Access before Authenticate

In this module, we're going to deep dive into the components involved in creating an SDP. We are implementing zero trust using the SDP architecture. We will further our knowledge of SDP controllers, gateways, the authentication and authorization process, the protocols, keys, and certificates used. Also, we will examine the different connectivity models for SDP as proposed by the Cloud Security Alliance. At the end of the module, we'll provide a hands-on demonstration of an SDP multi- cloud scenario. Using VPNs and firewalls to establish zero trust allows a user to connect to services, for example, a mail server. Firewalls can be used to set up bands and IPs, and services can be set up to configure which IPs are good and bad. VPNs can be set up to only let the users on the network who have authorized VPN client and the appropriate keys. Zero trust has seemingly been implemented. Unauthorized users who clone the VPN client can steal the keys and access, for example, the mail server, and then guess other users' names and passwords to perform malicious activity, such as a DDoS or a credential theft. The VPN allows you to log into the network, and not allow the user to use other services, such as SharePoint, that are not on the mail server network segment. But because unauthorized users are already in a network, they can get to the SharePoint server and other services by using hacking techniques. The traditional chain of events is to allow users access to your network, and access to the services, but then hand them off to the services to determine whether the user can access the service or not. This is becoming a really big security issue. Access before authentication allows users, both good and bad, to have access to all the services - not login, but access. Allowing access to the network changes with zero trust, as the concept implies that users aren't allowed any access until they say who they are. Think of security as putting up a wall with multiple doors, and allowing people to come in and pick a lock on the door. We are just relying on the locks for security. It is much better to put a fence around what you want to protect than rely on vetting the threat to keep out the bad traffic. One does want to see who is knocking, but one doesn't want the threat to do bad things, like pick the locks. This is the essence of zero trust. The rest of this section shows you how to implement zero trust using the SDP architecture.

SDP Connectivity Models

SDP is about securing connections regardless of the underlying IP infrastructure, and the Cloud Security Alliance have offered a number of connectivity models for this. Firstly, client-to-gateway. When one or more servers must be protected behind a gateway, the connections between the client and the gateway are secured, regardless of the underlying network topology. Gateways may be in the same location or distributed internationally. In this model, the client is connected to the gateway via a mutual TLS tunnel, where the connection terminates. To secure the connection to servers, additional precautions must be taken. The SDP controller may be located in the cloud or near the protected servers, so the controller and the servers are using the same SDP gateway. Next, client-to-server. When an organization wants secure connections end-to-end, this model combines a server and a gateway in a single host, where trust is established for the connection to the gateway. The client may be located in the same location as a server or distributed. In either case, connection between the client and the server are secured end-to-end. This is in comparison to the client-to- gateway connectivity model that does not secure end-to-end. This model provides organizations a great deal of flexibility, because the server-to-gateway combinations can be moved between multiple cloud providers as needed. This model is also appropriate for securing on-premise legacy applications that cannot be upgraded. In this model, the client is connected directly to a secure server via a mutual TLS tunnel, at which point, the connection terminates. The SDP controller may be located on the server so the controller and the server are using the same gateway, or located in the cloud. This server is protected behind an SDP gateway, which acts as the accepting host. This secure connection to the server, going through the gateway may be under the control of the owner of the application services in the server, giving the owner full control of these connections. Next, server-to-server. This model is best suited for Internet of Things and VM environments, and ensures that all connections between servers are encrypted regardless of the underlying network or IP infrastructure. The server-to-server model also ensures communications are explicitly permitted by an organization's SDP white list policy. Communications between servers across untrusted networks are secured, and servers remain hidden from all unauthorized connections using a lightweight SPA protocol. This model is similar to the client-to- server model above, except that the initiating host is itself a server, and can also act as the SDP accepting host. Like the client-to-server model, the server-to-server model requires that the SDP gateway or similar lightweight technology be installed on each server, and renders all server-to-server traffic dark to all other elements in the network ecosystem. Network-based IDS and IPS's will need to be configured to get drop packets from the SDP gateway instead of externally. In addition, organizations may relay on host-based IDS, IPS systems. The SDP controller may be located on the servers, so that the controller and the servers are using the same SDP gateway. The SDP controllers may also preside in the cloud; client-to-server-to-client. In some instances, peer-to-peer traffic must pass through an intermediary server; such as IP phone, chat, or video conferencing services. In these cases, the SDP hides the IP address of connecting clients, encrypts the network connections between the components, and protects the servers from all unauthorized network connections via SPA. The SDP controller may be located on the servers, so the controllers and servers are using the same gateway, or in the cloud. As depicted above, the server is protected behind the SDP gateway, acting as the accepting host. To secure connections to the server going through the gateway, are controlled by the owner of the application services on the server by default. Finally, gateway-to-gateway. The gateway-to-gateway model was not included in initial publication of the SDP specification version 1. This model is well-suited for certain IoT environments. In this scenario, one or more servers sits behind the accepting host, such that the accepting host acts as the gateway between clients and servers. At the same time, one or more clients sit behind the initiating host, such that the initiating host also acts as the gateway. In this model, client devices do not run the SDP software. The devices may include those which are not desirable or possible to install the SDP client; such as printers, scanners, certain sensors, and IoT devices. In this model, the gateways operate as firewalls, and also potentially as a route or a proxy, depending on the implementation.

SDP Controller

The SDP controller is a centralized policy enforcement engine that governs the control and data plane for the SDP components. For centralized authentication and authorization, the SDP controller keeps track of users, devices, and applications. It manages all of the SDP components, and the connections to the services behind the gateways. It acts as the controller, passing information between the initiating hosts and accepting hosts. This controller allows initiating and accepting hosts to be authenticated prior to allowing communication, and for the controllers, it determines the list of accepting hosts to which the initiating host is authorized to communicate to. This makes sure that the endpoint is fulfilling the enforced security posture assessment before access is granted to the network. In some cases, the SDP controller may relay information to external authentication services to determine its decision on what the requesting host can access. A key note about the controller is the type of integration points it can support. The SDP controller should have the capability to connect to authentication and authorization services, for example, PKI, device fingerprinting, geolocation, SAML LDAP, and Kerberos. The controller is responsible for generating and coordinating all cryptographic keys and client certificates for each client, ensuring that each client has up-to-date, valid keys and certificates. It maintains active connection information by receiving and storing connection tracking data reported by the SDP gateways. It generates encryption, HMAC keys, and client certificates. These will include the SPA encryptions, SPA HMAC, TLS private keys, and TLS certificates. The cryptographic keys are used for sending SPA packets, and the client certificates are used for the mutual TLS to the controller and to the services protected by the controller. When a controller provides a client with new data, it updates this information to all of the relevant SDP gateways, and also the associated user services that rely on this data to authenticate the user. All information about the user, device security state, along with, for example, geolocation information, is taken from the controller and sent to the gateways so it can carry out its tasks.

SDP Gateway

The gateway can be established in either a public cloud or on-premise location, and close to the requesting resource. The geolocation integration point is useful so that only certain locations, due to security reasons, can assess information, or enhance the user experience by redirecting the services that are logically closer to the gateway that is protecting those services. The gateway monitors incoming traffic at a TCP/IP level to identify connection attempts from the initiation hosts. It silently monitors this traffic, and when a valid connection request is identified, it dynamically modifies the firewall policy to accept the connection from that client. The gateway dynamically opens and closes ports on the firewall to enable the client to connect to the desired service that is posting behind the SDP gateway, but only once they have been properly authenticated and authorized to access that service. The new firewall rules that it's only open for a very short period of time. As soon as the client has connected to the desired service through the firewall, the firewall is again modified, returning to a drop all. The initiating hosts and accepting hosts are still able to communicate, because the firewall recognizes their communications as an established connection. All SDP gateways maintain a persistent connection to the SDP controller while operational. Any time that a relevant change occurs at the controller, such as change regarding what services our particular SDP client or group has permission to access, this information is immediately propagated to all relevant SDP gateways. With this persistent connection, it tracks and reports connections to the SDP controllers. It closes connections when instructed to do so, by the SDP controller. For example, if a user's access to an entire network is revoked, all relevant gateways are immediately notified, and any open connections between the SDP client and protected services are immediately shut off. The SDP gateway needs to know the encryption, HMAC keys, and client certificates to make the connection. The SDP gateway needs to know the following credentials; TLS keys, TLS certificates, SPA encryption key, and SPA HMAC key. These credentials are taken from the SDP controller. Mutual TLS tunnels are dynamically created either through the gateways between the clients and the services, or terminating on the gateways depending on the connectivity model chosen. The SDP gateway handles the authentication and authorization for all the connections after the firewall port is opened. No client is routed directly to a particular service. The SDP gateway receives all service requests, and is designed to prevent bottlenecks, as well as provide a TLS endpoint for connections to services. The gateway identifies and authenticates a SPA packet. The SPA message should be encrypted, and the default option is AES, for symmetric encryption. GPG asymmetric encryption is also supported. The AES encryption layer is implemented as defined in RFC 3602. The encrypted SPA packet must be signed with Message Authentication Code, MAC. This signature must use a key shared between a gateway and the client. The gateway holds a unique key per a client ID. The MAC string must be appended to the encrypted SPA message, and must include a clear client ID if present in the verified payload.

SDP Simulation

In its simplest form, we can strip the SDP architecture down to the following. Our requesting SDP client, which is the initiation host, that could for example, be a desktop, a mobile device, or a laptop; an SDP gateway and the firewall, along with the service that the SDP needs to protect, which is the accepting host. The service in this case is a hosting web service, and is protected by the SDP gateway. The SDP gateway works in conjunction with the firewall. The SDP gateway is firewall knock operator, and implements the authorization, Single Packet Authorization. In a normal working environment, in order for users to get access to services behind the firewall, you would have to open ports on the firewall to permit access. This presents security risks, as a bad actor could directly access the service via the open port, or exploit any vulnerability of that service itself. However, within the SDP architecture, the firewall, that is IP tables, has its policy configuration set to deny all. You can use any type of firewall here that supports connection tracking. With the SDP architecture, the firewall gateway combination is placed in front of the services or services that are protected, with a deny all firewall ruleset. Network scans will therefore not show any services present on a network being scanned. Unauthorized users cannot access what we cannot see, making the web service invisible to the outside world. Nmap is a tool used to conduct port sweeps across networks, looking for targets with known vulnerabilities. All attempts to connect to the website are stopped by the firewall, so if a bad actor runs a port scan to see what open ports are to break into your system, the SDP can make it infeasible for anyone to even detect services, let alone exploit a vulnerability or attempt to brute force a password, which is commonly done against SSHD daemons. HMAC as defined in RFC2104 is the preferred MAC scheme. HMAC is well known and tested, and there's no known problem, which is a good key to use. An advantage of HMAC is that it doesn't require any state tracking. The encrypted SPA message could also be a alternatively signed using GMAC. In this case, GMAC 4 is based on a counter that the client must track. This must be unique, and never reused for a given key. For this reason, when using GMAC, each client device must use a unique key. So we have mentioned SPA a few times, so let's go a little bit deeper. A SPA packet is a UDP packet, it's encrypted, cryptographically signed, and cannot be faked without using the legitimate users keys in order to formulate the packet. No two packets are ever the same; therefore, replay attacks simply do not exist. SPA is intentionally designed to be multiple layers of security built on each other. SPA itself is a layer of security on top of whatever port it is protecting. One of the paradigms is that the layers cannot be attacked simultaneously, as no acknowledgement is given if a layer is compromised. If an attacker happens upon a valid HMAC signature, the packet would still not be accepted if the encryption error's not correct. The SPA and SDP is focused on device authorization, not on user authorization, as in the SPA RFC. The user authorization happens inside of the mutual TLS connection. Credential theft is a major concern to large organizations, and they wanted the user credentials to be transmitted over a secure link. The legitimate client will still have an SDP client program running on their device, and this client program sends a SPA packet. While implementations of SPA may differ slightly, they should share the following. Packets must be encrypted and authenticated. Packets must self-sign all necessary information. Packet headers alone are not trusted. Packets must not depend on admin or route level access in order to generate and send no raw packet manipulations allowed. Servers must receive and process packets as silently as possible. No response or verification will be sent. The benefits of SPA include that it retains all the benefits of port locking, but has many other advantages over this, such as, it can utilize asymmetric ciphers for encryption. SPA is authenticated with a HMAC in an encrypt then authenticate model. SPA packets are non-replayable. SPA cannot be broken by trivial sequence busting techniques. SPA only sends a single packet over the wire, making it much faster. In our case, when a client wants to connect to the web server, it sends a SPA packet to the SDP gateway. The initial SPA packet contains all the relevant information about the requesting users and their devices that are verified before every connection. All other packets are a drop, as they are by default from unauthorized users. A unique SPA is triggered, and the information in the SPA packet must match the ID of the user. This is the key that opens the SDP to the client, which is the port on the firewall. With this single SPA packet, valid connection requests are processed by the SDP, which then creates a rule dynamically on the firewall after the authentication encryption tokens inside the SPA packet are verified. Within our example, prior to opening up the firewall to permit connectivity, the encryption and authentication token in the initial SPA packet is authenticated based upon the authentication of the required by the service, thereby enforcing user authentication policies defined by the organization for every single connection. Once the SPA packet is sent to the SDP gateway that is sniffing the TCP/IP stack and plot checks for the interesting UDP packets that are arriving on the right port, it will then verify the HMAC signature and decrypt the packet. By doing these two steps, the SDP gateway knows it has a legitimate user sending a legitimate SPA packet.

SDP Authentication and Authorization

Authentication is the process or action of verifying the identity of a user or a process. The policies that govern the authentication process may require single or multiple factors. SDP uses a combination of both the user and device authentication for each individual connection between initiating and accepting hosts. Within the authentication state, we have what's known as authenticators. These are factors that are presented by users to the system or application to verify they are who they claim they be. This could be a password, a cryptographic key, a fingerprint, or some other type of biometric data. The connections between all hosts must use mutual authentication to validate the devices as an authorized member of the SDP prior to further device validation and/or user authentication. This can be done with SPA, host-specific firewalls, mutual TLS, device fingerprinting, software verification, and geolocation. User authentication can be provided by, for example, a trusted browser, SAML, an authentication to an identity provider. The second stage is authorization, which allows the user access to various resources based on the user's identity. SDP used the combination of user authorization, as well as device verification or attestation. Authorization can be done by, for example, the SDP gateway. Once a user or device is authorized, the SDP gateway allows access to protected processes or services. Geolocation can be used as a source of information upon which to make access decisions to an SDP. For example, access to resources from users in certain countries may be blocked. The authorization SDP can also be done with Single Packet Authorization. SPA plays a critical role by hiding servers, including the SDP components, such as the controller and gateway, until and unless the initiating host sends a valid SPA packet as an initial connection request. Next in our example, the authorization phase occurs, and SDP gateway checks that the client has access to the requested service. If so, the gateway will dynamically open the desired port on the firewall, so the client can access the service. It does not, however, send response back to the client, thereby ensuring zero trust. We now have a new accept rule in the IP tables, allowing a user from a specific source address for a limited time granted access only to the desired service. So we now have a port open for a single service that was dynamically created. The SDP gateway adds a rule to the firewall dynamically, so that a mutual TLS connection to the web server can be established. With a mutual TLS session, both hosts must present signed certificates to each other, negotiate an encryption scheme, and agree to a shared secret key. Once this connection's established, all further communications are securely encrypted. The negotiation of the mutual TLS connection within the open source SDP is a standardized process. Once the connection has been established, the gateway will remove the rule from the firewall, so the port is not open. As a result, for the majority of time, the ports are just not open. After a period of time, the gateway will instruct the firewall back to its previous deny all, and even when the port is now closed, the client will still have connectivity to the desired service, due to the connection tracking on the firewall. All of these steps happen in the background for a seamless user experience. Any attempt to connect to the SDP service must know the SPA packet, and even if the packet can be recreated, the sequence numbers, which are random, of the packet must be established prior to the connection.

SDP Demonstration

In the previous modules, we discussed an SDP architecture, which is client initiated. In order to get access to protected services behind SDP, the client has initiated connection that triggers the authentication. This can be done with a SPA packet, or some other lightweight security protocol. As a path is created to the client, the SDP needs to have a route to the client. In this section that includes a hands-on demo, we are going to further our knowledge and discuss a variant to this SDP architecture. It's service-initiated, where both the requesting client and authorized service, SDP broker or cloud service. Let's call this an SDP cloud. The SDP cloud becomes the globalized connection point. Think about a Zoom meeting; you're connected to the other party, but there is a broker in the middle. This allows you to be anywhere without caring or knowing where the other party is. This type of SDP architecture provides additional benefits in the sense that you can use overlapping IP addresses. You can have applications with the same IP address in different parts to the world, which could be useful for mergers acquisitions. Also, applications can literally exist anywhere, as you don't need an inbound path which offers additional levels of security against DDoS. With the service-initiated model, the client does not connect to the location of the application, or have any idea where it is. Instead, trust is built between the two sides of the service. There is a service-initiated connection from the client to the SDP cloud on the internet, there's also a service-initiated connector from an SDP device that sits near the application to the global SDP service. Between these paths, trust is initiated. Only once this is being initiated, and the access control approved, will the user be able to see and connect to the application. There are really only three pieces to this architecture. There's an SDP client application on the client device, which has a sole role of just forwarding traffic once it gets the policy from the SDP cloud. The policy in the SDP cloud is old enforcement, and the user connects to the most effective cloud location for the location they are in. Here, we have policy connectivity and policy enforcement. Again, we are using mutual TLS, but pinned certificates with a trust back to the CA. Then we have another SDP device known as a connector running near the application. Similar to the client, the SDP connector connects outbound, so both the client and the connector connect outbound on port TCP 443 to the SDP cloud. Once security controls are established, a TCP session will then be created from the connector to the application on behalf of the client. This architecture completely isolates the application from network access. All applications and infrastructure are completely dark. For applications within our multi-cloud demo, you won't even be able to resolve the domain names, because it's not an actual publicly reachable domain. The application's housed in both clouds, Google and Azure, are not known until the users are properly authenticated and authorized. Once security controls are passed, there is still no direct connectivity path between the user and the application. There is just an outbound tunnel path from the connector and the client to the SDP cloud. In this multi-cloud demo, we are using domain names that are 100% private. There is no way to get to them publicly. The two domain names specified on the screen are not real domains, you can't find them anywhere. These are application stacks in two of the public clouds. Both of these application stacks could be using the same IP address, it wouldn't matter to the access. But when the user has the SDP client running, there is a route path for those domains, even though the user is not on the actual network. Access path is managed through the SDP cloud. As you can see from the client, service status is turned off. So let's try and ping the first domain name. Let's try curl. This proves that the application stack is completely dark to the outside. As far as the internet is concerned, these domains do not exist. So let's go a little bit further and actually go to the browser. As you can see, nothing exists. Now let's go to the client and connect to the SDP. So let's go back to the CLI and see if we can ping these domain names. As you can see, the ICMP response is an IP address that doesn't exist on the public network. It exists in the SDP private environment. Even though we are going through the internet, we can resolve the IP address, but the IP address you get is not pingable. With SDP, the IP address is irrelevant. We are abstracting application from the network. Now, go to the browser. Now you can see you have access to these applications, but anyone else that doesn't have access to the SDP environment will not be able to see these applications. So you may be thinking that this is just like a normal VPN, but keep in mind these domain names, .ggl and .azu, don't even exist. In terms of top-level domains, there is no .ggl or .azu. Also, the response from the ICMP returned back a private address. The key point here is that the entire application is dark to the user, even if a user wants to access the application, they don't even know where it is, as the SDP abstracts the application from the network. Within the SDP cloud, a path is created from the connector that is located close to the application out to the SDP. There are two outbound tunnels. There's an outbound tunnel from the client side and the connector that sits next to the application. Both of these tunnels provide a path for that application access, and nothing more. Think of it in terms of an old telephone system; first you would have to call the operator to get connected. This inverses our entire network in a security model, and makes the application completely invisible to the internet. The user is never given access to the network. The way security was initially set was in order to get access to an application. You would have to connect a user to a network. Users that were not on the network, for example a remote worker, would need to create a VPN. To enable connectivity, users had to connect to the network where applications were housed. An inbound port needed to be exposed to the public internet; however, this port is visible to anyone on the internet, not just a remote worker. So from a security standpoint, the idea of network connectivity to access an application is not a very good idea. However, with this SDP model, the SDP connector gives a direct path for the user, there's no need to add latency or backhaul the user to a central location first for security screening. They connect directly to the connector via the SDP cloud. The application location becomes completely irrelevant and dark to the user. So let's jump onto the UI and see how this is built. As you can see, the first screen checks the top applications for a bandwidth error message and policy issues; and we've also a way to dynamically discover unknown applications. Within the health section shows the health of the application based on their location and connectivity path. The connection displays a status of the connection, whether the connection is successful or not, and shows how many bytes are sent per application, and also things like connection ID. The important point to note is that it's bytes per application. With a traditional VPN, you would see that user A has connected for a period of time, and transmitted a certain amount of data, but here, we show each individual application. The policy is where the user tried to access an application, and what policy was hid. There are basically two types of policies; A policy that permits and denies, which is generic, but also a policy that can permit, for example, on extended perimeters. So just the time of day or application access. This expires a SAML certificate for the client, which can be very useful for sensitive applications. The user tab displays identity of who was authenticated. It shows the egress IP address, and this IP address is linked to a location. You can also see the type of identity provider that authenticates the user. You can use different identity providers for different users, based on the trust of the user. It may be useful to have a couple of different identity providers for third-party access. The ZEN is the SDP cloud where the user and application tunnels were brokered, essentially, the policy enforcement infrastructure. With the connector, we are getting closer to the application. This displays where the connector is. The connector is simply a piece of software that could be in a cloud or on-premise, but it's always close to the application side. It uses a random hide port and creates a TCP session on behalf of the user to connect to the application. The connector makes a connection from itself to the application, so the application will only see a TCP connection from the connector. You can also measure performance. This shows how long it took the connector to make the connection to the application. Finally, the application. This is the application that the user is actually connected to.

Course Summary

In this course, we discussed a need to leverage zero trust to create a software- defined perimeter, along with the many challenges we were having using current network and security architectures. We also discussed the functions of SDP and associated benefits that it brings to organizations. We dived into the components for creating an SDP, and demonstrated the service initiative approach with a multi-cloud environment. SDP is not about putting a lock on a network to defend against attacks. SDP makes the entire network infrastructure and the applications dark to anyone who is not entitled to see them. You cannot attack what you cannot see. I'd like to thank you for joining me on this SDP journey. I hope you enjoyed this course. I thoroughly enjoyed making it for you. This is Matt Conran from Network Insight, expect more from me on zero trust in the near future.