Software Defined Perimeter Working Group

SDP Specification 1.0

April 2014

Acknowledgments

Authors

Brent Bilger

Alan Boehme

Bob Flores

Zvi Guterman

Mark Hoover

Michaela Iorga

Junaid Islam

Marc Kolenko

Juanita Koilpilla

Gabor Lengyel

Gram Ludlow

Ted Schroeder

Jeff Schweitzer

Table of Contents

Acknowledgments 3
Status of This Memo 6
Abstract 6
1 Introduction 7
1.1 Audience 7
2 Design Objectives 7
3 System Overview 7
3.1 The Changing Perimeter 7
3.2 SDP Concept 8
3.3 SDP Architecture 8
3.3.1 SDP Controller 9
3.3.2 Initiating SDP Hosts 9
3.3.3 Accepting SDP Hosts 9
3.4 SDP Workflow 10
3.5 SDP Implementations 11
3.5.1 Client-to-Gateway 11
3.5.2 Client-to-Server 11
3.5.3 Server-to-Server 11
3.5.4 Client-to-Server-to-Client 11
3.6 SDP Applications 12
3.6.1 Enterprise Application Isolation 12
3.6.2 Private Cloud and Hybrid Cloud 12
3.6.3 Software as a Service 12
3.6.4 Infrastructure as a Service 12
3.6.5 Platform as a Service 12
3.6.6 Cloud-Based VDI 12
3.6.7 Internet-of-Things 13
3.7 SDP's Relationship to IKE/IPsec and TLS 13
4 Glossary 13
5 SDP Protocol 15
5.1 Onboarding 15
5.2 Single Packet Authorization (SPA) 15
5.3 Mutual TLS or IKE 16
5.4 Device Validation 17
5.5 AH-Controller Protocol 17
5.5.1 Login Request Message 17
5.5.2 Login Response Message 17
5.5.3 Logout Request Message 18
5.5.4 Keep-Alive Message 18

5.5.5 AH Services Message 18
5.5.6 IH Authenticated Message 19
5.5.7 Reserved for Private Use 19
5.6 AH to Controller Sequence Diagram 19
5.7 IH-Controller Protocol 20
5.7.1 Login Request Message 20
5.7.2 Login Response Message 20
5.7.3 Logout Request Message 21
5.7.4 Keep-Alive Message 21
5.7.5 IH Services Message 21
5.7.6 Reserved for Private Use 22
5.8 IH to Controller Sequence Diagram 22
5.9 IH-AH Protocol in Dynamic Tunnel Mode (DTM) 23
5.9.1 Keep-Alive Message 23
5.9.2 Open Connection Request Message 23
5.9.3 Open Connection Response Message 23
5.9.4 Data Message 23
5.9.5 Connection Closed Message 23
5.9.6 User-Defined Message 24
5.10 IH to AH Sequence Diagram (Sample) 24
5.11 Controller determines the list of AHs to which the IH connects 25
6 Logging 25
6.1 Fields of a log message 25
6.2 Operations 25
6.3 Security 27
6.4 Performance 28
6.5 Compliance 28
6.6 Security Information and Event Management (SIEM) integration 28
7 SDP Specification 28

Status of This Memo

This document outlines a Cloud Security Alliance (CSA) initiated protocol for the Software Defined Perimeter specification, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited. This document is patterned after RFC 4301 Security Architecture for IP.

Abstract

This document describes the "Software Defined Perimeter (SDP) protocol," which is designed to provide on-demand, dynamically provisioned, air-gapped networks. Air-gapped networks are trusted networks that are isolated from all unsecured networks and this may allow them to mitigate network-based attacks. The SDP protocol is based on workflows invented by the Department of Defense (DoD) and used by some Federal Agencies. Networks based on these workflows provide a higher level of security, but are thought to be very difficult to use compared to traditional enterprise networks.

The Software Defined Perimeter (SDP) has adapted the generalized DoD workflow but has modified it for commercial use and made it compatible with existing enterprise security controls. Where applicable, SDP has followed NIST guidelines on cryptographic protocols. SDP can be used in government applications such as enabling secure access to FedRAMP certified cloud networks as well as enterprise applications such as enabling secure mobile phone access to public clouds.

1 Introduction

This document specifies the base architecture for Software Defined Perimeter (SDP)-compliant systems. The protocol is divided into two sections: one describes the control plane and the other describes the data plane. The control plane describes how Initiating Hosts (IH) and Accepting Hosts (AH) communicate with the Controller. The data plane describes how IHs communicate with AHs.

1.1 Audience

The target audience for this document is implementers of the SDP protocol.

2 Design Objectives

The design objective of the SDP protocol is to provide an interoperable security control for IPv4 and IPv6 including obfuscation and access control of the Controllers and of the services protected by the AHs, as well as, the confidentiality and integrity of communication from the IH to the Controllers and AHs.

The specification provides the protocol for the control plane and one option for the data plane. It is expected that additional options will be offered for the data plane.

3 System Overview

The goal of this section is to provide a high level overview of the protocol and to define the components involved in the protocol. Detailed implementation is provided in later sections.

3.1 The Changing Perimeter

Historically, enterprises deployed a perimeter security solution in their data center to protect against external threats to their application infrastructure. However, the traditional perimeter model is rapidly becoming obsolete for two reasons:

1. Hackers can easily gain access to devices inside the perimeter (for example via phishing attacks) and attack application infrastructure from within. Moreover, this vulnerability continues to increase as the number of devices inside the perimeter grows due to Bring Your Own Device (BYOD), on-site contractors, and partners.

2. Traditional data center infrastructure is being supplemented with external resources such as PaaS, IaaS, and SaaS. Subsequently, networking equipment used for perimeter security is topologically ill-located to protect application infrastructure.

The growth of devices moving inside the perimeter and the migration of application resources to outside the perimeter has stretched the traditional security model used by enterprises. Existing workaround solutions that involve backhauling users to a data center for identity verification and packet inspection do not scale well. A new approach is needed that enables application owners to protect infrastructure in a public or private cloud, a server in a data center, or even inside an application server.

3.2 SDP Concept

The SDP aims to give application owners the ability to deploy perimeter functionality where needed in order to isolate services from unsecured networks. SDPs replace physical appliances with logical components that operate under the control of the application owner. SDPs provide access to application infrastructure only after device attestation and identity verification.

The principles behind SDPs are not entirely new. Multiple organizations within the Department of Defense (DoD) and Intelligence Communities (IC) have implemented a similar network architecture based on authentication and authorization prior to network access. Typically used in classified or high-side networks (as defined by the DoD), every server is hidden behind a remote access gateway appliance to which a user must authenticate before visibility of authorized services is available and access is provided. An SDP leverages the logical model used in classified networks and incorporates that model into standard workflow (Section 2.4).

SDPs maintain the benefits of the need-to-know model described above but eliminate the disadvantages of requiring a remote access gateway appliance. SDPs require endpoints to authenticate and be authorized first before obtaining network access to protected servers. Then, encrypted connections are created in real time between requesting systems and application infrastructure.

3.3 SDP Architecture

In its simplest form, the architecture of the SDP consists of two components: SDP Hosts and SDP Controllers. SDP Hosts can either initiate connections or accept connections. These actions are managed by interactions with the SDP Controllers via a secure control channel (see Figure 1 on the following page). Thus, in SDPs, the control plane is separated from the data plane to enable a completely scalable system. In addition, all of the components can be redundant for scale or uptime purposes.

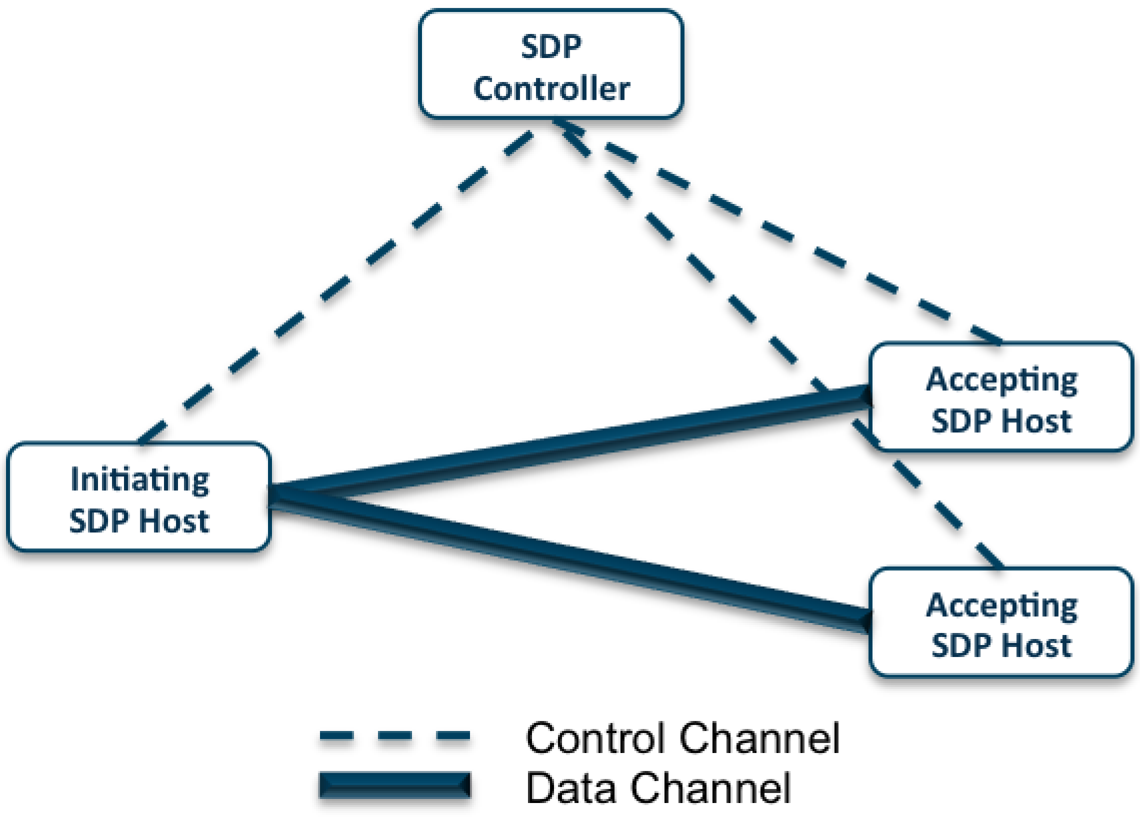

Figure 1: The architecture of the Software Defined Perimeter consists of two components: SDP Hosts and SDP Controllers

3.3.1 SDP Controller

The SDP Controller determines which SDP Hosts can communicate with each other. The Controller may relay information to external authentication services such as attestation, geo-location, and/or identity servers.

3.3.2 Initiating SDP Hosts

Initiating SDP Hosts (IH) communicate with the SDP Controller to request a list of Accepting Hosts (AH) to which they can connect. The Controller may request information such as hardware or software inventory from the IH before providing any information.

3.3.3 Accepting SDP Hosts

By default, an AH rejects all communication from all hosts other than the SDP Controller. The AH accepts connections from an IH only after instructed to do so by the Controller.

3.4 SDP Workflow

The SDP workflow has the following workflow (see Figure 2).

1. One or more SDP Controllers are brought online and connected to the appropriate optional authentication and authorization services (e.g., PKI Issuing Certificate Authority service, device attestation, geolocation, SAML, OpenID, OAuth, LDAP, Kerberos, multifactor authentication, and other such services).

2. One or more AHs are brought online. These hosts connect to and authenticate to the Controllers. However, they do not acknowledge communication from any other Host and will not respond to any non-provisioned request.

3. Each IH that is brought on line connects with, and authenticates to, the SDP Controllers.

4. After authenticating the IH, the SDP Controllers determine a list of AHs to which the IH is authorized to communicate.

5. The SDP Controller instructs the AHs to accept communication from the IH as well as any optional policies required for encrypted communications.

6. The SDP Controller gives the IH the list of authorized AHs as well as any optional policies required for encrypted communications.

7. The IH initiates an SPA to each authorized AH. It then creates a mutual TLS connection to those AHs.

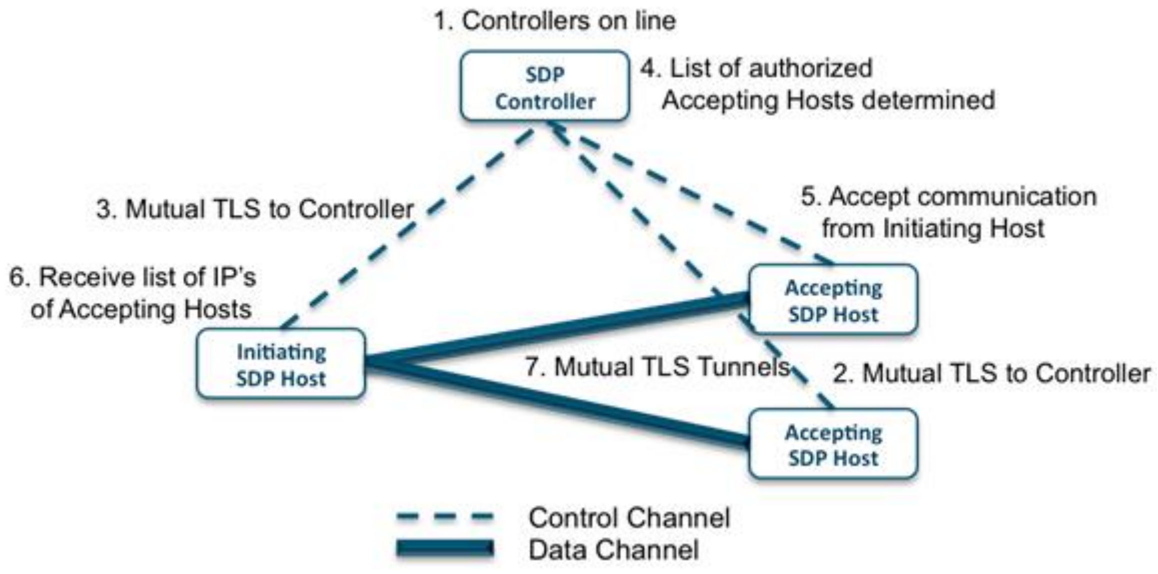
1. Controllers on line
SDP Controller
4. List of authorized Accepting Hosts determined
3. Mutual TLS to Controller
5. Accept communication from Initiating Host
6. Receive list of IP's of Accepting Hosts
Accepting SDP Host
Initiating SDP Host
7. Mutual TLS Tunnels
2. Mutual TLS to Controller
Accepting SDP Host
Control Channel
Data Channel

Figure 2: Workflow of the architecture of the Software Defined Perimeter showing separation between control and data planes.

3.5 SDP Implementations

While the general workflow remains the same for all implementations, the application of SDPs can favor certain implementations over others.

3.5.1 Client-to-Gateway

In the client-to-gateway implementation, one or more servers are protected behind an AH such that the AH acts as a gateway between the clients and the protected servers. This implementation can be executed inside an enterprise network to mitigate common lateral movement attacks such as server scanning, OS and application vulnerability exploits, man-in-the-middle attacks, Pass-the-Hash, and many others. Alternatively, it can be implemented on the Internet to isolate protected servers from unauthorized users and mitigate attacks such as denial of service, SQL injection, OS and application vulnerability exploits, man-in-the-middle attacks, cross-site scripting (XSS), cross-site request forgery (CSRF), and many others.

3.5.2 Client-to-Server

The client-to-server implementation is similar in features and benefits to the client-to-gateway implementation discussed above. However, in this case, the server being protected will be running the AH software instead of a gateway sitting in front of the server running that software. The choice between the client-to-gateway implementation and the client-to-server implementation is typically based on number of servers being protected, the load balancing methodology, elasticity of servers, and other similar topological factors.

3.5.3 Server-to-Server

In the server-to-server implementation, servers offering a Representational State Transfer (REST) service, a Simple Object Access Protocol (SOAP) service, a remote procedure call (RPC), or any kind of application programming interface (API) over the Internet can be protected from all unauthorized hosts on the network. For example, in the case of a REST service, the server initiating the REST call would be the IH and the server offering the REST service would be the AH. Implementing an SDP for this use case can significantly reduce the load on these services and mitigate a number of attacks similar to the ones mitigated above. This concept can be used for any server-to-server communication.

3.5.4 Client-to-Server-to-Client

The client-to-server-to-client implementation results in a peer-to-peer relationship between the two clients and can be used for applications such as IP telephone, chat, and video conferencing. In these cases, the SDP obfuscates the IP addresses of the connecting clients. As a minor variation, a user can also have a client-to-gateway-to-client configuration if the user wishes to hide the application server as well.

3.6 SDP Applications

The SDP can protect servers of all types from network-based attacks. Some of the more interesting applications of SDP are described below.

3.6.1 Enterprise Application Isolation

For data breaches that involve intellectual property, financial information, HR data, and other sets of data that are only available within the enterprise network, attackers may gain entrance to the internal network by compromising one of the computers in the network and then move laterally to get access to the high value information assets. In this case, an enterprise can deploy an SDP inside its data center in order to isolate high-value applications from other applications in the data center and isolate them from unauthorized users throughout the network. Unauthorized users will not be able to detect the protected application, and this will mitigate the lateral movement these attacks depend on.

3.6.2 Private Cloud and Hybrid Cloud

While useful to protect physical machines, the software overlay nature of the SDP allows it to be easily integrated into private clouds to leverage the flexibility and elasticity of such environments. Also, SDPs can be used by enterprises to hide and secure their public cloud instances in isolation, or as a unified system that includes private and public cloud instances and/or cross-cloud clusters.

3.6.3 Software as a Service

Software-as-a-Service (SaaS) vendors can use SDP architecture to protect their services. In this application, the software service would be an AH, and all users desiring connectivity to the service would be the IHs. This allows a SaaS to leverage the global reach of the Internet without its associated security concerns.

3.6.4 Infrastructure as a Service

Infrastructure-as-a-Service (IaaS) vendors can offer SDP-as-a-Service as a protected on-ramp to their customers. This allows their customers to take advantage of the agility and cost savings of IaaS while mitigating a wide range of potential attacks.

3.6.5 Platform as a Service

Platform-as-a-Service (PaaS) vendors can differentiate their offering by including the SDP architecture as part of their service. This gives end users an embedded security service that mitigates network-based attacks.

3.6.6 Cloud-Based VDI

Virtual Desktop Infrastructure (VDI) can be deployed in an elastic cloud in which use of the VDI is paid for by the hour. Unfortunately, if the VDI user needs to access servers inside the corporate network, VDI can be challenging to use and can open up security holes. However, VDI coupled with an SDP solves both of these problems through simpler user interaction and granular access.

3.6.7 Internet-of-Things

A vast number of new devices are being connected to the Internet. Back-end applications that manage these devices or extract information from these devices, or both, are often mission-critical and act as a custodian for private or sensitive data. SDPs can be used to hide these servers and their interactions over the Internet to maximize both security and uptime.

3.7 SDP’s Relationship to IKE/IPsec and TLS

As discussed in the previous sections, SDPs may use protocols such as IKE/IPsec and TLS to create VPNs between IHs and AHs. However, SDPs are not the same as VPNs. The differences between them are outlined below:

- Different amounts of effort are required to create SDP-protected servers compared to VPN gateway-protected servers. In the SDP case, once the SDP Controller is online, the user can create as many protected servers as desired via software and can differentiate authorized from unauthorized users via LDAP associations.

- There are greater capital and operating costs associated with setting up VPN gateways to protect individual servers than an SDP. SDPs are a software construct that can be deployed in a cloud environment.

- SDPs can be used for both security and remote access simultaneously, while VPN gateways cannot. If one were to try to use a VPN client and VPN gateway within an enterprise to protect a server, then users could not use a remote access VPN to access the server (because the VPN client already connected to the remote access VPN gateway). However, SDP communication can occur over a remote access VPN.

- SDP protects against bandwidth denial of service attacks, while VPN gateways do not. AHs can be deployed in a topologically different location than the application server it is protecting, thereby hiding the true location even from authorized users.

4 Glossary

This document uses the following terms.

- **Accepting Host** (AH)
  The Accepting Host accepts the communication from the Initiating Host after the Controller authenticates and authorizes the connection.

- **Agent ID** (AID)

The Agent ID (AID) is a 32-bit unique unsigned value that identifies a given Initiating Host (IH) and/or Accepting Host (AH). Its primary use is during Single Packet Authorization.

- **AH-Controller Path**
  The AH-Controller Path is the channel used for communication between each Accepting Host (AH) and the Controller.

- **AH Session**
  The AH Session is the period of time that a particular Accepting Host (AH) is connected to a Controller.

- **AH Session ID**
  256-bit randomized NONCE managed by the Controller and used to refer to a particular Accepting Host (AH) Session.

- **Dynamical Tunnel Mode (DTM)**
  Dynamic Tunnel Mode (DTM) is the proposed protocol and encapsulation for the IH to communicate with one or more AHs. It is expected that alternative protocols will be proposed.

- **Initiating Host (IH)**
  The Initiating Host (IH) is the host that initiates communication to the Controller and to the AHs.

- **Initiating Host (IH) Session**
  The Initiating Host (IH) Session is the period of time that a particular IH is connected to a Controller.

- **Initiating Host (IH) Session ID**
  256-bit randomized NONCE managed by the Controller and used to refer to a particular IH Session.

- **Mux ID** (MID)
  The 64-bit Mux ID (MID) is used to multiplex connections across a single IH-AH Tunnel in Dynamic Tunnel Mode. The most significant 32 bits form a unique value assigned by the Controller for each remote Service. It is referred to as the Service ID of the MID. The least significant 32 bits form a value maintained by the IH and the AH to differentiate among different TCP connections for a specific remote Service. This is referred to as the Session ID of the MID.

- **Service**
  A Service is an application and its associated data that is protected by the AH.

- **Service ID**
  The Service ID is the most significant 32 bits of the Mux ID.

- **Session ID**
  The Session ID is the least significant 32 bits of the Mux ID.

- **Single Packet Authorization One Time Password (SPA OTP)**

A Single Packet Authorization based on RFC 4226 but modified to include the counter value (see below). It is used to uniquely identify the IH when initiating communication to both the Controller and the AH.

5 SDP Protocol

The SDP protocol is explained below. The description follows the Architecture of the Software Defined Perimeter diagramed in Figure 3.

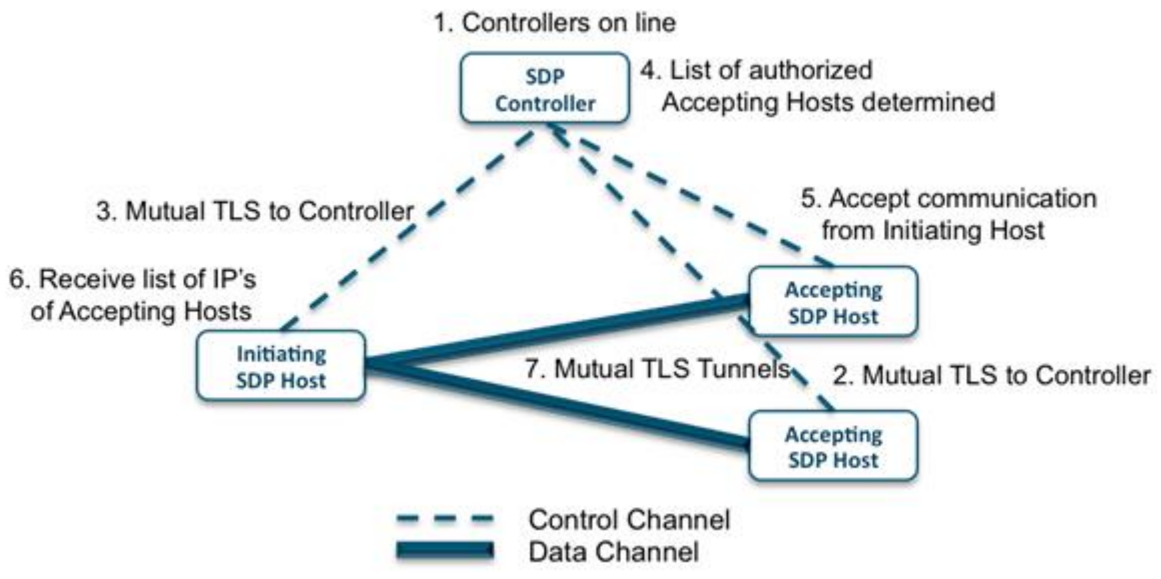
1. Controllers on line
SDP Controller
4. List of authorized Accepting Hosts determined
3. Mutual TLS to Controller
5. Accept communication from Initiating Host
6. Receive list of IP's of Accepting Hosts
Accepting SDP Host
Initiating SDP Host
7. Mutual TLS Tunnels
2. Mutual TLS to Controller
Accepting SDP Host
Control Channel
Data Channel

Figure 3: Architecture of the Software Defined Perimeter

5.1 Onboarding

The onboarding of one or more SDP Controllers, IHs, and AHs is outside the initial protocol specification. Typical methods would be via Chef or Puppet or their hosting service equivalents (e.g., RightScale, AWS CloudFormation, etc.).

5.2 Single Packet Authorization (SPA)

Single Packet Authorization (SPA) is used to initiate communication in all of the following: IH-Controller, AH-Controller, and IH-AH. SPA provides the following security benefits to the SPA-protected server.

- **Blackens the server**: The server will not respond to any connections from any clients until they have provided an authentic SPA.

- **Mitigates Denial of Service attacks on TLS**: Internet-facing servers running the https protocol are highly susceptible to Denial-of-Service (DoS) attacks. SPA mitigates these attacks because it allows the server to discard the TLS DoS attempt before entering the TLS handshake.
- **Attack detection**: The first packet to an AH from any other host must be an SPA. If an AH receives any other packet, it should be viewed as an attack. Therefore, the SPA enables the SDP to determine an attack based on a single malicious packet.

SPA is based on RFC 4226 (HOTP) with the following parameters:

- **Client**: RFC 4226 uses the term "client" to refer to the generator of the SPA packet. In the SDP architecture, the client is either the IH or the AH.
- **Server**: RFC 4226 uses the term "server" to refer to the authenticator of the SPA packet In the SDP architecture, the server is either the Controller or the AH.
- **Seed**: The seed is a 32-bit unsigned integer shared between each communicating pair (i.e., IH-Controller, AH-Controller, and IH-AH). The seed must be kept secret.
- **Counter**: The counter is a 64-bit unsigned integer that must be synchronized between communicating pairs. In RFC 4226, this is done via a "look-ahead window" (because the typical use case for RFC 4226 is a hardware OTP token). However, for the SDP protocol, the counter can be sent in the SDP packet obviating the need for a look-ahead window and the potential for the communicating pair to be out of sync. Note that the counter does not need to be kept secret.
- **Password**: The HOTP value generated by the RFC 4226 algorithm.
- **Password Length**: The password length is fixed to 8 digits.

For the SPA protocol, a single SPA packet is sent from the client to the server. The server does not reply. The format of the packet is:

IP	TCP	AID (32-bit)	Password (32-bit)	Counter (64-bit)

After receiving the packet, the server must enable the client to connect via mutual TLS on port 443.

5.3 Mutual TLS or IKE

The connections between all hosts must use TLS or Internet Key Exchange (IKE) with mutual authentication to validate the device as an authorized member of the SDP prior to further device validation and/or user authentication. All weak cipher suites and all suites that do not support mutual authentication must be disallowed.

The root certificate for both the TLS (IPsec) client and server must be pinned to a known valid root and should not consist of the hundreds of root certificates trusted by most consumer browsers. This mitigates impersonation attacks whereby an attacker can forge a certificate from a compromised certificate authority. The methodology by which the root certificate is on boarded to the IH, AH, and Controller is outside the initial protocol specification. Typical methods would be via Chef or Puppet or their hosting service equivalents (e.g., RightScale, AWS CloudFormation, etc.).

The TLS (IPsec) server shall use OCSP response stapling as defined by the IETF working draft “X.509v3 Extension: OCSP Stapling Required draft-hallambaker-muststaple-00”, which references the stapling implementation in RFC 4366 “Transport Layer Security (TLS) Extensions”. OCSP response stapling mitigates DoS attacks on the OCSP responders and also mitigates man-in-the-middle attacks using obsolete OCSP responses from before the server certificate was revoked.

5.4 Device Validation

Mutual TLS (IKE) proves that the device requesting access to the SDP possesses a private key that has not expired and that has not been revoked, but it does not prove that the key has not been stolen. The objective of Device Validation is to prove that the proper device holds the private key and that the software running on the device can be trusted. In an SDP, the Controller is assumed to be a trusted device (because it exists in the most controlled environment) and the IHs and AHs must validate to it. Device Validation mitigates credential theft and the resultant impersonation attacks. Device validation is beyond the scope of this version of the SDP, but should be addressed in future versions.

5.5 AH-Controller Protocol

The following subsections define the various messages and their formats that are passed between the AH and the Controller. The basic protocol is of the form:

`Command (8-bit)`	`Command specific data (command specific length)`

5.5.1 Login Request Message

The login request message is sent by the AH to the Controller to indicate that it is available and is able to accept other messages from the Controller.

`0x00`	`No command specific data`

5.5.2 Login Response Message

The login response message is sent by the Controller to indicate whether the login request was successful and, if successful, to provide the AH Session ID.

0x01	Status Code (16 bits)	AH Session ID (256 bits)

5.5.3 Logout Request Message

The logout request message is sent by the AH to the Controller to indicate that it is no longer available and is not able to accept other messages from the Controller. There is no response.

0x02	No command specific data

5.5.4 Keep-Alive Message

The Keep-Alive message is sent by either the AH or the Controller to indicate that it is still active.

0x03	No command specific data

5.5.5 AH Services Message

The services message is sent by the Controller to indicate to the AH that set of services that this AH is protecting

0x04	JSON formatted array of Services

The JSON specification is:

Format	Example

{“services”:
 [“port”: <Server port>,
  “id”: <32-bit Service ID>,
  “address”: <Server IP>,
  “name”: <service name>
 ]
}
	{“services”:
	 [“port”: “443”,
	  “id”: “123445678”,
	  “address”: “100.100.100.100”,
	  “name”: “SharePoint”
	]
	}

5.5.6 IH Authenticated Message

The IH Authenticated Message is sent by the Controller to the AH to indicate to the AH that a new IH has been validated and that the AH should allow access to this IH for the specified services.

`0x05`	`JSON formatted array of IH information`

The JSON specification is:

```
{“sid”:  <256-bit IH Session ID>,

 “seed”: <32-bit SPA seed>,

 “counter”: <32-bit SPA counter>

 [“id”:<32-bit Service ID>

 ]

}
```

5.5.7 Reserved for Private Use

This command (0xff) is reserved for any non-standard messages between the AH and the Controller.

`0xff`	`User specified`

5.6 AH to Controller Sequence Diagram

The sequence diagram for the protocol for the AH to connect to the Controller is shown on the next page. This sequence diagram only describes the message sequence associated with the initial login. The messages sent when an IH connects to a Controller are shown in the IH-Controller Sequence Diagram.

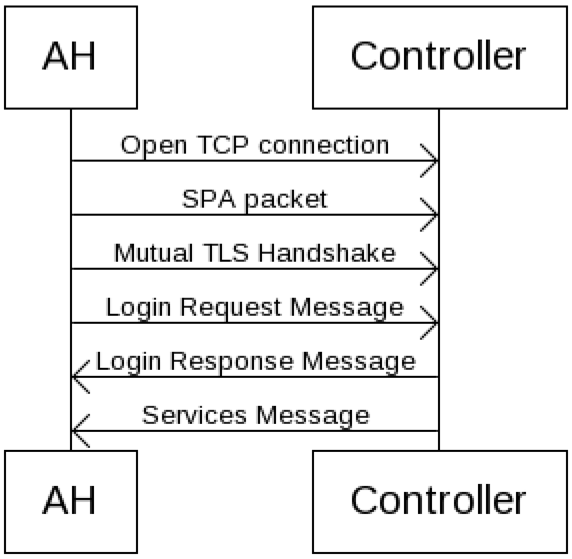

Figure 4: Accepting Host connects to the Controller

5.7 IH-Controller Protocol

The following subsections define the various messages and their formats that are passed between the IH and the Controller. The basic protocol is of the form:

`Command (8-bit)`	`Command specific data of command specific length`

5.7.1 Login Request Message

The login request message is sent by the IH to the Controller to indicate that it is available and would like to be part of the SDP.

`0x00`	`No command specific data`

5.7.2 Login Response Message

The login response message is sent by the Controller to indicate whether the login request was successful and, if successful, to provide the IH Session ID.

`0x01`	`Status Code (16 bits)`	`IH Session ID (256 bits)`

5.7.3 Logout Request Message

The logout request message is sent by the IH to the Controller to indicate that it no longer wishes to be part of the SDP. There is no response.

0x02	No command specific data

5.7.4 Keep-Alive Message

The Keep-Alive message is sent by either the IH or the Controller to indicate that it is still active.

0x03	No command specific data

5.7.5 IH Services Message

The services message is sent by the Controller to indicate to the IH the list of available services and the IP addresses of the AHs protecting them.

0x06	JSON formatted array of Services

The JSON specification is:

Format	Example

{"services": [
  {"address" : <AH IP>,
   "id": <32-bit Service ID>,
   "name": <service name>,
   "type" : <service type>[1] }
   ]
 }
	{"services": [
	  {"address" : "200.200.200.200",
	   "id": "12345678",
	   "name": "SharePoint",
	   "type" : "https" }
	   ]
	 }

[1] The type is used to differentiate the way that the services are reach. For example, http could be a service type. Https could be a service type.

5.7.6 Reserved for Private Use

This command (0xff) is reserved for any non-standard messages between the IH and the Controller.

`0xff`	`User specified`

5.8 IH to Controller Sequence Diagram

The sequence diagram for the protocol for the AH to connect to the Controller is shown below.

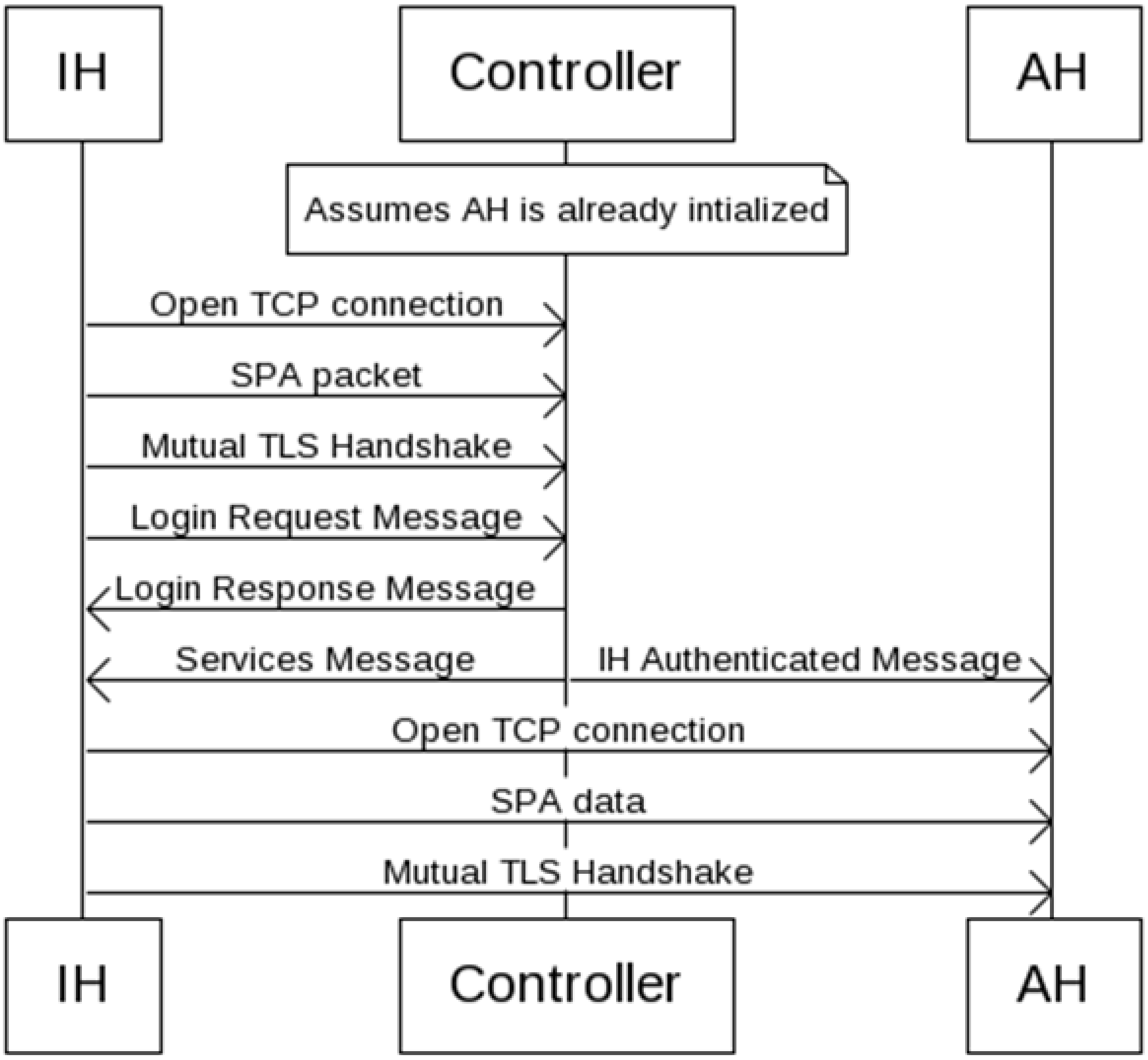

Figure 5: Initiating Host connects to the Controller and an Accepting Host

5.9 IH-AH Protocol in Dynamic Tunnel Mode (DTM)

The following subsections define the various messages and their formats that are passed between the IH and the AH in DTM mode. The basic protocol is of the form:

`Command (8-bit)`	`Command specific data of command specific length`

5.9.1 Keep-Alive Message

The Keep-Alive message is sent by either the IH or the AH to indicate that it is still active.

`0x03`	`No command specific data`

5.9.2 Open Connection Request Message

The open request message is sent by the IH to the AH to indicate that it would like to open a connection to a particular Service.

`0x07`	`Mux ID (64 bits)`

5.9.3 Open Connection Response Message

The open response message is sent by the AH to the IH to indicate whether the open request was successful.

`0x08`	`Status Code (16 bits)`	`Mux ID (64 bits)`

5.9.4 Data Message

The data message is sent by either the IH or the AH. It is used to push data on an open connection. There is no response.

`0x09`	`Data Length (16 bits)`	`Mux ID (64 bits)`

5.9.5 Connection Closed Message

The connection closed message is sent by either the IH or the AH. It is used to either indicate that a connection has been closed by the AH or that the IH is requesting a connection be closed. There is no response.

`0x0A`	`Mux ID (64 bits)`

5.9.6 User-Defined Message

This command (0xff) is reserved for any non-standard messages between the IH and the Controller.

`0xff`	`User specified`

5.10 IH to AH Sequence Diagram (Sample)

A sample sequence diagram for the protocol between the IH and the AH is shown below. This sequence diagram only describes the message sequence associated with the initial login. The messages sent when an IH connects to a Controller are shown in the AH-Controller Sequence Diagram.

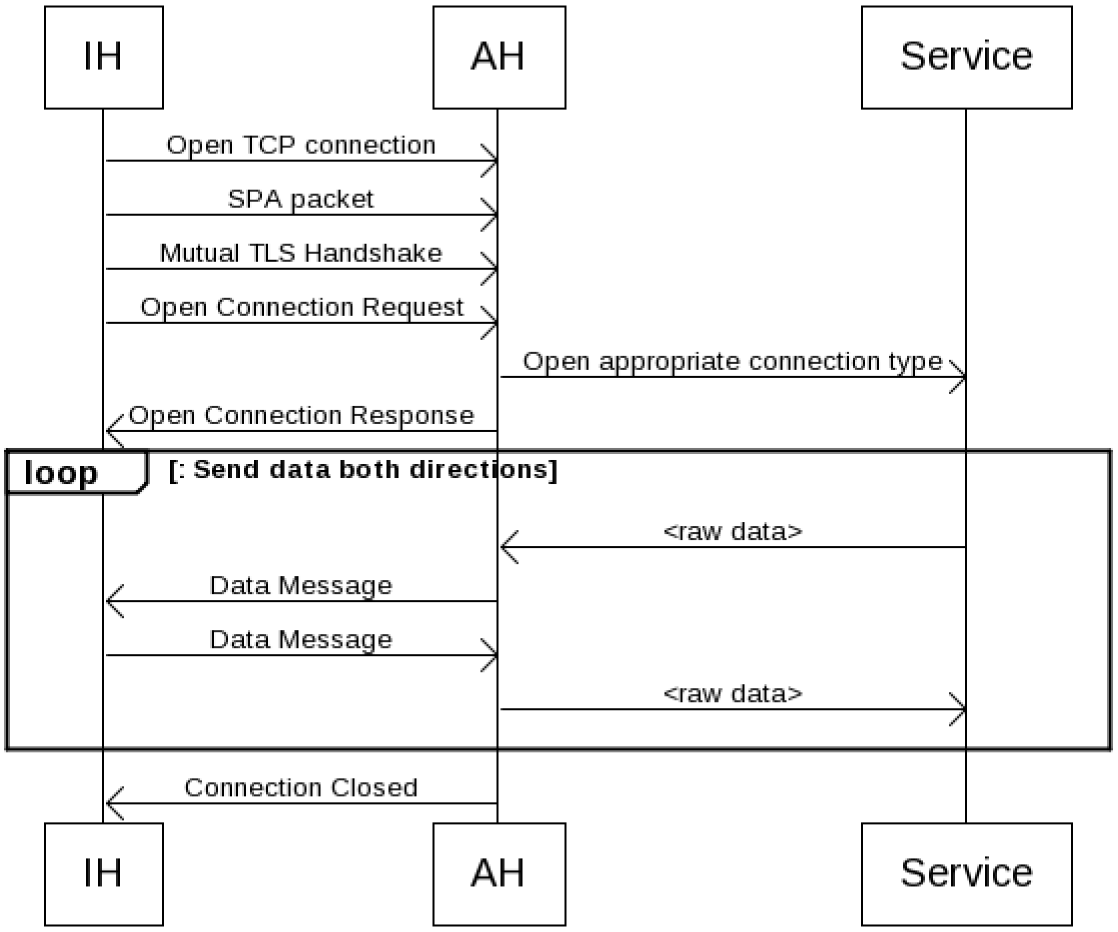

Figure 5: IH connects to an AH and then sends data to a Service

5.11 Controller determines the list of AHs to which the IH connects

The methodology by which the Controller determines the list of AHs to which the IH connects is outside the initial protocol specification. If this is an Internet of Things application, the list may be static or be based on the type of device connecting to the Software Defined Perimeter. If this is a Server-to-Server application, the list may come from a protected database server. If this is a Client/Server application the list may come from an LDAP server. Other applications may determine the list in other ways.

6 Logging

Creating logs to determine availability and performance of services and the security of the server is a requirement of all systems.

6.1 Fields of a log message

All logs will have the following fields.

Field Name	**Meaning**
`time`	time at which log record was generated
`name`	human readable name for the event. Note: do not include any mutable data nuggets, such as usernames, ip addresses, hostnames, etc. That information is contained in the additional fields of the log record already and we don’t want to repeat them.
`severity`	a severity for this event ranging from debug to critical (see below)
`deviceAddress`	the ip address of the machine generating the log record

6.2 Operations

The following is a list of operational use cases or activities that need to be logged.

The signature_id is an identifier, making it possible to identify the type of event. The third column contains the additional fields that need to be logged for the specific log messages.

activity	signature_id	data/information to log
component startup, shutdown, restart (e.g., Controller coming up, Host restart)	`ops:startup` `ops:shutdown` `ops:restart`	***reason*** indicating why a restart or shutdown has occurred ***component*** indicating what component was affected
connection between components (Controller, IH, AH, 3rd party component, DB) up, down, reconnect	`ops:conn:up` `ops:conn:down` `ops:conn:reconnect`	***src*** origin of the connection as an ip address as seen by the reporting entity ***dst*** destination of the connection as an ip address as seen by the reporting entity ***reconnect_count*** how many times reconnect was attempted ***reason*** indicating why the communication went **down**

Here is a quick scenario that outlines a complete outage, showing what log entries are written where. In this scenario, we assume that a Controller goes down:

1. Controller goes down [no log, a failing component is not able to log]
2. IH tries to reconnect to Controller n times
   ops:conn:reconnect log messages
3. After n times, the client declares the connection to the Controller to be down and it looks for a new Controller
   ops:conn:down log message, with a severity of *error*
4. IH connects to the newly found Controller
   ops:conn:up log message
5. If no more Controllers are available
   ops:conn:down log message, with a severity of *critical*

Another similar case would be a client that goes down without warning (e.g., laptop being closed). In that case, the Controller would detect a failing connection, as well as the AH. Each of them would log a

ops:conn:down log message, with a severity of *error*

6.3 Security

The security logs are core to the SDP and are also going to be important in a broader context to detect larger scale infrastructure attacks. Therefore, these logs are very useful if forwarded to a SIEM product.

The signature_id is an identifier, making it possible to identify the type of event. The third column contains the additional fields that need to be logged for the specific log messages.

activity	**signature_id**	**data/information to log**
AH login success	*sec:login*	**src** the IP address of the AH as seen by the Controller **AH Session ID** the session ID of the AH
AH login failure		
IH login success	*sec:login*	**src** the IP address of the IH as seen by the Controller **IH Session ID** the session ID of the IH
IH login failure		
component authentication (e.g., IH -> Controller)		
denied inbound connection	*sec:fw:denied*	***src*** source of the attempted connection ***dst*** destination of the attempted connection

Here is an example of how a complete user login (IH connecting to AH) looks:

1. IH connects to Controller
   ops:conn:up log message
2. IH mutually authenticates to Controller
   sec:auth log message

3. IH connects to AH
   ops:conn:up log message

4. IH mutually authenticates to AH
   sec:auth log message

6.4 Performance

Performance information generally doesn't lend itself too well to be logged the traditional way. A lot of metrics would overwhelm the logging system and the analysis processes are not built to deal with this kind of information. Therefore, we recommend a separate way of dealing with performance logging.

6.5 Compliance

All compliance mandates, such as PCI or SOX are very vague when it comes to logging guidelines. SOX, for example, requires that one record all privileged access to financial systems or any activity that could have an impact on the financial statements/results. The way we covered the 'security' logging use cases, we are already covering the logging use cases for compliance.

6.6 Security Information and Event Management (SIEM) integration

We recommend forwarding all the security events to a SIEM. This will help the SIEM to generate an overall picture of the security posture of the network. Having the SDP security logs as part of the picture will greatly enhance the visibility and understanding of the environment.

The operational log records should be used in-product to manage the availability and performance of the product. This information is not very useful in a broader context, outside of the SDP. However, we recommend that there is a way the user can specify which logs to forward to a central console (i.e, SIEM). That way, if a user sees benefit in having any other type of information available, they have a way to do so.

7 SDP Specification

This document specifies a Cloud Security Alliance (CSA) initiated protocol for Software Defined Perimeter specification. Distribution of this memo is unlimited.