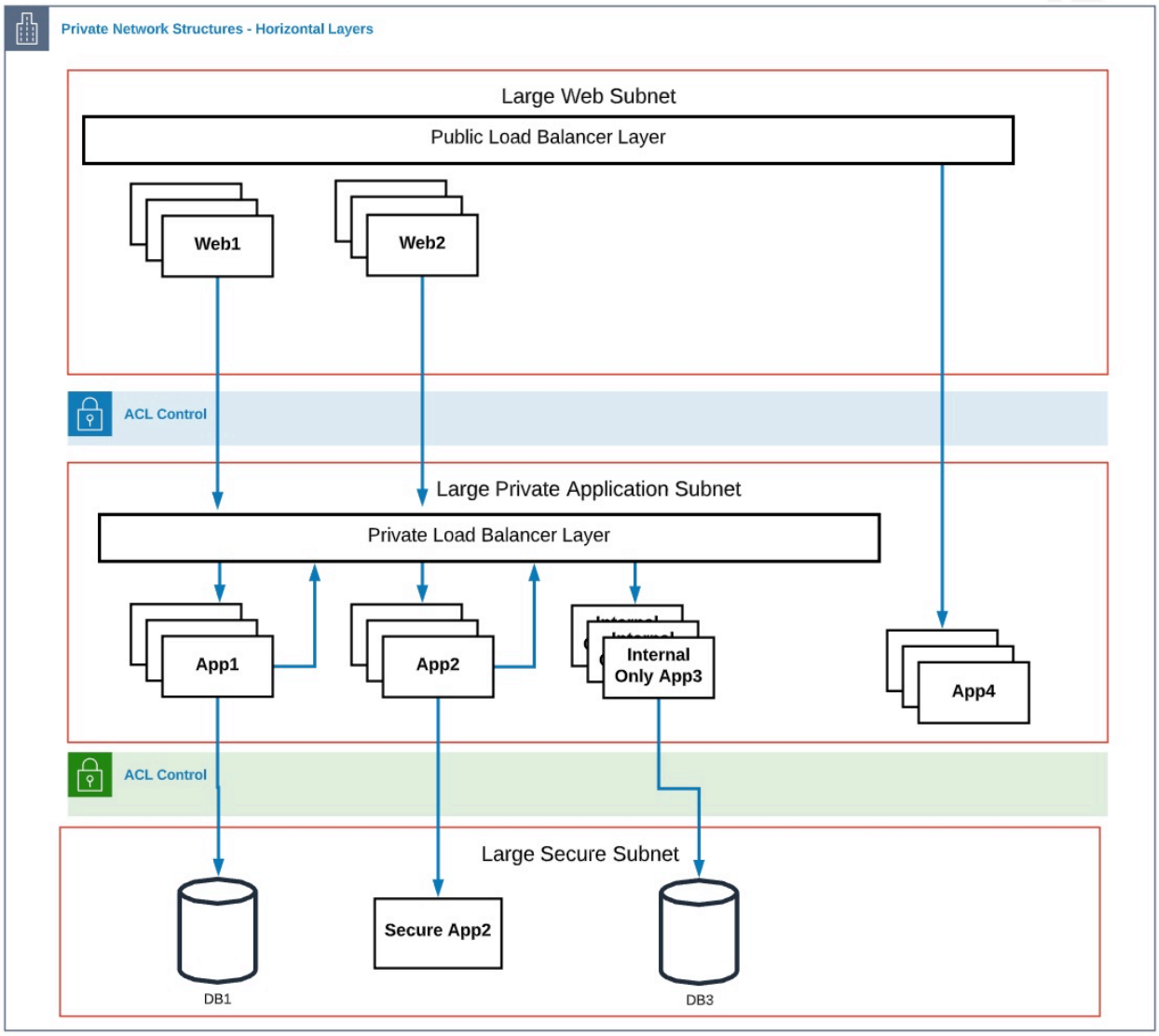

Figure 1: Horizontal Stack Layout – Large Subnets with multiple projects.

Many organizations have opted to move into a vertical stack structure when setting up a new project or migrating an existing project into the cloud. Virtual Private Cloud structures allow development teams to define and create a network space and allocate subnets for use independently. Therefore, each project may opt to have their network structure to isolate themselves. Cloud providers utilize this as a way to isolate customers, and customers increasingly have opted to use this mechanism to isolate projects or additional tenants within the account. This design affords each team within the organization the ability to stand up their network to support its own set of systems and applications. In Figure 2, this structure shows that each project operates by itself. The

application team now can define and react to the needs of the application. Any changes related to the flow structure can also be modified relatively quickly by the project team without impact or risk to another project.

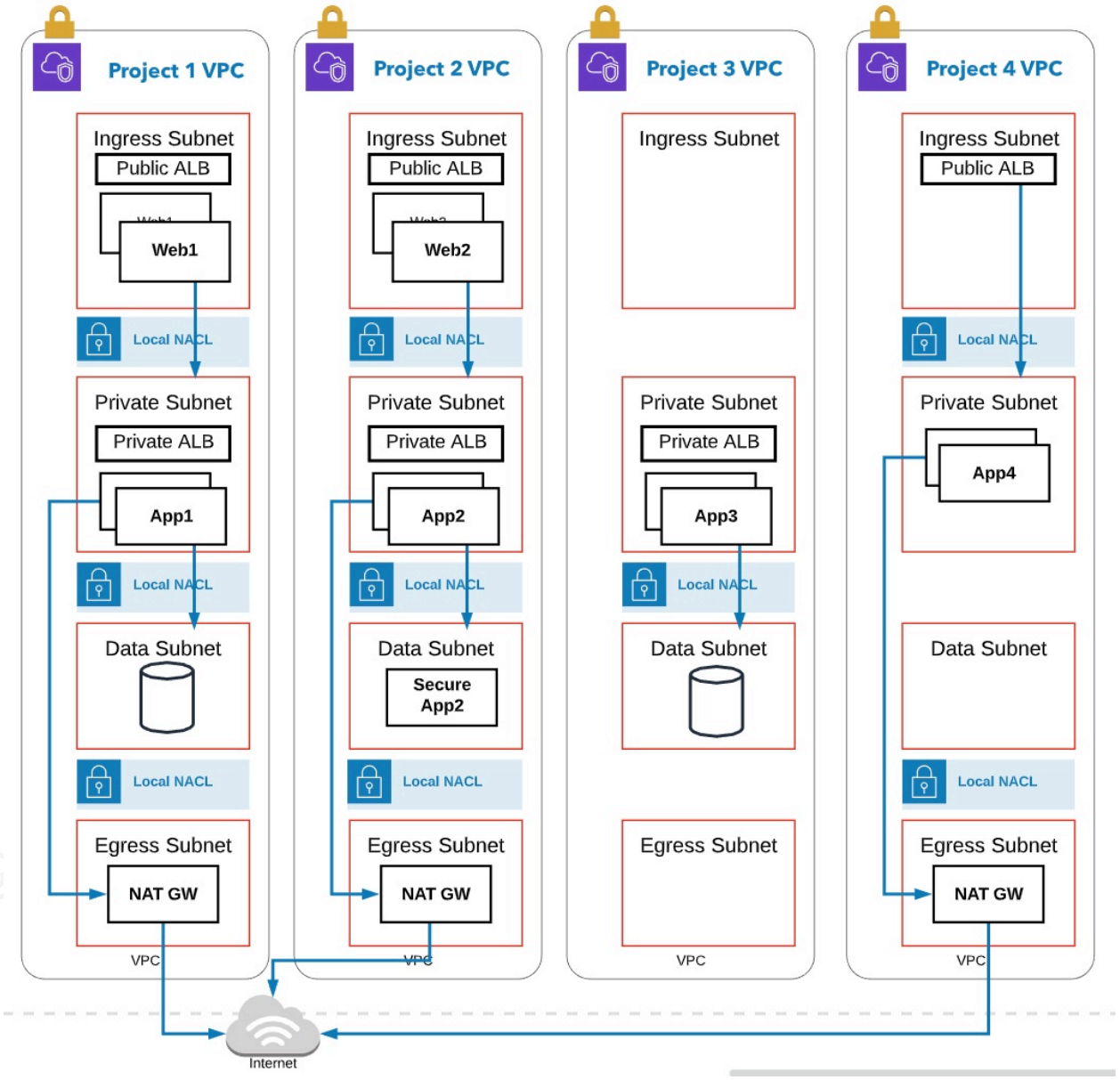

Figure 2: Complete independent project structures within several isolated VPCs.

1.2. Vertical stack challenges

While the vertical segmentation model provides applications with independence, it also creates possible boundary issues. When applications need to communicate with other applications, the interaction can be different than the horizontal stacking application models. Vertical structures guard against lateral movement of traffic between VPCs since each application is in its own set of subnets and does not have access to the other subnets. This control makes the access explicit, and specific steps must be taken to allow access. The result of this access restriction increases security stance dramatically, and also increases the adoption of technologies such as API gateways. However, it does present a challenge in cross-application communications.

The toolsets provided by the cloud provider makes it very easy for any engineer to create network structures; however, this simplicity and ease also overshadows security and networking best practices. Best practices that span across cloud and on-premise designs such as Network Access Control List (NACL) and segregation of deployment zones must still be used to define boundary areas between VPCs (Priyam, 2018). Cloud vendors have recognized the need for secure communications between applications in independent VPC structures and introduced tools and methods. The ideas that control segmentation from an on-premise design, such as subnet definition, route tables, and network Access Control Lists (ACLs) are exposed as configurable functionalities of a VPC. However, the ease with which independent teams can establish communication channels may not follow best practice and can present security concerns. Improper allocation of permissions can cause overexposure between accounts and allow for unintended lateral movement across VPCs. The challenges to the security team is to find and ensure that distributed teams properly configure and implement access controls, especially across VPCs.

2. Virtual Private Cloud (VPC)

2.1. VPC structures.

Across the three major cloud providers, Amazon AWS, Google Cloud, and Microsoft Azure, VPC structures have a common pattern. They all provide the ability to define a large private address space and the reallocation of the address space into smaller subnets. The subnets can define different functions of the application and therefore separate different components of the application. VPCs also offer the ability to create network access control lists, connectivity options to on-premise data centers, cross-region connections within accounts, security control groups, and connections to other VPCs.

Microsoft's Virtual Network technology provides a similar concept to Amazon AWS VPC technology with many of the features directly correlating with one another ("MS VirtualNet," 2019) Google Cloud's Virtual Private Cloud provides some additional differences where one VPC can span multiple regions. This difference in design from Google Cloud allows for global load balancing using a true static public ip address. This approach is in contrast to a DNS based solution adopted by AWS and Microsoft Azure which requires the use of naming schemes to achieve load balancing ("GCP CLB," 2019).

Per an AWS VPC use case scenario, a VPC is divided into four different functionality areas: ingress, private, data, and egress areas, each consists of a subnet. This set of subnets provide critical functionality for a standard multi-tier application. Ingress subnets support inbound traffic from the internet, and also contains the bastion host as a hardened barrier for internal systems access. The ingress subnet offers public-facing load balancers that provide load balancing to the application servers in the private subnet. Application systems handle business logic in the private subnet. Databases and other applications that persist data reside in the data subnet which provides isolation with a dedicated access control. Application traffic that needs to reach into the internet go through NAT gateways that is situated in the egress subnet ("AWS VPC," 2019).

2.2. VPC internal isolation

VPC access control is typically done via security groups and Network Access Control Lists (NACL). In AWS, route tables associated with each subnet can help direct traffic and provide inbound and outbound traffic control between subnets ("AWS Subnets," 2019). Route tables work with NACLs to provide stateless control, inbound and outbound rules have to be explicitly defined across both the source and target. Beyond network access control lists, security groups provide more defined access control by allowing the control of traffic inbound and outbound between a group of applications ("AWS SG," 2019). The concept of security groups extends even within one subnet; therefore, it is possible to deny access between groups of applications even within one subnet. This design control allows for very defined and specialized access. AWS security groups are used extensively to define finer grain access control and control access to resources. Google Cloud goes a step further by utilizing routes and tags instead of security groups. Those systems with specific or similar tags can utilize particular routes associated with those tags. This approach provides finer grain control down to the system resource level. Google Cloud's implementation reduces the overhead of creating security groups, by directly controlling access to the systems based on the tags of those systems (El-shaw, 2018).

While route tables control the rough access on the subnet level, the use of security groups clarifies the logical grouping of applications. Grouping application functionalities via security groups make it easy to understand from a development team's perspective without having to dive into the framework of ip networks. In addition, resources in AWS usually have dynamic ip address assignments, making small, incremental adjustments based on ip addresses difficult. The dynamic address allocation leads to NACLs being used to control access to subnets utilizing route tables, leaving finer grain control to security groups at the application level.

3. Cross project communications.

3.1. Use cases for cross VPC communication.

With applications deployed in separate VPCs, communications between projects are inherently isolated. The only external inbound traffic flow in the example pattern shown in Figure 2 comes in through the ingress subnet. Per this design, a system that needs to communicate with any dependencies or other services would have to send request traffic to the internet to reach access endpoints in their ingress subnet. This mechanism does isolate traffic to just inbound and outbound channels within the project, but there are situations where internal communications are preferred. In an on-premise private cloud design, sensitive data usually do not leave the data center or internal networks to traverse the internet. In addition, there may be compliance and security implications of allowing customer data to enter the internet.

A use case of traffic that does not need to enter the internet to obtain necessary data are business applications that orchestrate business transaction flows. These applications rely on multiple internal centralized business services to provide information so it can form a complete picture before processing the request. Many of these centralized services are provided as internal microservices, which has a recommended pattern of small and discrete components but do not necessarily need to be exposed to the internet (Christudas 2019).

For example, in an order subscription design, shown in Figure 3, a project provides the activation of a subscription for an external customer. The project would rely on information and data kept in other projects, such as customer identity, payment validation, restricted party screening, and other sensitive information that would usually be kept internal. These centralized systems do not have a public interface because they do not directly interact with customers to reduce threat exposure. In this case, a mechanism to access the data via internal network communication within the context of multiple VPCs is needed.

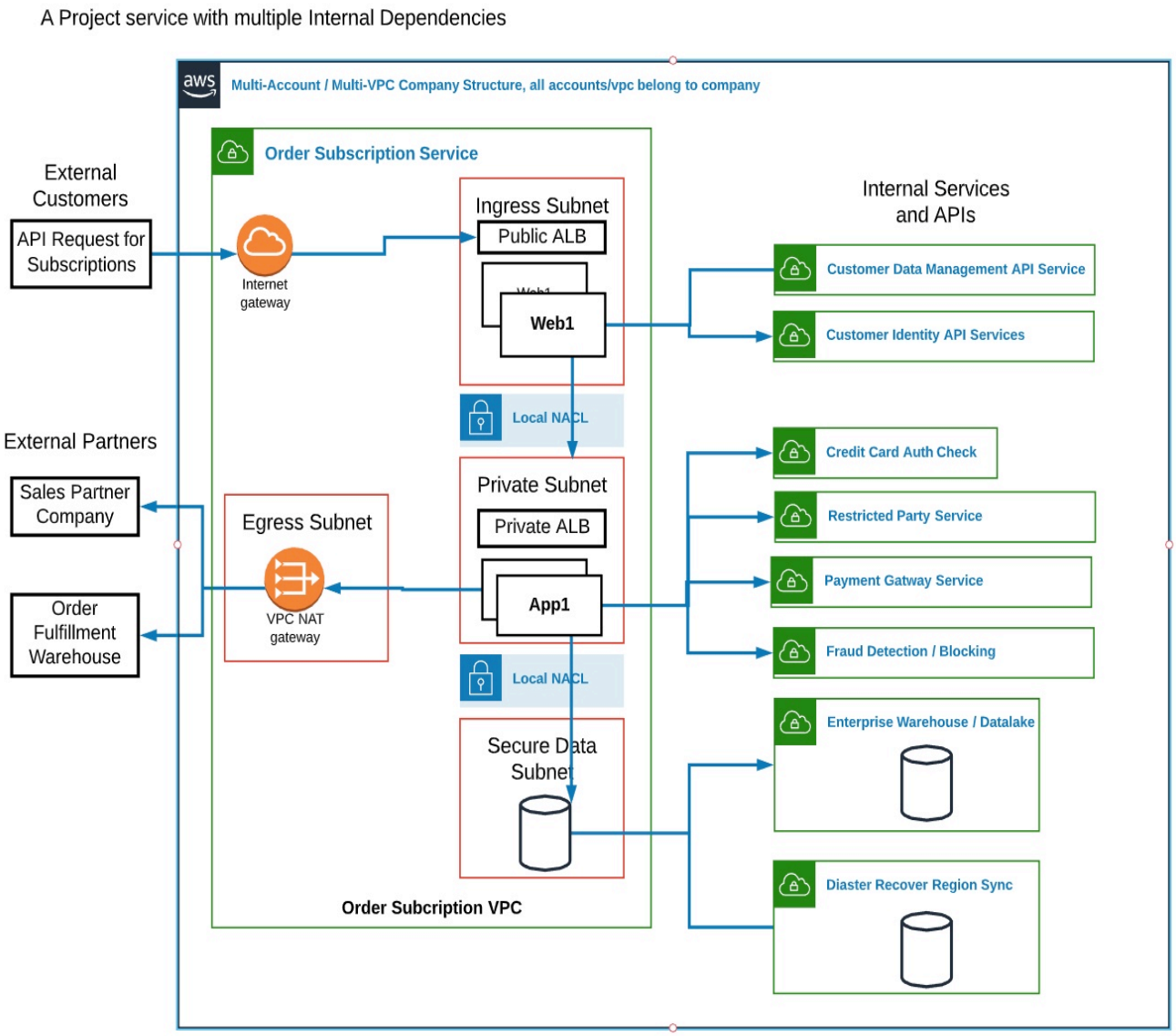

Figure 3: Order Subscription core project, with dependencies on multiple supporting applications.

Another use case pattern involves the use of resource VPCs that support application container designs. Microservices applications provisioned in container clusters, such as Kubernetes, often needs to access internal resources related to the application. These resources, such as data persistence layers, file systems, and other on-premise data sources, are built in separate VPCs to isolate data. This design provides isolation to the Kubernetes cluster so that the cluster only provides business logic services in a standard structure, without the need for specific project data.

Andy Huang, tokidoki2@gmail.com

The data and file resources are typically only used by the microservice in the VPC , and due to the isolation design, they are not built in the shared account. This isolation leads to the need for internal connectivity from the business logic layer provided by the Kubernetes account to the resource accounts. Figure 4 shows that the customer interface resides with the centralized container account; therefore, resource VPCs that provide data support may have no private subnet and also do not need an internet interface. In this case, internal communication channels require access from the customer interface and business logic layer, which resides in the container VPC to the resources located in the resource VPCs.

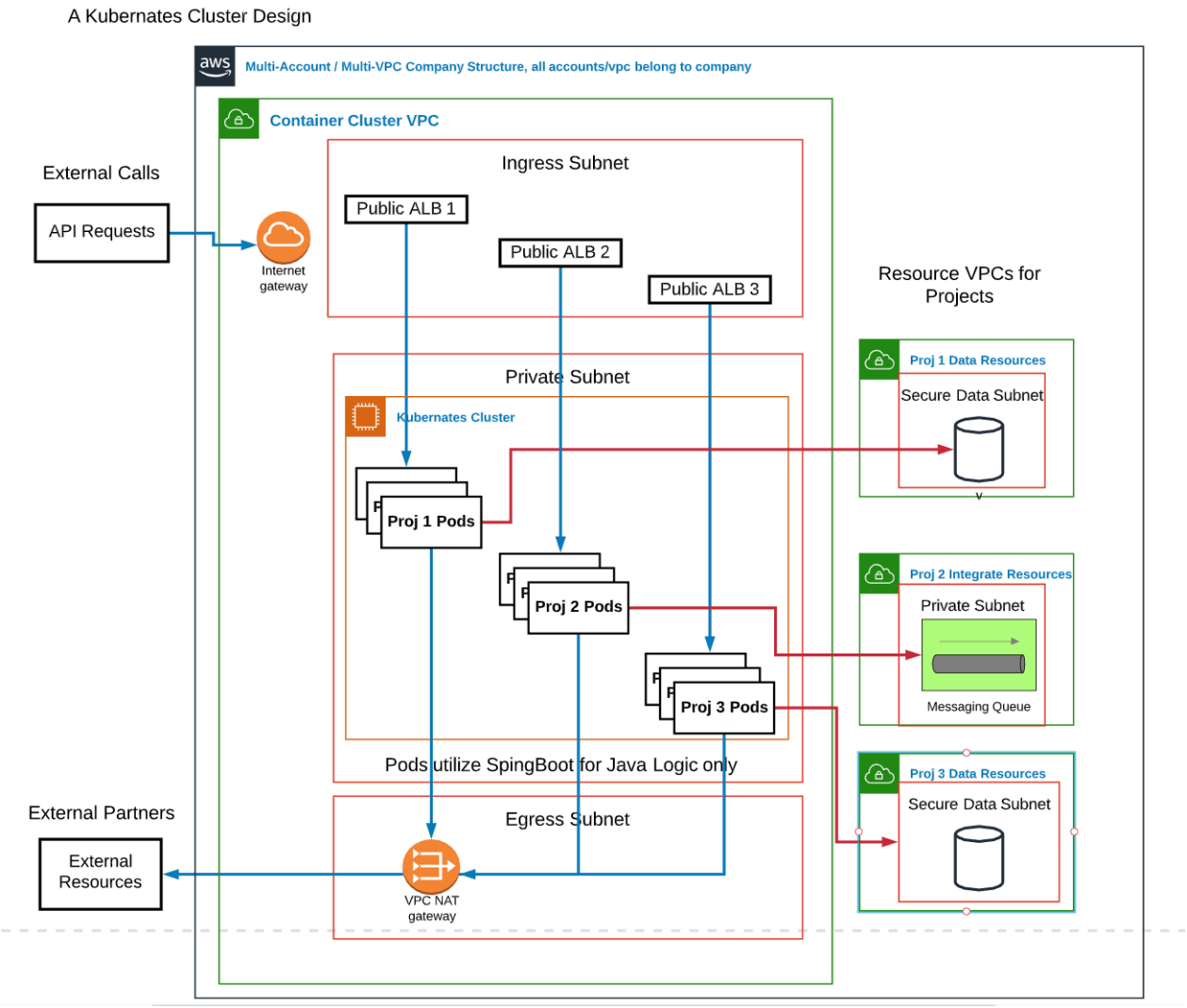

Figure 4: Central Kubernetes cluster with dependencies on resources in separate VPCs.

The third use case for cross VPC communications involves the setup of Disaster Recovery scenarios. For example, AWS VPC structures do not span regions and in order to support failover or active-active use cases, separate VPCs are set up to allow for synchronization of data and processes. These synchronizations between applications and data require data flow between VPCs in order to update resources between two different sites. In Figure 5, this shows a cross VPC communications channel within the same project, but between two different regions. The example shows database traffic that traverse between two regions constantly in order to synchronize between an active database and a passive database.

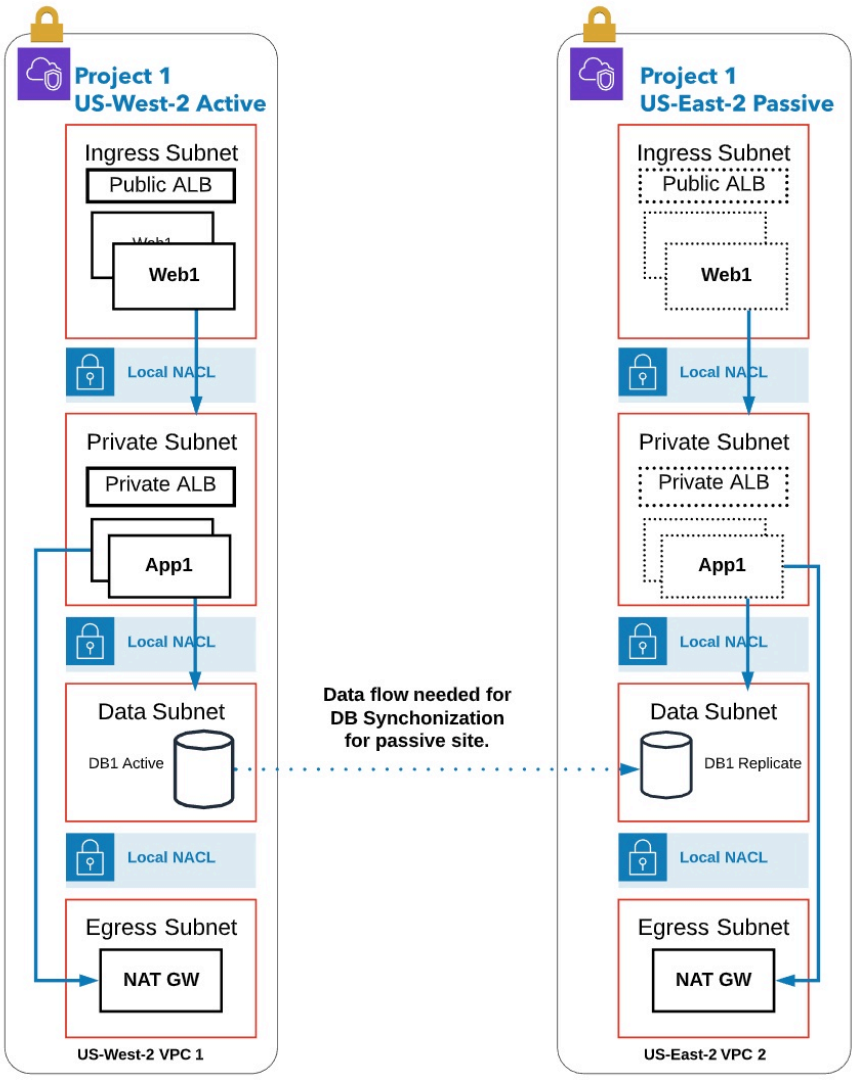

Figure 5: Disaster Recovery structure which needed cross VPC communications.

3.2. Cloud vendor cross VPC patterns

Amazon AWS, Google Cloud and Microsoft Azure have established patterns and techniques for cross VPC communications to address the use cases described. Techniques and tools, such as VPC peering and Privatelink mechanisms, allow for traffic to remain internally within the cloud provider's network. Previously mentioned ingress and egress whitelisting, which allows for one project to communicate with another project is also an established pattern, but this pattern requires traffic to exit the VPC into the internet before rerouting back into the cloud network. There are multiple challenges with the ingress and egress pattern; once traffic enters the internet, the routers on the internet dictate the path that the traffic takes. This uncertainty can significantly impact latency times, as well as expose the traffic to attackers and other actors. Also, there may be compliance reasons to restrict data flowing into the internet between two identified internal projects.

3.3. VPC Peering

Peering connections such as AWS VPC Peering, or similarly called Google Cloud VPC Network Peering and Azure Virtual Network Peering, allow an internal relationship between all subnets between two VPCs. Commonalities on the peering space allow traffic to stay on the provider cloud's network entirely, and no traffic exits into the internet ("AWS VPCPeer," 2019). This implementation creates a multi-mesh pattern where all subnets from one VPC are visible and connected to all subnets of the other VPC. The subnets in both VPCs have access to each other. Additional security controls are put in place to limit and control access between resources within the subnets. These controls include route table adjustments, security group adjustments, and other access controls. By default, even though VPC peering allows the ability for subnets to communicate to each other, network access controls and security groups introduce additional granular layers that prevents resources from communicating with each other solely due to setting up VPC peering ("AWS PeerAccess," 2019).

The strength in a peering pattern is the ability to allow for multiple resources in both VPCs, under different subnets, to communicate with each other. If resources within different subnets in one VPC need to communicate with one or more resources in one or more subnets on the second VPC, peering and the associate controls allows for this. To

guard against free movement of traffic across multiple VPCs, the peering connection is not transitive between multiple VPCs. There is always a direct one to one relationship between two VPCs. The connections between one pair of VPCs is not carried to additional VPCs even if each VPCs have additional relationships with other VPCs. Additional granular access controls restrict subnet access, one subnet does not have arbitrary access to another subnet in another VPC without explicitly granting of access, even when they are VPC peered.

VPC Peering setup is relatively easy, and the process usually restricts and calls out two different distinct accounts and VPCs. However, post peering access control set up can be challenging and error-prone, since this setup depends highly on the understanding of application communications, as well as network patterns for access ("AWS PeerAccess," 2019). Therefore, configurations implemented after the initial establishment of peering need to be done correctly to avoid any overexposure of permissions.

Figure 6 explains the access between two VPCs in a peered relationship. Application one (App 1) needs access to application two (App 2) and database two (DB 2). To control access, the route table for the private subnet of application one needs to allow outbound traffic, while the route table for application two and database two need to grant inbound access. In addition, the security groups for both application two and database two have to explicitly grant inbound access from application one. A reverse flow indicated with the dotted line from application two accessing database one shows a similar set up. In this case application two route table needs to allow outbound traffic from the private subnet ip address space, while the route table and security group for database grants inbound access. The post configuration steps need to be executed with care.

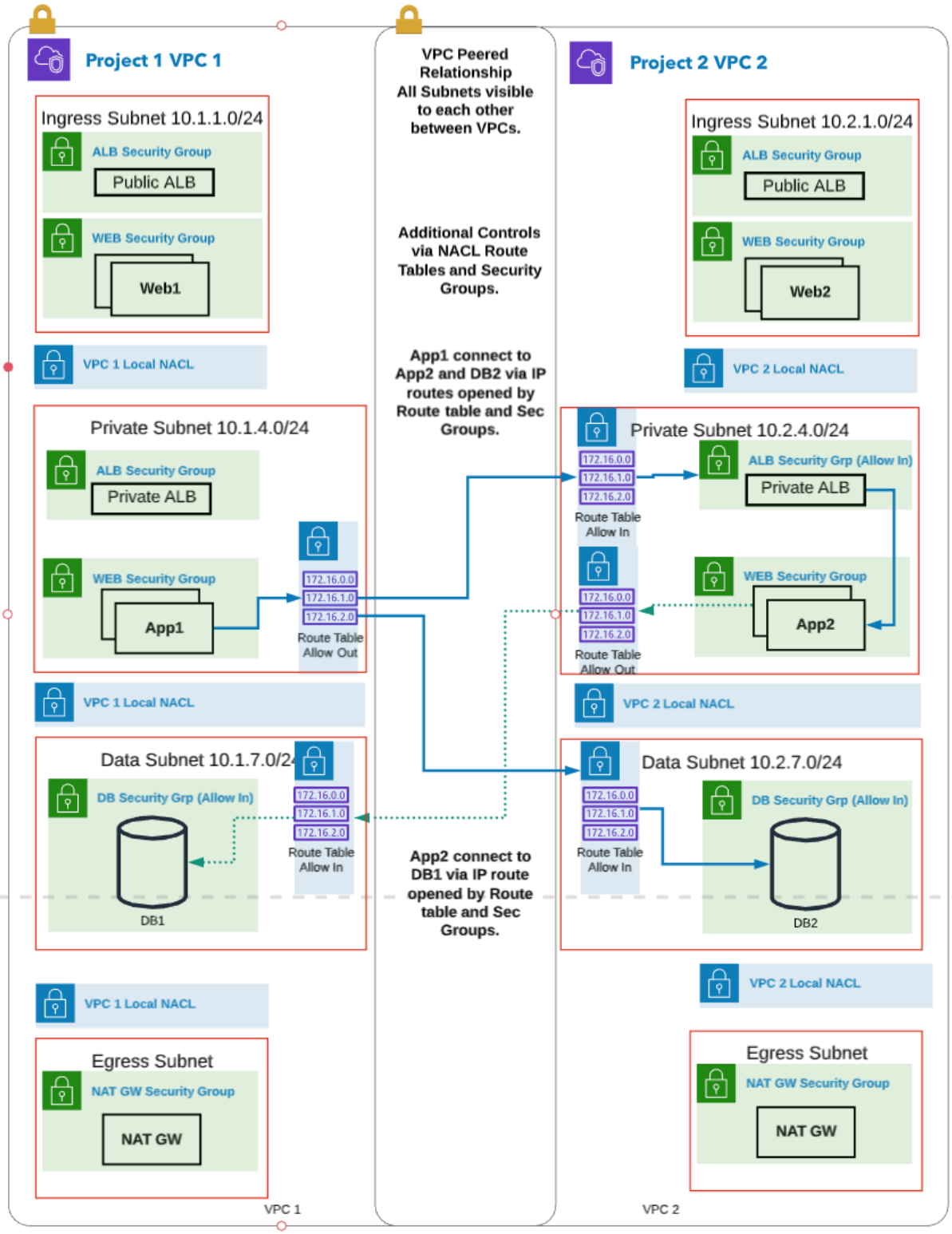

Figure 6: VPC Peering model showing distinct access control between subnets using route tables and security groups.

Andy Huang, tokidoki2@gmail.com

3.4. VPC Privatelinks

VPC Peering provides bi-directional access and can open broad access across two different VPCs for use cases that require such access. For more restricted access, such as unidirectional access or precise endpoint access, a more restricted mechanism is needed. Amazon offers this pattern as AWS PrivateLink while Google cloud offers this as GCP Private cloud access. Microsoft Azure has named this as Azure Privatelink. All three patterns offer very similar functionalities, where traffic does not enter the internet, and access is more restricted than peering patterns.

Privatelink patterns were initially scoped to allow for third-party commercial product vendors to offer services within the cloud providers. This is so that the third-party vendors do not need to utilize peering, which can expose their accounts and entire subnets to the customers. In addition, VPC peering relies heavily on non-overlapping ip address ranges, which is not guaranteed between cloud accounts ("AWS CIDR," 2019). Cloud providers themselves have also exposed native cloud services to customers, providing distinct endpoints so that traffic natively traverses the internal network. An example of this is the exposure of AWS S3 bucket access. Previously, all access would have the traffic enter the internet, however with AWS PrivateLink S3 endpoint access, traffic accessing S3 buckets remains within the AWS network ("AWS VPCE", 2019).

As with VPC peering, traffic between endpoints is contained within the cloud provider's network layer. Privatelink allows a resource to expose an endpoint service for clients to connect to; this endpoint service can be exposed to certain accounts, or to all accounts. Clients establish and creates an endpoint within the client's own VPC, this endpoint has a direct connection to the endpoint service in the other VPC providing the interface. This has additional security implications in that there is only one channel and path outbound from the client to the target endpoint. The pattern is much more restricted then VPC peering since an explicit endpoint is opened for connection between two resources ("AWS PL," 2019). The Privatelink pattern also allows the formation of multiple fan-in to single connection endpoint pattern, where multiple resources from multiple clients can access a single endpoint on the VPC providing the service.

In Figure 7, application one utilizes two dedicated established Privatelink channel to access application two and database two. Application two also utilizes the dedicated channel to access database one.

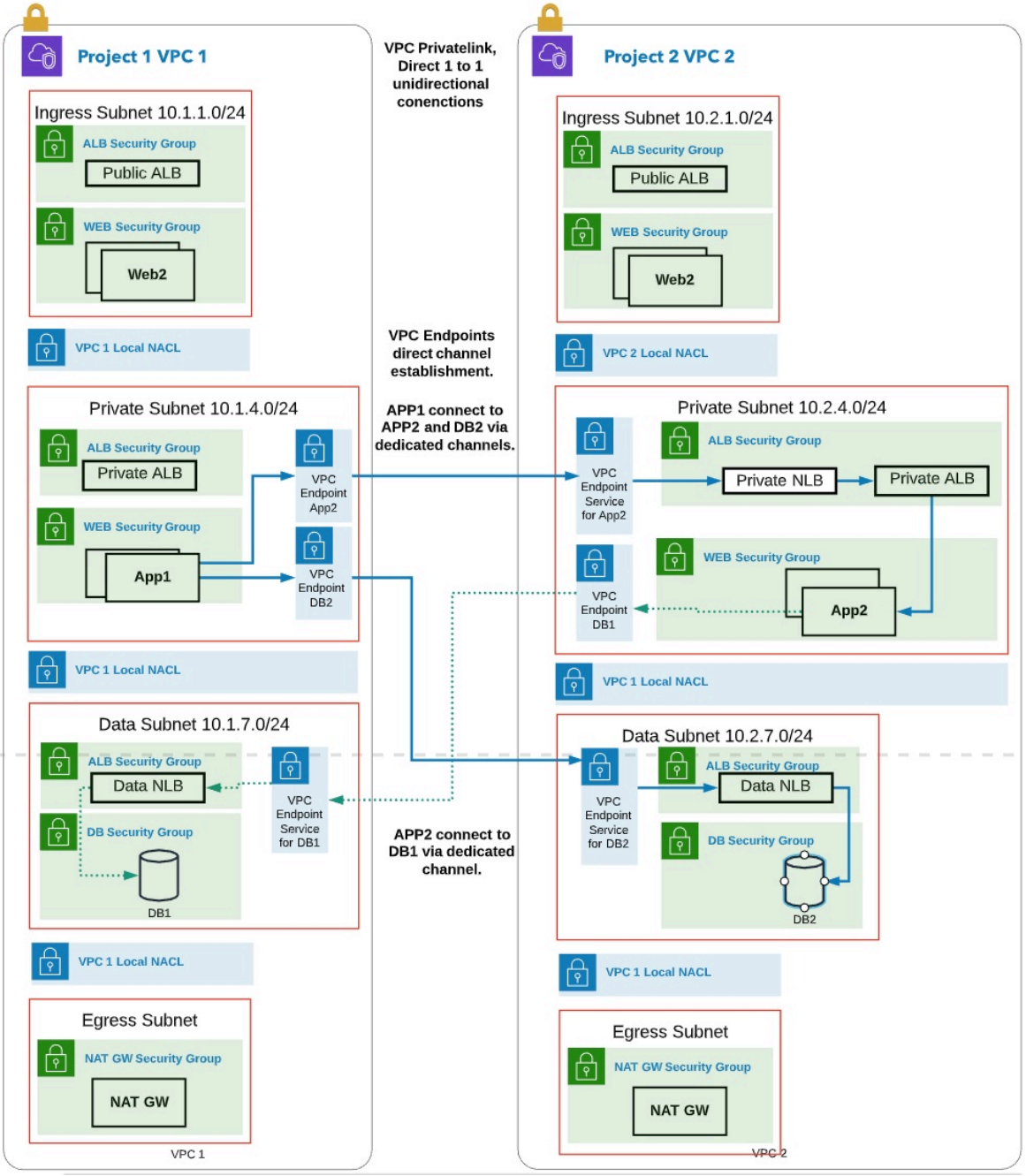

Figure 7: AWS VPC PrivateLink connections between App1 and App2 and DB 2, and App2 to DB1

There are drawbacks to Privatelink patterns. This mechanism is not cross-region capable, and there are usually no other options for internal traffic between regions other than integrating with VPC peering ("AWS PL interregion," 2019). The restrictive nature of Privatelink can also create multiple provider endpoints for any particular VPC if the project needs to expose multiple endpoints, leading to multiple connection setups. However, teams usually opt for Privatelink mechanisms when available to avoid the potential for overexposure that can happen with VPC peering.

Unlike VPC Peering, there are no post configuration items for a Privatelink. When a link has been set up, a connection is established between an endpoint on the client side and an endpoint service on the provider side. While Privatelink is more restrictive and allows for tighter integration controls, there are areas where set misconfigurations can happen in the setup stages along with additional configurations. These errors can lead to unintended over exposure of access for the VPC endpoint service.

3.5. Additional patterns for cross VPC

The third traffic pattern for cross VPC traffic is the traditional services call through the internet. Each application calls another service by sending traffic outbound to the internet-facing endpoint on the other service. For internal applications communicating with each other, this is less secure since teams and the cloud provider lose control of the part of the path of the traffic. The pattern is very similar to a proxy call, with source systems whitelisting the target endpoints, and the target systems whitelisting source systems. The challenges introduced by this form of access is beyond the scope of the paper since cross project traffic exit into the internet, for compliance and performance reasons, this is also not an optimal access pattern.

A new pattern and service offered by AWS utilize the AWS transit gateway service. This pattern enables a centralized area between VPCs, allowing for multi-VPC to multi-VPC control and access. This pattern requires a centralized security or network team to help set up and enable traffic rules. While this is advantageous from a configuration standpoint since it provides overall visibility between connections, a central team is heavily relied upon to ensure to connect the two separate VPCs. This overhead

also removes the flexibility between two teams coordinating services access to each other.

3.6. Authentication, Authorization and Accounting (AAA)

All patterns of VPC connectivity allow for some level of authentication and authorization during setup, but they do not extend this into the application level. Applications must still provide authentication and authorization. Initial setup tracks access permission from the account level and VPC level. Post configuration steps are needed to adjust route tables and security groups to tighten access to the subnets or application. This do not necessarily implement application authentication and further application authentication and authorization must be implemented. Privatelink setups can constraint connections to single endpoints when implemented, but this is on the network flow level, and does not address application authentication or authorization.

4. Complications

4.1. Unintended lateral movement.

Since additional configuration needs to be in place for VPC Peering and Privatelink mechanisms, configuration mistakes can lead to unintended access. Inadvertent misconfigurations are common and not explicitly warned by cloud providers. Different teams have different processes and procedures for implementation. For example, the application teams may be more concerned with application connectivity rather than network security and may unintentionally misconfigure the connectivity.

Figure 8 shows the standard VPC peering connection between two VPCs. They can be from different accounts which serves different projects or from the same account and the same project, but different application within the project. The configuration screenshot shows the entire VPC CIDR ranges opened between the two VPCs as part of initial configuration. Additional security controls in VPC peering is introduced by using network access control and application access controls. An error or misconfiguration in the follow-up implementation in the two areas create vulnerabilities.

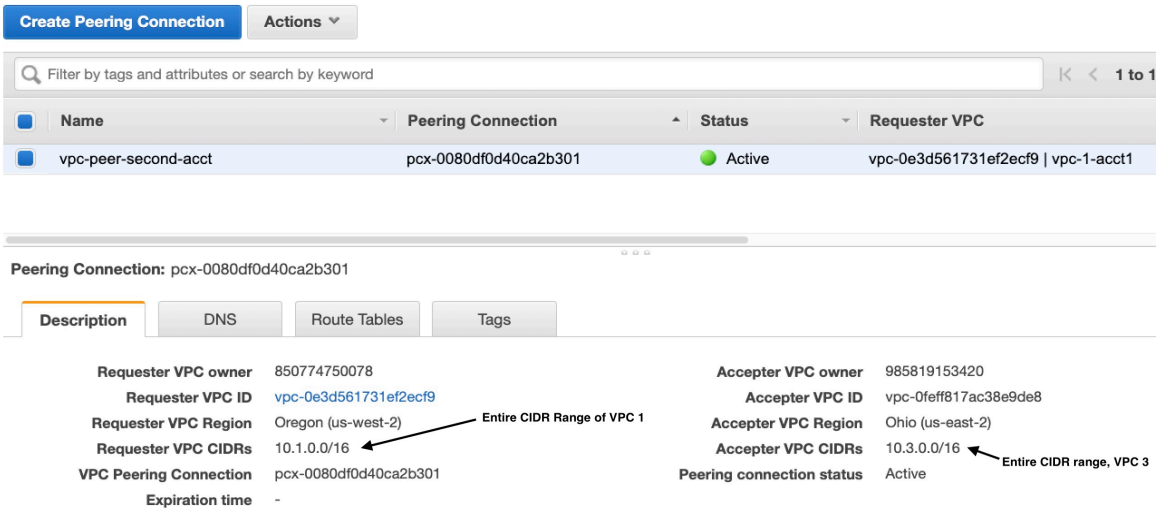

Figure 8: AWS VPC Peering console connection set up, note entire CIDR range is used in peering setup.

The application teams adjust network route tables and security groups in follow up configurations to allow for subnet to subnet communications. Since VPC peering relationships are established between two application teams, the level and degree of security understanding may be different between the two teams. The second team that configures application access for inbound communications may be unaware of the consequences and may choose to open up the entire subnet range, rather than defining proper subnet ranges for specific use cases for the application. In figure 9, the project team for application one has implemented an outbound route table that requested access to the entire subnet space of application three in VPC-3. A proper subnet destination target should be either 10.3.4.0/24, which would represent the private subnet of VPC-3 or 10.3.7.0/24, which would represent the secure subnet of VPC-3. The current subnet request represents all subnets and seem to request broader access than needed. This opens up a larger outbound range than what is needed for application one.

Figure 10 shows project team one has also configured access from the ingress subnet to the entire subnet range of application three. Ingress subnets usually have an internet-facing endpoint and should be much more restricted, it would be unusual to grant access from an internet facing subnet for one VPC to all subnets in another VPC.

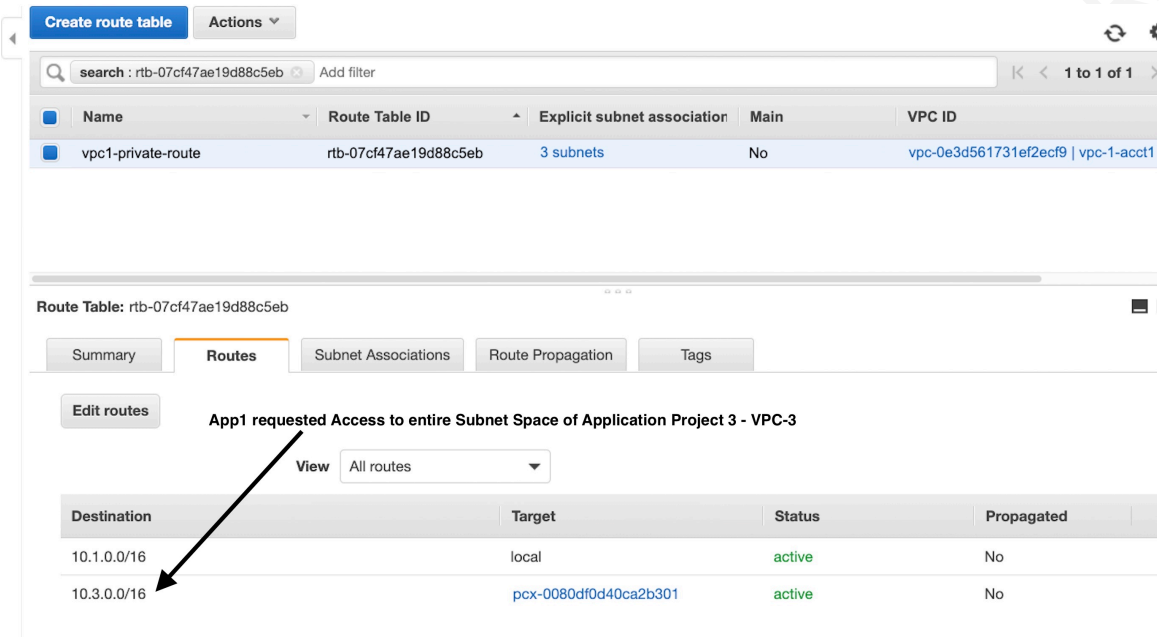

Figure 9: Does application one really need access to all the subnets in application three?

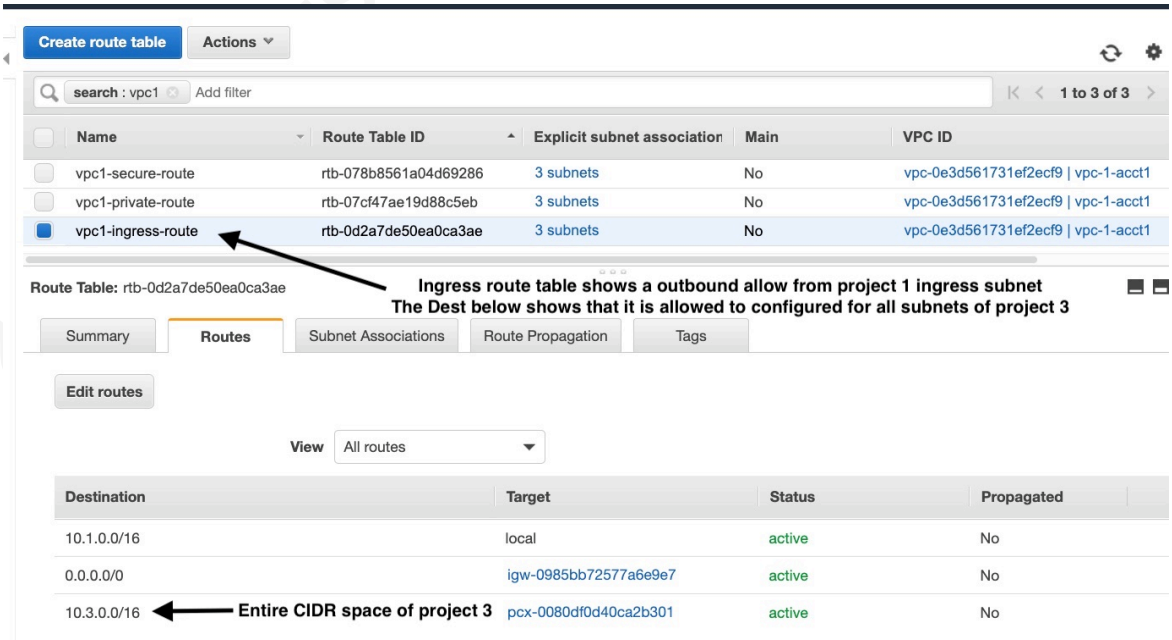

Figure 10: Ingress subnet from application one allowed outbound access to all subnets on application 3.

Additional configurations of security groups on the target project and application grant access to the resources in the subnet. In this case, the application team for vpc-3, incorrectly added route information for the secure subnet. Due to overly permissive configurations, this allowed resources in ingress subnet of application one in VPC-1 to

gain access to the resources in the secure subnet of application three in VPC-3. The following figures show the configurations in place, with no warnings from the cloud vendor. In addition, the configurations shown is allow all subnets of application one to access the secure subnet of application three, and is not restricting inbound access to the resources in the private subnet of application one.

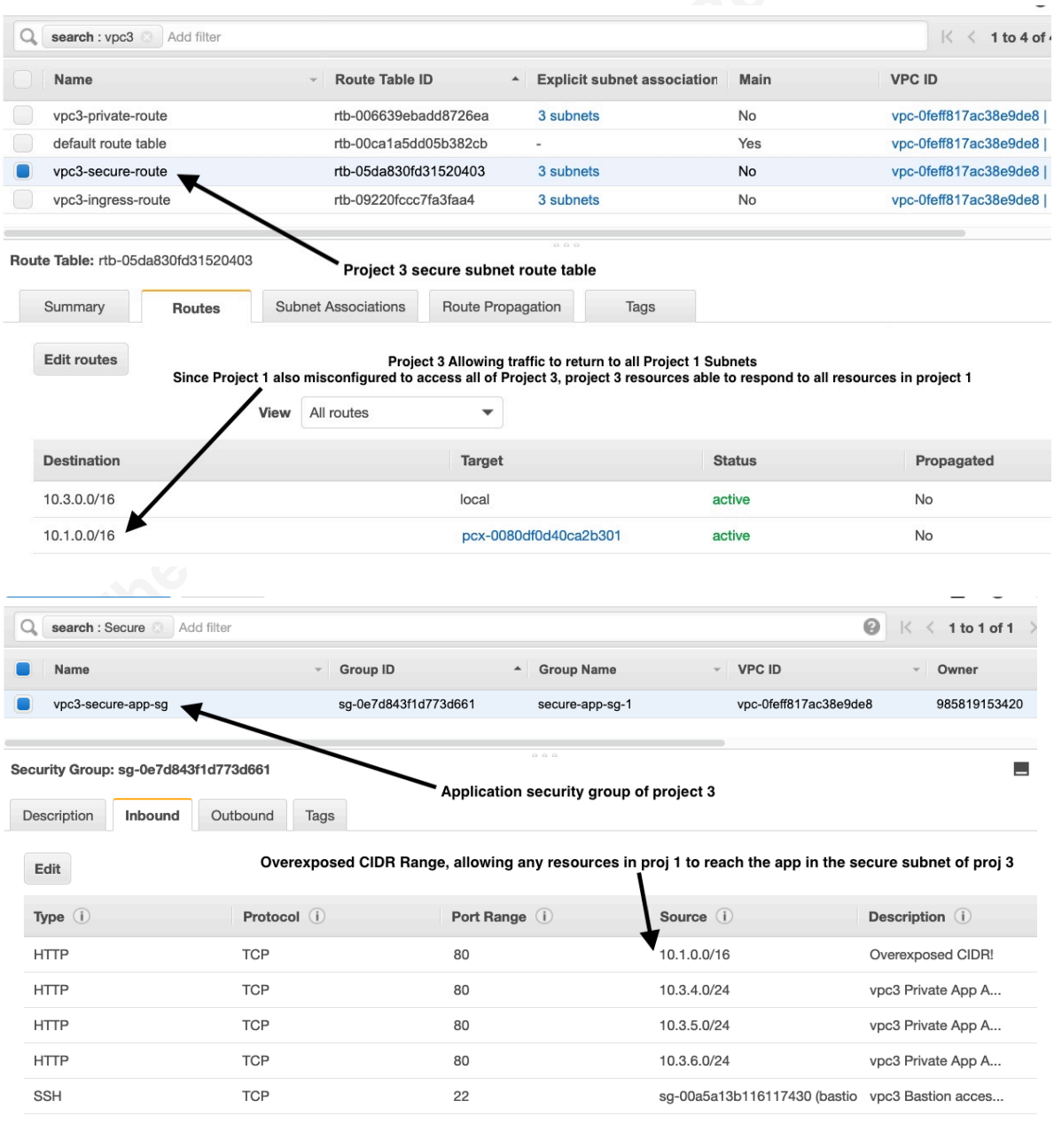

Figure 11,12: Route table for application three allowed to respond to ingress subnet from application one VPC.

A curl command is executed from a system in the ingress network from VPC-1, where the IPs are in the 10.1.1.x range, which shows that it is able to gain access to the secure app on vpc-3, in the 10.3.7.x range. This is not the desired outcome as applications in the ingress subnet may have external internet facing interfaces, and the secure application in VPC-3 is now exposed to a system that has internet access.

```
[ec2-user@ip-10-1-1-23 ~]$ curl -v http://10.3.7.107/Hello.txt
*       Trying 10.3.7.107…
* TCP_NODELAY set
* Connected to 10.3.7.107 (10.3.7.107) port 80 (#0)
> GET /Hello.txt HTTP/1.1
> Host: 10.3.7.107
> User-Agent: curl/7.61.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/2.7.16

< Date: Mon, 14 Oct 2019 21:50:41 GMT
< Content-type: text/plain
< Content-Length: 28
< Last-Modified: Mon, 14 Oct 2019 21:36:49 GMT
<
Hello from VPC-3 Secure App
* Closing connection 0
[ec2-user@ip-10-1-1-23 ~] $
```

This exercise demonstrates the ability for over-permissive connections to allow movement from unintended sources into secure areas between two VPCs. In this case, both VPCs belong to two different accounts and to teams in two different regions. Both teams may be unaware of the other team's structure. The team for application three may believe that 10.1.0.0/16 is the correct subnet to allow access to if application team one gave them the information. In this example, the subnet structure is oversimplified to show the correlation and the teams are likely to raise a red flag upon seeing the much broader allowed CIDR range. However, in practice, teams usually do not ask about the internal structure of each other's application implementations, and smaller subnets permission access can easily slip through approvals and be allowed.

In order to prevent this type of lateral movement, it is critical that both teams clarify requirements and meet to show the structure of the applications during VPC peering setup and subsequent traffic enablement. AWS in this context would not be able to raise alerts since it cannot determine the subnet relationships and the policies of the clients. AWS would be unaware of security policies or practices to avoid direct connection

access between ingress subnets and secure subnets. Network access via route tables between subnets for VPCs and security groups for application access needs to be correctly defined and controlled. Templates for setting up VPC peering should be considered to prevent overexposure of the subnets.

The Privatelink patterns offer narrower access pathways then VPC peering, but the connections are also susceptible to misconfiguration that may overexpose permissions. During setup utilizing endpoint services and endpoints, it is critical to define to which account the current service should be allowed to advertise towards. The team providing the service must pay special attention to granting access to specific project or application requests. The whitelisting mechanism to whitelist proper principles needs to be set up properly to prevent overexposing the service. Unless a commercial vendor that is providing a service, a "*" global exposure should rarely be used since it opens up broad exposure and advertises this service to all possible clients. Figure 13 shows the "*" that now advertise this endpoint service to all possible clients.

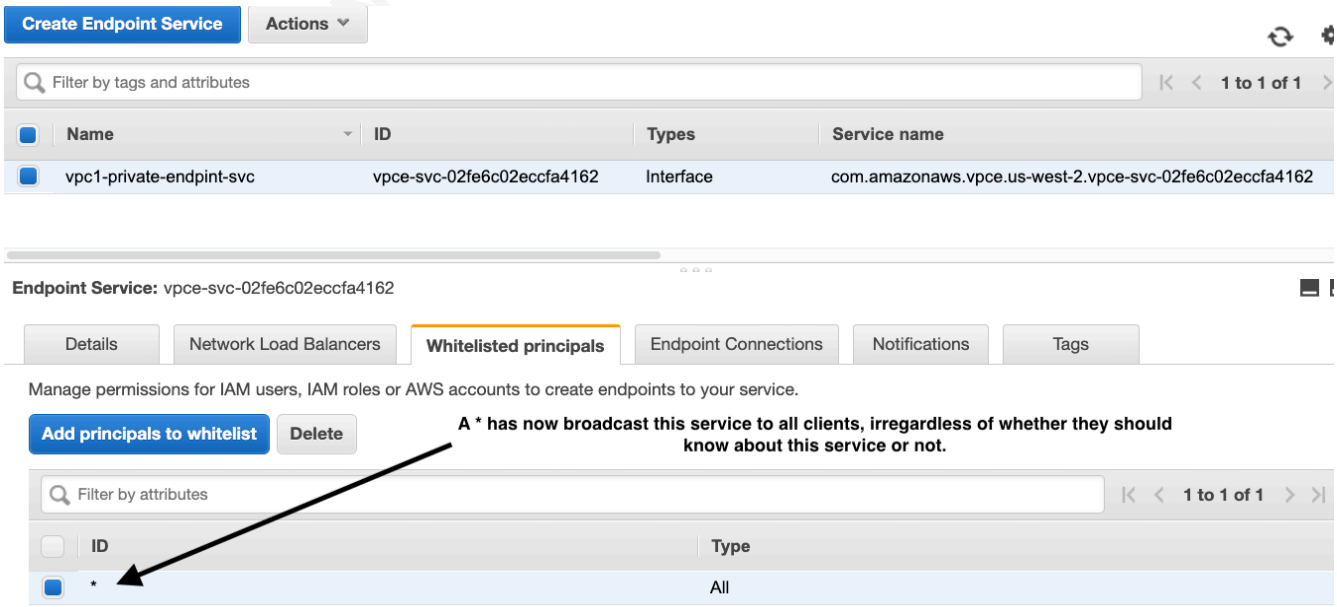

Figure 13: A * is seldom used for project to project communications.

For a Privatelink setup, there is no additional scoping and configuration on the service provider side once the provider accepts the request, and there is no indication on which project generated the request. This result makes the audit and tracing more complicated. The following screenshot demonstrates that there is one request that has

already established a connection from a VPC endpoint. The accepted request is a legitimate connection and originated from VPC-2 private subnet to connect to VPC-1 private subnet resources. However, if a new team is looking at this account for connection status, or trying to trace some information, the endpoint ID information presented here means that the team needs to trace via other mechanisms to determine where this connection had originated.

The first Endpoint ID in Figure 14 shows a new connection request, but in this case from the ingress subnet. This is an unintentional request made by the first application team. However, from the provider side, it is difficult to tell which subnet the requestor id originates from. A misconfiguration on the requestor side, coupled by the acceptance of the provider side, can lead to an unexpected exposure of the provider to the wrong client subnet. It is important that the provider team confirms with the requestor team before accepting the request.

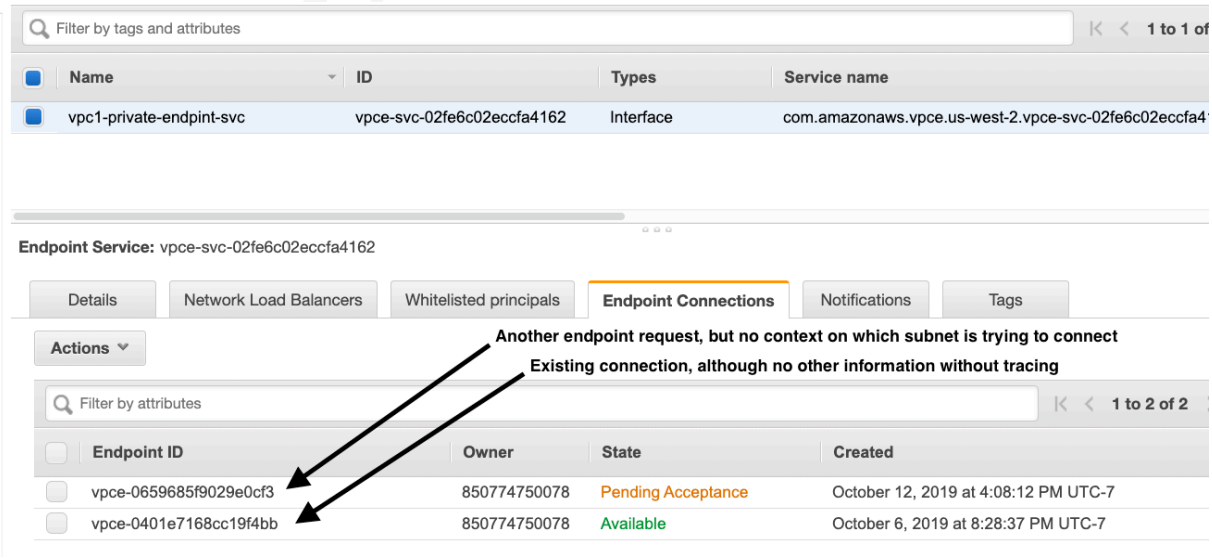

Figure 14: Requests are shown as endpoint IDs, and from unknown subnets or situations.

An AWS PrivateLink VPC endpoint service is exposed behind a Network Load Balancer (NLB) on the provider VPC, therefore the security group configuration is configured to trust the NLB. Once a Privatelink is set up, the provider application will explicitly trust the requestor. A provider does not necessarily know which subnets are coming from the requestor project, it is up to both teams to communicate closely to avoid a misinterpretation of the setup. Figure 15 shows a connection from application two from

the VPC-2 standpoint, where a connection in VPC-2 has requested a connection to VPC-1 provider service endpoint. The name of the connection is added here to indicate that this connection actually comes from the ingress subnet from the requestor. If the name is not added, the provider account would not know which subnet is truly calling the VPC endpoint service.

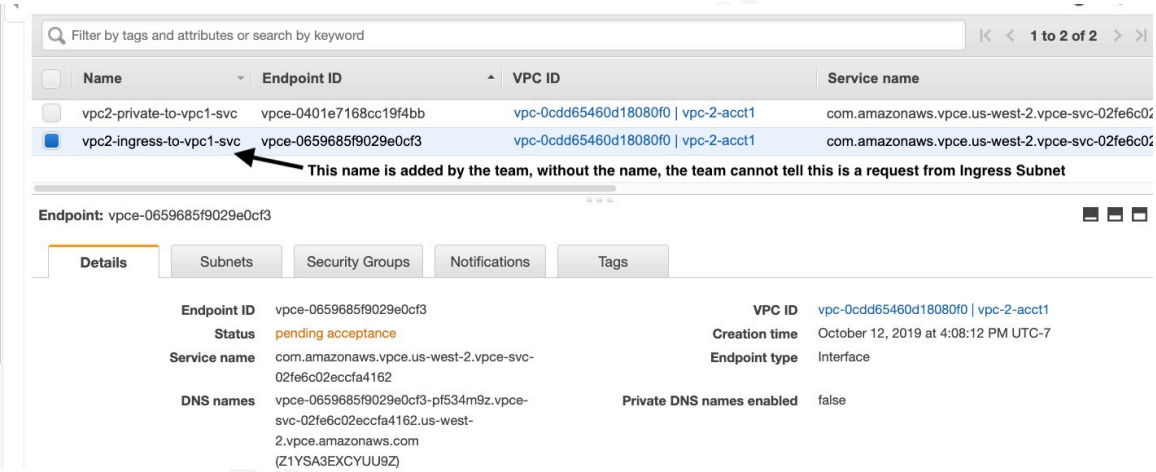

Figure 15: An AWS PrivateLink request from the ingress subnet.

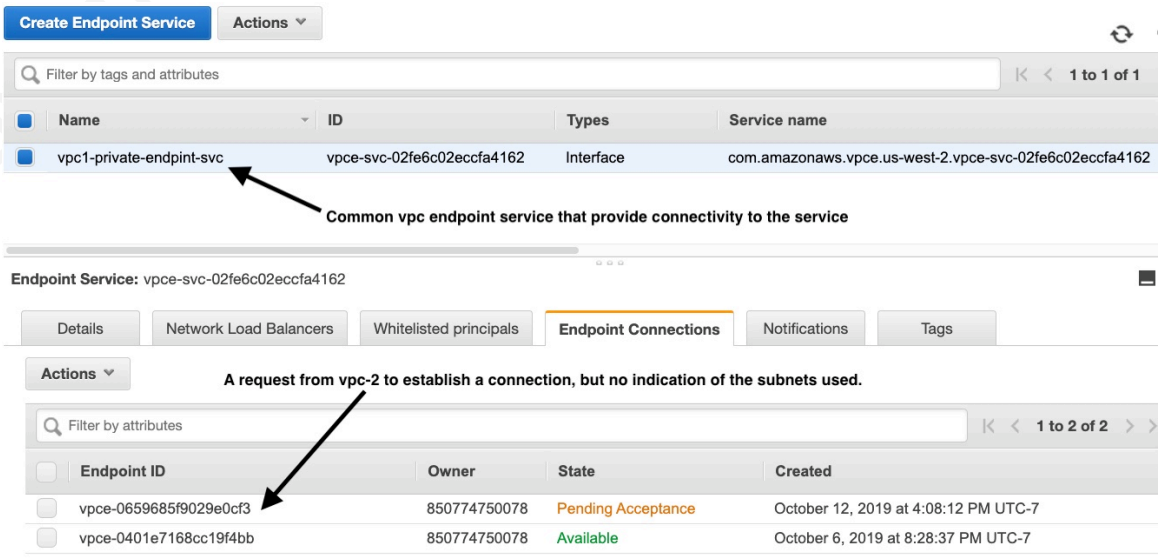

Figure 16: From the provider standpoint, the same ID just shows up as another endpoint ID requesting access.

When project team one accepts the Privatelink request in VPC-1, a system in the VPC-2 ingress subnet will now be able to access resources in the private subnet of VPC-1. This connection is likely unintended, but from project team one's perspective, they did not know which subnet is connected inward from project two. The following screenshots show the before acceptance and after acceptance of the AWS PrivateLink request. Before the connection is established, the VPC endpoint name is not resolvable.

```
[ec2-user@ip-10-2-1-19 ~] $ curl -v vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-
west-2.vpce.amazonaws.com
* rebuilt UURL to: vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-west-
2.vpce.amazonaws.com/
* Could not resolve host: vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-west-
2.vpce.amazonaws.com
* Closing connection 0
Curl: (6) Could not resolve host: vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-
west-2.vpce.amazonaws.com
[ec2-user@ip-10-2-1-19 ~]
```

After the acceptance, the VPC endpoint is resolvable, and in this case, the curl command is executed from the ingress subnet of VPC-2.

```
[ec2-user@ip-10-2-1-19 ~] $ curl -v vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-
west-2.vpce.amazonaws.com
* Trying 10.2.1.186
* TCP_NODELAY set
* Connected to vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-west-
2.vpce.amazonaws.com (10.2.1.186) port 80 (#0)
> GET /hello.txt HTTP/1.1
> Host: vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-west-2.vpce.amazonaws.com
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 14 Oct 2019 22:54:56 GMT
< Content-Length: 43
< Connection: keep-alive
< Server: SimpleHTTP/0.6 Python/2.7.16
< Last-Modified: Mon, 14 Oct 2019 22:20:49 GMT
<
HELLO from VPC-1 Private Application Stack
* Connection #0 to host vpce-0659685f9029e0cf3-pf534m9z.vpce-svc-02fe6c02eccfa4162.us-west-
2.vpce.amazonaws.com left intact
[ec2-user@ip-10-2-1-19 ~]
```

The connection is initiated from host 10.2.1.19 to an endpoint in the same subnet, 10.2.1.186. The essence of the AWS PrivateLink is that the requestor establishes a connection to an endpoint within its own subnet, and this endpoint has a connection to the provider subnet. This behavior masks unintended connection between resources in application one ingress subnet and the target resources in application three private subnet.

4.2. Detection

It is challenging to detect connections after the set up for VPC Peering or Privatelink mechanisms. Automated mapping tools can help in determining connections within a VPC, but the tools lack the ability to read into VPC Peering and Privatelink connection setups to show cross-communication scenarios. Using the tool CloudMapper from DUO networks as an example, the tool can map to the edge of one particular VPC, but only to AWS native services such as S3 and this seem to be only in regard to an endpoint that may exist (“CloudMapper,” 2019)

The VPC structures used in the tests created the following map from CloudMapper in Figure 17. It shows where VPC-1 and VPC-2 resides but the VPC peering connection between VPC-1 and VPC-3 is not detected, and outbound connection scenarios between the two VPCs are not shown. The successful tcp connection test which shows incorrect access from a system in the ingress subnet (10.1.1.23 VPC-1 ingress) to the secure subnet (10.3.7.107 VPC-3 secure) is hidden from this audit.

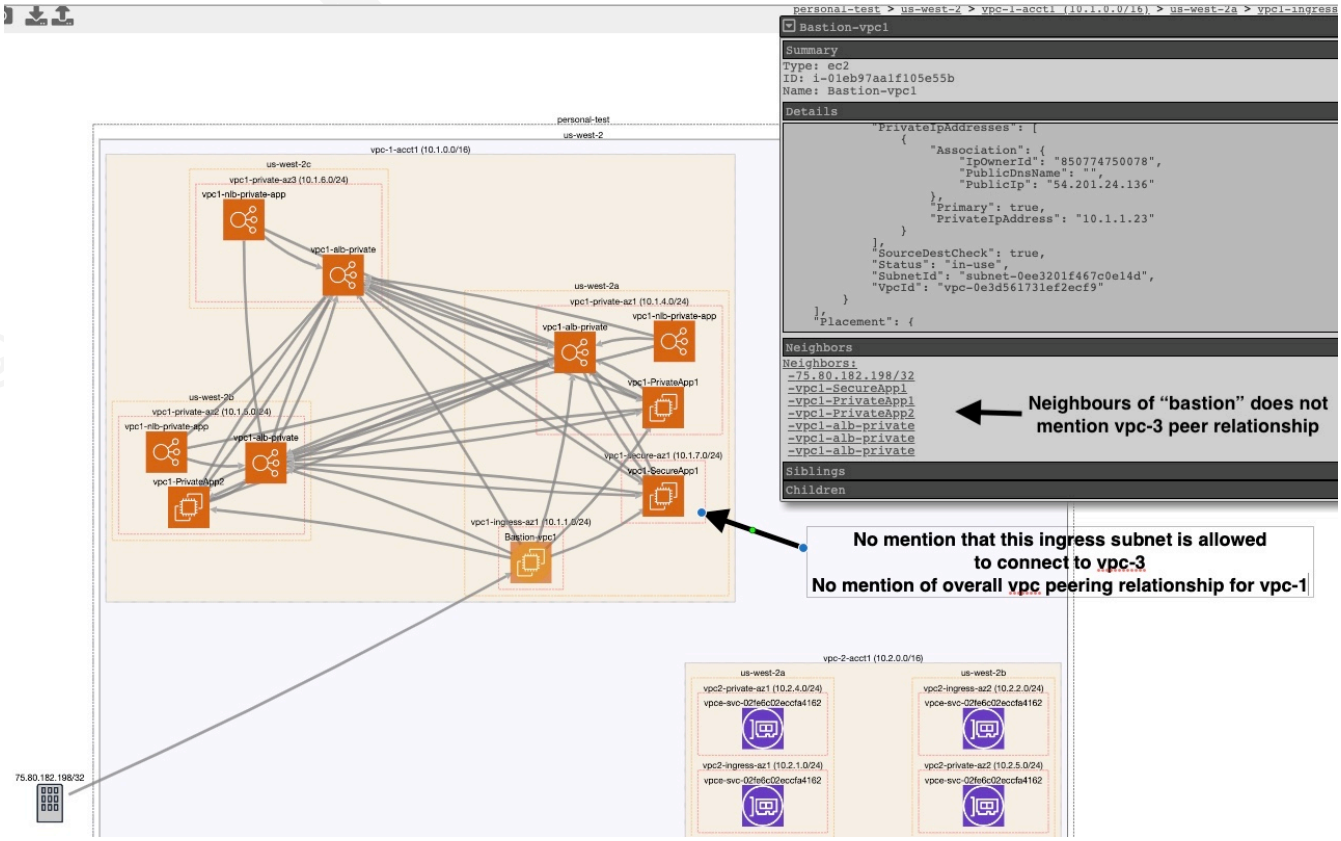

Figure 17: Cloud Mapper visualization, but lacks insight into VPC Peering between VPCs

The AWS PrivateLink relationship between VPC-1 and VPC-2 within the same account, is not detected as well. The network map does indicate VPC endpoints because these endpoints are created inside the requestor subnet. The entries can be used to indicate a connection to a VPC endpoint service; however, it is unclear which endpoint service this is connected to based on the diagram in Figure 18. This can be used as an initial vector to start from the requestor perspective to start tracing a connection to determine the provider. The diagram does not show any accepted connections if this is a provider service. From a detection perspective, the VPC endpoints are displayed, and the diagram shows a set of suspicious endpoint connections in the ingress 10.2.1.0/24 subnet. This can trigger further actions since it is unusual to have VPC endpoints set up in the ingress subnet.

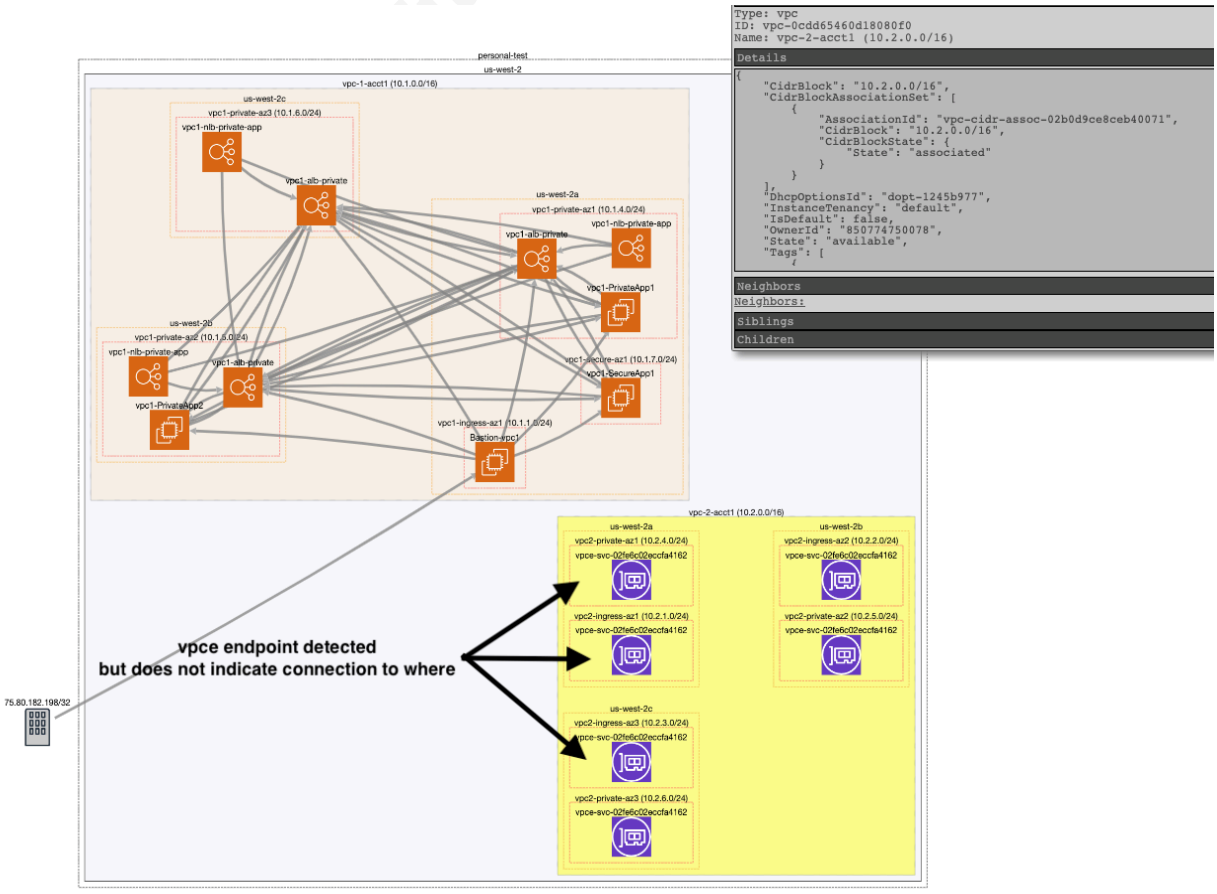

Figure 18: Cloud Mapper visualization, shows VPC endpoints, but not relationship to where they connect to.

The diagram becomes clear when traffic is initiated from bastion host in the ingress subnet. The following diagram shows connectivity between bastion host in the ingress to the VPC endpoints. Hopefully, this can tip off the team to show that this may be an unnecessary connection and may potentially be dangerous in allowing outside connections into bastion and then into a VPC endpoint that leads to another VPC.

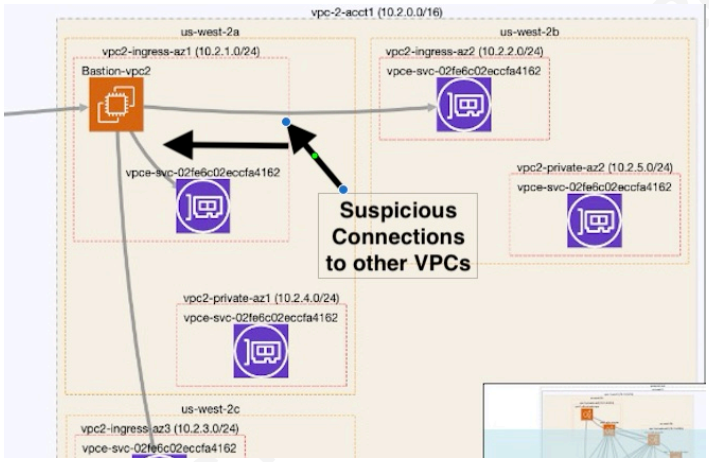

Figure 19: Cloud Mapper visualization, suspicious VPC connection from the bastion, which is usually locked down.

5. Audit

5.1. Approach

To truly determine the connection status and connections between subnets using VPC Peering or Privatelinks, these connections need to be traced using AWS command-line interface or the AWS API set ("AWS CLI," 2019). The toolset can gather data from both VPCs to form the complete picture. The data can then be consolidated to provide information regarding connectivity and a determination can be made as to whether the connections are intentional or warrant further investigation. Connections between VPCs can lie dormant and remain in place for an extended amount of time since it is a configuration item that is set up and left alone as long as the VPCs exist. Over time, the number of connections can build up and become stale as retired connections are left unused as dependencies retire.

5.2. Tracing through a VPC-peered connection

The AWS command set is used from one account to determine where connections exist, and this can be documented and periodically compared. The data can be stored in a secure repository so that there can be continuous audits over time to compare current connections. Automated diagramming tools such as Cloudmapper do not expose VPC peering information. The tool show Privatelink endpoints in between VPCs from the requestor side only. Therefore, the tools cannot reliably be used to determine these relationships between VPCs.

In the example setup to use AWS command line interface, a route has been implemented that allows for the VPC-1 private subnet to route data to the VPC-3 subnets over a peering connection. The AWS commands set can expose and categorize this connection. Several commands are executed from the VPC that has requested the peering connection, in this case, VPC-1. The describe-vpc-peering connection command, along with filtering logic, will capture peering related information related to all other VPCs that has requested a peering connection.

```
    aws ec2 describe-vpc-peering-connections --filter Name=status-code,Values=active
Name=requester-vpc-info.vpc-id,Values=vpc-0e3d561731ef2ecf9 | jq
.VpcPeeringConnections[].AccepterVpcInfo
  {
    "CidrBlock": "10.4.0.0/16",
    "CidrBlockSet": [
      {
        "CidrBlock": "10.4.0.0/16"
      }
    ],
    "OwnerId": "985819153420",
    "VpcId": "vpc-050d43a4e610b4f9f",
    "Region": "us-east-2"
  }
  {
    "CidrBlock": "10.3.0.0/16",
    "CidrBlockSet": [
      {
        "CidrBlock": "10.3.0.0/16"
```

```
      }
    ],
    "OwnerId": "985819153420",
    "VpcId": "vpc-0feff817ac38e9de8",
    "Region": "us-east-2"
}
```


The result set returned indicates that VPC-1 has peered with two other different VPCs. This shows that additional peering relationships was established with VPC-3 and VPC-4 in the us-east-2 region in another AWS account. The VPC identifier has been bold to show the differently marked VPCs. The result set also indicates large CIDR address ranges for VPC-3 and VPC-4, indicating peering with the entire CIDR space for VPC-3 and VPC-4.

Since a VPC peering relationship only establishes the relationship between two VPCs, additional route tables and security groups checks need to be done to determine the exact connections between the subnets and applications. If the routes are overly permissive, or origin subnets are not open in the right area, it can eventually lead to over-exposure of the data set in the target VPC. To continue the check for connectivity, the describe route-tables command can be used to determine which routes and associated subnets have opened access from VPC-1 (10.1.0.0 subnet) to VPC-3 (10.3.0.0 subnet). The filter here searches through the route table of VPC-1 subnets to determine how many routes have a destination set to the VPC-3 (10.3.x.x) subnet. The grep command restricts the search criteria to 10.3, so that more defined routes may be found. In this case, only one entry is returned.

```
aws ec2 describe-route-tables | jq .RouteTables[].Routes[].DestinationCidrBlock
|grep 10.3.

"10.3.0.0/16"
```

There is one route entry that has been opened towards the entire range of vpc-3. The command can be iterated through to determine the subnets in VPC-1 that utilizes this route table. This query is the final step which determines the outbound pathway of connection from VPC-1 subnets to VPC-3.

```
aws ec2 describe-route-tables --filter Name=route.destination-cidr-
block,Values=10.3.0.0/16 | jq .RouteTables[].Associations[].SubnetId
```

Andy Huang, tokidoki2@gmail.com

```
    "subnet-04afd108631b916c7"
    "subnet-039ad8877d8ced078"
    "subnet-01b27218f3072850a"
    aws ec2 describe-route-tables --filter Name=route.destination-cidr-
block,Values=10.3.0.0/16 | jq .RouteTables[].VpcId
    "vpc-0e3d561731ef2ecf9"
```

The result set shows 3 subnets from VPC-1. The team can now determine whether they are the correct subnets that should be opened to allow outbound access from VPC-1 by utilizing commands to query the tags of the subnets to show that this is indeed a private subnet that is allowed to connect to vpc-3.

```
    aws ec2 describe-subnets --filters Name=vpc-id,Values=vpc-0e3d561731ef2ecf9
Name=subnet-id,Values=subnet-04afd108631b916c7 Name=tag-key,Values=Name | jq
'.Subnets[].Tags[]'
    {
      "Key": "Name",
      "Value": "vpc1-private-az2"
    }
```

The result shows that a private subnet is allowed outbound to target VPC-3 subnets. The intention may be for a VPC-1 private subnet to reach VPC-3 private subnet, and in this case, the target route IP address space of 10.3.0.0/16 is too wide and should be re-adjusted to the private subnet space of vpc-3. A similar trace can be run from VPC-3 to determine connectivity back to VPC-1 and whether the allowed CIDR ranges are too broad. The command set can be put into a set of scripts to generate reporting that allows the detection of over-permission outbound for the private subnet, and to verify that the whether the accessible subnet space is too broad for the intended use case.

5.3. Tracing through a private-link connection

AWS Privatelink allows for point-to-point connections and fan-in connections where many clients can connect to a Privatelink provider, also called endpoint service. The automated diagramming models from CloudMapper can capture the existence of VPC endpoints in the request VPC; however, it is not able to shed further details on the where those endpoints have connections. On the provider side, there are no indications at all on inbound connections from the CloudMapper diagram. Tracing from the VPC endpoint

service (provider) perspective, can uncover details on how many connections and who has connections to the service. The describe-vpc-endpoint-service-configurations command provides details of any particular service endpoints. This command is executed in the context of VPC-1 to list the endpoint service from the provider perspective.

```
    aws ec2 describe-vpc-endpoint-service-configurations --filter Name=service-
name,Values=com.amazonaws.vpce.us-west-2.vpce-svc-02fe6c02eccfa4162 |jq
.ServiceConfigurations[].ServiceId

    "vpce-svc-02fe6c02eccfa4162"
```

The result is the VPC endpoint service id which can be used to enumerate other information.

```
    aws ec2 describe-vpc-endpoint-service-permissions --service-id vpce-svc-
02fe6c02eccfa4162
    {
        "AllowedPrincipals": [
            {
                "PrincipalType": "All",
                "Principal": "*"
            }
        ]
    }
```

This particular response detected the overexposed permission in the AllowedPrinciples list that was seen in the graphical console earlier. This is something the team needs to look into in order to tighten the exposure of this endpoint service. The next set of commands can determine what connections have been established to the endpoint service.

```
    aws ec2 describe-vpc-endpoint-connections --filter Name=service-id,Values=vpce-
svc-02fe6c02eccfa4162
    {
       "VpcEndpointConnections": [
           {
               "ServiceId": "vpce-svc-02fe6c02eccfa4162",
               "VpcEndpointId": "vpce-0659685f9029e0cf3",
               "VpcEndpointOwner": "850774750078",
               "VpcEndpointState": "pendingAcceptance",
               "CreationTimestamp": "2019-10-12T23:08:12.000Z"
```

```
        },
        {
            "ServiceId": "vpce-svc-02fe6c02eccfa4162",
            "VpcEndpointId": "vpce-0401e7168cc19f4bb",
            "VpcEndpointOwner": "850774750078",
            "VpcEndpointState": "available",
            "CreationTimestamp": "2019-10-07T03:28:37.000Z"
        }
    ]
}
```


This result shows that there are two VPC endpoint connections established to this Privatelink provider endpoint service, from two different VPCs (vpc endpoint `vpce-0659685f9029e0cf3` and vpc endpoint `vpce-0401e7168cc19f4bb`). Since information regarding endpoint is based on the requestor side, we can gather the endpoint owner data and proceed to the owner VPC to determine the structure of the connection from that account.

Once on the VPC endpoint (requestor) owner account 8507-7475-0078, a series of steps can be run to determine information regarding VPC endpoint connections and information related to the subnets. The goal is to ensure that the proper subnets have established the Privatelink connection and that no additional permissions are granted. The service name remains the same as in the previous query, and the vpc-id can be gathered via a command using the describe-vpc-endpoints with the VpcEndPointId as a reference value. Using the endpoint `"vpce-0659685f9029e0cf3"` to retrieve the vpcId, can determine which subnet within the VPC is using this VPC endpoint.

```
    aws ec2 describe-vpc-endpoints --filter Name=vpc-endpoint-id,Values=vpce-
0659685f9029e0cf3 |jq .VpcEndpoints[].VpcId

    "vpc-0cdd65460d18080f0"

    aws ec2 describe-vpc-endpoints --filter Name=vpc-id,Values=vpc-0cdd65460d18080f0
Name=service-name,Values=com.amazonaws.vpce.us-west-2.vpce-svc-02fe6c02eccfa4162| jq
.VpcEndpoints[].SubnetIds[]

    "subnet-086dd4574ef12a4a2"

    "subnet-02c37b706b2f4ee26"

    "subnet-033aa3e94b09fb112"

    "subnet-0b4ac64caf95448fe"

    "subnet-030d0f45d79ae7596"

    "subnet-0d43fc1ce5b2cfdb7"
```

The resulting six subnets returned can be then used to query specific subnet information. Additional information is gathered about the subnet to determine if this connection is really warranted.

```
    aws ec2 describe-subnets --filters Name=vpc-id,Values=vpc-0cdd65460d18080f0
Name=subnet-id,Values=subnet-086dd4574ef12a4a2 Name=tag-key,Values=Name | jq
'.Subnets[].Tags[]'
    {
      "Key": "Name",
      "Value": "vpc2-ingress-az3"
    }
    aws ec2 describe-subnets --filters Name=vpc-id,Values=vpc-0cdd65460d18080f0
Name=subnet-id,Values=subnet-02c37b706b2f4ee26 Name=tag-key,Values=Name | jq
'.Subnets[].Tags[]'
    {
      "Key": "Name",
      "Value": "vpc2-ingress-az2"
    }
```

Based on the result that indicates two of the subnets that have access are ingress subnets, this should trigger the team to take a closer look to determine if this conforms to policy. These subnets should not be allowed to establish connections to the Privatelink provider unless there are explicit reasons to do so. The last subnet is a legitimate connection, since it is an access from the private subnet.

```
    aws ec2 describe-subnets --filters Name=vpc-id,Values=vpc-0cdd65460d18080f0
Name=subnet-id,Values=subnet-0d43fc1ce5b2cfdb7 Name=tag-key,Values=Name | jq
'.Subnets[].Tags[]'
    {
      "Key": "Name",
      "Value": "vpc2-private-az3"
    }
```

The results show that of the six subnets that have requested connections to this VPC endpoint which provides access to a secure data subnet in VPC-1, three are part of an ingress subnet that has internet interfaces. This represents a risk via access directly from internet accessible subnets to secure data subnets and should trigger a review on whether these connections are warranted. The commands can be placed in a script with IAM roles that can reach into and provide visibility between the 2 different VPCs. The script can be run periodically or expanded from a point to start mapping the connections between

VPCs within AWS. This information can be diagrammed out to become an initial baseline from where teams can determine how to enhance the security stance between multiple projects.

6. Conclusion

Virtual Private Cloud structures are gaining momentum in organizations due to the ease of set up and the isolation properties that they provide. Many development teams are also empowered to start and maintain projects from an end-to-end perspective. The dedicated VPC is a way to isolate and operate independently for the teams. However, applications and services do not operate in complete isolation; they need to call dependent systems and other services to complete transactions.

When applications are within a single VPC and need to call dependent systems in other VPCs, typical approaches for connectivity involve using a peering mechanism or a Privatelink provider/requestor mechanism. In both mechanisms, access is granted between the two VPCs for resources to communicate. However, application teams do not necessarily understand the scope of access needs and they may create overly permissive access between the two VPCs, thereby leading to larger attack surface on both sides. This results in the ability to laterally move between two projects by an adversary that has gained a foothold in one VPC.

To prevent over exposure of access, it is necessary to understand VPC connection technologies and to understand scoping and implementation details. Access restrictions needs to be reviewed periodically to prevent and detect overly permissive configurations. Current mapping technologies map VPC internals and do not map across VPC connections. Cloud providers such as AWS have provided extensive command line and API sets to allow for query of the different aspects of the VPC peering and Privatelink connections. A combination of command can be used to map out the connectivity between VPCs to look for overly exposed attack surfaces.

Andy Huang, tokidoki2@gmail.com

7. Resources

1. Mell, Grance (2011) The NIST Definition of Cloud Computing, NIST SP800-145

2. Jackson, K & Goessling, S (2018) Architecting Cloud Computing Solutions, Virtual private cloud. Retrieved from https://learning.oreilly.com/library/view/architecting-cloud-computing/9781788472425/20e66690-cff1-43f3-98ff-89dec8fec5c6.xhtml

3.Rouse, M (Oct 2019) virtual private cloud (VPC). Retrieved from https://searchcloudcomputing.techtarget.com/definition/virtual-private-cloud-VPC

4. Christudas, B (2019)(Oct, 2019) Practical Microservices Architectural Patterns: Event-Based Java Microservices. Figure 4-4 Retrieved from https://learning.oreilly.com/library/view/practical-microservices-architectural/9781484245019/html/477630_1_En_4_Chapter.xhtml

5. AWS VPC: AWS Documentation (Oct, 2019) Scenario 3: VPC with Public and Private Subnets. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario3.html

6. GCP CLB: GCP Documentation (Oct, 2019) Cloud Load Balancing, Retrieved from https://cloud.google.com/load-balancing/

7. El-shaw, M (March, 2018) Networking in AWS vs. Google Cloud Platform – Design Considerations. Retrieved from http://www.netdesignarena.com/index.php/2018/03/19/networking-in-aws-vs-google-cloud-platform-design-considerations/

8. MS VirtualNet: Microsoft Azure Documentation (June, 2019) Virtual Networks. Retrieved from https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview

9. AWS Routes: AWS Documentation (Oct 2019) Route Tables User Guide. Retrieved from https://docs.aws.amazon.com/en_pv/vpc/latest/userguide/VPC_Route_Tables.html

10. CloudMapper: Cloud Mapper documentation(Oct 2019) Cloudmapper from duo-labs. Retrieved from https://github.com/duo-labs/cloudmapper

11. AWS CLI: AWS Documentation (Oct 2019) AWS CLI command reference. Retrieved from https://docs.aws.amazon.com/cli/latest/reference/

12. AWS Subnets: AWS Documentation (Oct 2019) AWS Subnets with route table. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/WorkWithRouteTables.html#AssociateSubnet

13. AWS SG: AWS Documentation (Oct 2019) AWS Security Groups. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

14. AWS VPCPeer: AWS Documentation (Oct 2019) AWS Peering configuration with controls. Retrieved from https://docs.aws.amazon.com/vpc/latest/peering/peering-configurations-full-access.html

15. AWS PeerAccess: AWS Documentation (Oct 2019) AWS Peering with specific access. Retrieved from https://docs.aws.amazon.com/vpc/latest/peering/peering-configurations-partial-access.html

16. AWS VPCE: AWS Documentation (Oct 2019) AWS VPC Endpoint for AWS services. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-endpoints.html

17. AWS CIDR: AWS Documentation (Oct 2019) AWS overlapping CIDRs. Retrieved from https://docs.aws.amazon.com/vpc/latest/peering/invalid-peering-configurations.html

18. AWS PL: AWS Documentation (Oct 2019) AWS Privatelink Overview. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/endpoint-service.html#endpoint-service-overview

19. AWS PL interregion: AWS Documentation (Oct 2019) AWS Privatelink inter-region peering. Retrieved from https://aws.amazon.com/about-aws/whats-new/2018/10/aws-privatelink-now-supports-access-over-inter-region-vpc-peering/