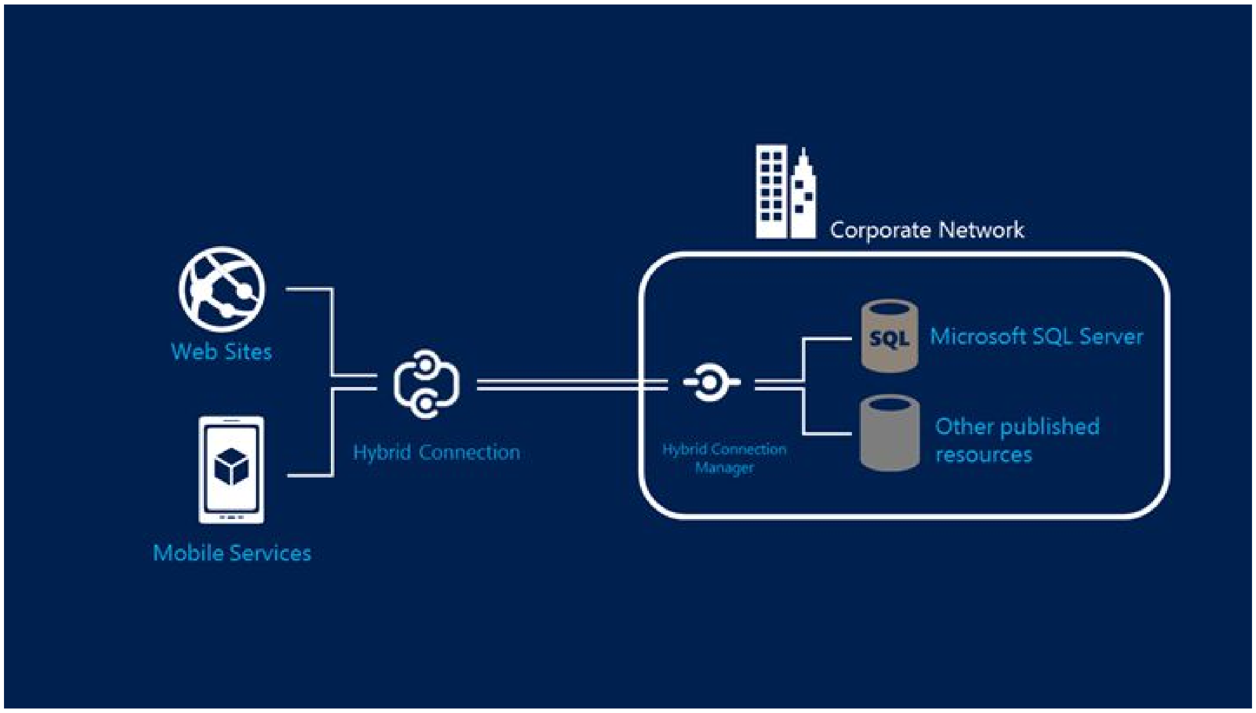
Azure Hybrid Connection Security Review

Conducted by: David Kane-Parry, 18F Security Lead
August 3, 2017

Introduction

Microsoft describes Azure Hybrid Connections (AHCs) as “an easy and convenient way” to connect apps running in the Azure App Service to on-premises resources behind a firewall.



AHCs are easily understood as an outer proxy (Hybrid Connection) and an inner proxy (Hybrid Connection Manager). The outer proxy forwards traffic to the inner proxy and returns traffic to the app; the inner proxy forwards traffic to the otherwise-firewalled resources and returns traffic to the outer proxy. As the inner proxy initiates the connection to the outer proxy, the firewall does not need to be configured to allow externally-initiated connections. This is an attractive feature for organizations using Azure with on-premise resources that need to be Internet-accessible, but that are unable to leverage a “zero-trust” network with direct connections between apps and resources.

For an organization to move from a position of “no Internet access for these resources” to “*some* Internet access for these resources,” the proposition of having that access mediated by a third party might give some pause. For the purposes of this review, since AHCs are only available to Azure-based apps, it will be practical to assume that Microsoft is a trusted third-party. Indeed, the Azure Government environment has been granted a P-ATO at the High Impact Level by the Joint Authorization Board. It will also be practical to be assume that the apps and resources are not vulnerable in any way other than possibly through their dependence on AHCs. This review is focused on the security of AHCs only, independent of any risk that might emerge for a particular app or resource when it integrates with AHCs.

Transport Layer Security

Confidential and tamper-evident connections between apps and the outer proxy, and between the outer proxy and the the inner proxy, are secured using TLS 1.2. Since the AHC endpoint of the outer proxy is only available on a per AHC basis, no endpoint was available for testing, but one can reasonably assume that the TLS configuration of an AHC endpoint is identical with the TLS configuration of portal.azure.com. Based on the SSL Labs report for portal.azure.com, we can conclude that the AHC endpoints are probably even stronger since they do not support earlier flawed versions of TLS. In addition, because Microsoft controls the TLS client at both ends, they are in a position to disable all but the strongest of the cipher suites supported in TLS 1.2.

Authentication

The outer proxy authenticates itself to the app and the inner proxy with its X.509 server certificate. Its server certificate is issued by Microsoft's own intermediate certificate authority, which is itself signed by the Baltimore CyberTrust Root certificate authority operated by DigiCert. Because Microsoft controls the TLS client at both ends, there is no requirement to trust the root certificate authority, and so an attacker who compromises the root certificate authority will not necessarily be able to spoof the outer proxy to any TLS clients. TLS clients can safely pin their trust to the intermediate certificate authority that Microsoft owns, and so only an attacker who could compromise Microsoft's authority could spoof the outer proxy. As documented by their independent auditor, compromising Microsoft's authority would be rather difficult.

Apps and the inner proxy authenticate themselves to the outer proxy by means of what Microsoft calls a Shared Access Signature (SAS). For each AHC, up to twelve policies may be defined. Each policy is scoped to a name, a cryptographically random primary and secondary key, and access rights (to listen, send, and/or manage). When the app or inner proxy has a request to send to the outer proxy, elements of the request are signed by combining them with one of the keys to produce an hash-based message authentication code (HMAC). This is the SAS. The outer proxy authenticates the request by combining the same elements from the request with the same key it has that policy to produce its own HMAC and compares the HMACs for equality. The HMACs will only be identical when the client knows the key that was issued for the given policy name. It is computationally infeasible to either brute-force the key or to reverse the key, should an attacker successfully intercept an HMAC. In addition, because one of the signed elements of the request is a timestamp, an attacker who can successfully intercept an HMAC has a limited window of opportunity to replay the request.

Please note that this authentication protocol is only for establishing the TCP tunnel between the app and the firewalled resource. The firewalled resource may, and possibly should, enforce its own authentication protocol. For example, the use of an AHC does not obviate the need to enforce SQL Server authentication via username and password.

Authorization

As hinted at in the previous section, apps and the inner proxy are authorized by means of access rights that are defined in the policies that are tied to their keys. An app or inner proxy that requests access to a resource not listed in its access rights will be denied, regardless of whether the HMACs are authentic. The complexity of the access rights that can be managed for AHC participants is outside the scope of this review; suffice it to say that clients do not have any rights by default, but all rights must be assigned to them, and clients may not elevate their rights except by obtaining keys for policies that have more rights granted than intended for the client. Rights are easily removed at any time, without requiring redistribution of new keys. It is easy to rotate keys, because the secondary key is simply replaced with the primary key, and a new primary key is created, enabling the outer proxy to continue to authenticate requests that might have been made with the previous primary key during the rotation.

Logging

To be able to continuously validate that the AHC is performing as expected, logging is essential as a feedback mechanism, and not just for forensics in service of incident response. In a less restricted environment, logging on the on-premise resource’s host would provide all of the information necessary to detect undesirable or unauthorized behavior. In an AHC model, from the perspective of the on-premise resource, there is only one client and it is the inner proxy, and all external communication is secured such that traffic inspection by a middlebox would require intentional sabotage of TLS. To gain a greater level of detail, the inner proxy and the outer proxy must both be queried directly.

The inner proxy has some logging enabled by default and can be inspected through the Event Viewer or other interfaces. Since the inner proxy uses the .NET framework to implement communication with the outer proxy, even more logging can be obtained by turning on diagnostics for the System.Net namespace. There does not appear to be any documentation regarding logs that are available from the outer proxy, so teams are advised to inspect if the level of logging provided by the outer proxy is acceptable for their mission. It is also possible that Microsoft has documentation regarding outer proxy logging that was not found in the course of this review, and one would need only ask for it to obtain it.

Conclusion

From all available documentation, Azure Hybrid Connections appear to be a safe and easy way to inch toward Internet access for firewalled resources. AHCs utilize standard security protocols to protect the confidentiality, integrity, and authenticity of data in transit, as well as best practices for authenticating and authorizing the participants. No technology is without risk, and while security ultimately is a responsibility shared between the app and on-premise resource, AHCs do what they need to do to provide a trustworthy tunnel between them.