Exchanging Demands

Peter Hannay
SECAU Security Resarch Center, Edith Cowan University
p.hannay@ecu.edu.au

Abstract

Smart phones and other portable devices are often used with Microsoft Exchange to allow users to check their corporate emails, sync their calendars remotely and perform other tasks. Exchange has an interesting relationship with its mobile clients, demanding a certain level of control over the devices, enforcing policy such as password complexity, screen timeouts, remote lock out and remote wipe functionality. The protocol for updating these policies provides very little in the way of security and is often silently accepted by the device. In this paper we will discuss on the remote wipe functionality and how a potential attacker could abuse this functionality to remotely wipe devices that are connected to Exchange.

Keywords
Network security, network, security, exchange, protocol attack, wipe, activesync

INTRODUCTION

Microsoft Exchange has a unique relationship with its mobile clients, demanding the ability to control policy on the device, such as password complexity, encryption, screen timeout, etc. These policies are pushed through ActiveSync at the time of account creation on the device. The user is given the option to accept these or decline, however declining the policies will render the account inaccessible from the device. As such all devices connected to an Exchange server have this policy agreement in place.

In this paper we explore the mechanism through which a policy push is carried out and demonstrate how a malicious actor can leverage the same to remotely erase a device.

THE EXCHANGE

A policy push can occur at any point when the device makes a request to the ActiveSync server. The push is comprised of two distinct phases, firstly an error is provided indicating that a new policy has been issued, forcing the device to request a policy update prior to any other action, secondly the device requests and accepts the new policy.

Phase 1: Provision Demand

When the client makes a request to the Exchange server (via ActiveSync) a specific request is passed, when there is a policy push to be made the server will issue a HTTP error 449, refusing the action until the device has accepted a policy update. This conversation is shown in figure 1 below.

```
POST /Microsoft-Server-ActiveSync?Cmd=...........&DeviceType=Android HTTP/1.1
Content-Type: application/vnd.ms-sync.wbxml
Authorization: Basic cGFzc3dvcmQgZ29lcyBoZXJl
MS-ASProtocolVersion: 12.0
Connection: keep-alive
User-Agent: Android/0.3
X-MS-PolicyKey: 358347207
Content-Length: 13
Host: 192.168.1.218

HTTP/1.1 449 Retry after sending a PROVISION command
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/7.5
MS-Server-ActiveSync: 14.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Tue, 08 May 2012 07:08:22 GMT
Content-Length: 54
The custom error module does not recognize this error.
```

Figure 1 – HTTP Error 449

Phase 2: Policy Push

On receipt of HTTP error 449 the client should make a policy provision request to the server. The server then responds with a policy update encoded as ActiveSync WAP Binary XML (WBXML). A sample of the request/response is shown below in figure 2.

```
POST /Microsoft-Server-ActiveSync?Cmd=Provision&User=........&DeviceType=Android HTTP/1.1
Content-Type: application/vnd.ms-sync.wbxml
Authorization: Basic cGFzc3dvcmQgZ29lcyBoZXJl
MS-ASProtocolVersion: 12.0
Connection: keep-alive
User-Agent: Android/0.3
X-MS-PolicyKey: 0
Content-Length: 41

Host: 192.168.1.218
..j...EFGH.MS-EAS-Provisioning-WBXML.....

HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/vnd.ms-sync.wbxml
Server: Microsoft-IIS/7.5
MS-Server-ActiveSync: 14.0
Date: Tue, 08 May 2012 07:00:04 GMT
Content-Length: 123

..j...EK.1..FGH.MS-EAS-Provisioning-WBXML..K.1..I.2761868790..JMN.0..O.0..Q.0..P.0..S.1..T.4..U.900..
V.8...X.1...Z.0.......
```

Figure 2 – Provision Request/Response

The response from the server is binary encoded XML, which can be decoded as per the published specification (MS-ASWBXML). If this is decoded it appears as shown below in Figure 3, this is an approximation however as there exists no direct or specific way to decode this into readable XML.

```
<Provision>
   <Status>1</Status>
   <Policies>
      <Policy>
         <PolicyType>MS-EAS-Provisioning-WBXML</PolicyType>
         <Status>1</Status>
         <PolicyKey>2761868790</PolicyKey>
         <Data>
            <EASProvisionDoc>
               <DevicePasswordEnabled>0</DevicePasswordEnabled>
               <AlphanumericDevicePasswordRequired>0</AlphanumericDevicePasswordRequired>
               <PasswordRecoveryEnabled>0</PasswordRecoveryEnabled>
               <DeviceEncryptionEnabled>0</DeviceEncryptionEnabled>
               <AttachmentsEnabled>1</AttachmentsEnabled>
               <MinDevicePasswordLength>4</MinDevicePasswordLength>
               <MaxInactivityTimeDeviceLock>900</MaxInactivityTimeDeviceLock>
               <MaxDevicePasswordFailedAttempts>8</MaxDevicePasswordFailedAttempts>
               <MaxAttachmentSize />
               <AllowSimpleDevicePassword>1</AllowSimpleDevicePassword>
               <DevicePasswordExpiration />
               <DevicePasswordHistory>0</DevicePasswordHistory>
            </EASProvisionDoc>
         </Data>
      </Policy>
   </Policies>
   <RemoteWipe />
</Provision>
```

Figure 3 – Decoded Policy Data

We can see the remote wipe command toward the end of the policy data. This command is interpreted by the device as an immediate instruction to begin self-erasure.

IMPLEMENTATION

The above exchange was implemented as a proof of concept in order to test the efficiency of this as an attack, as well as to identify any security measures implemented to prevent a rouge wipe command. The attack that we will discuss comprised two parts the first involves establishing a man in the middle condition through the use of a

hostile access point known as the pineapple. The second phase of the attack is to inject the command to initiate a remote device wipe.

Establishing Man in the Middle

The Wi-Fi pineapple is an off-the-shelf device manufactured by Hak5, for our purposes the device has been used its default configuration. In essence the Wi-Fi pineapple listens for probe sent by prospective Wi-Fi clients searching for remembered networks. On receipt of these probes the device broadcasts an SSID inviting the client to connect. For our purposes the device has been configured to forward this went all connection attempts to a system running the PoC script (covered later in this document).

Initiating the Wipe

The vast majority of exchange and mobile device deployments make use of SSL to implement some level of security. In order to successfully accept the connection from the device we need to negotiate an SSL handshake. It is assumed that we do not possess the private key for the exchange server to which our victim is attempting a connection as such we will be making use of a self signed certificate. The details of the certificates will not match those of the intended server, a “one size fits all” invalid certificate will be used to attack all test devices.

Our PoC script will listen for connections from the victim device and accept those connections. Upon connection it will check to see if the client has issued a provisioning request, if it has done so the wipe command will be issued. If the provisioning request has not been sent, a HTTP 449 error will be issued requesting provisioning.

Testing

In testing we evaluated Android, iOS & Windows Phone 7.5. We set up two exchange servers, both running Exchange Server 2010. The first server made use of a self-signed certificate, the second made use of a certificate signed by a valid CA. The attack was then conducted to issue remote wipe commands with a third, self signed certificate.

Each device was tested twice, once with each exchange server. The devices were then connected to the network with the PoC script running and results recorded. This allows us to determine the efficacy of the attack on clients who use servers with either self-signed certificates or valid root signed certificates.

Results

The following results (see table 1) were attained from the testing process.

Device Tested	**Self-Signed Cert**	**Trusted Cert**
Android (2.3 & 4.0)	Wiped (no user interaction)	Not wiped (security error displayed)
iOS 5	Wiped (w/ new certificate warning)	Wiped (w/ new certificate warning)
Windows Phone 7.5	Not wiped (cert error displayed)	Not wiped (cert error displayed)

Table 1 – Results of Testing

As you can see from the results above, clients of exchange servers making use of self-signed certificates (our research indicates this is the most common deployment style for small to medium businesses) are most vulnerable, with associated handsets being subject to remote wipe without prompt for android handhelds, and with a certificate error prompt for iOS devices. In the case of iOS devices the prompt displayed was for a certificate error, providing no advice with a clearly available "continue" button. Windows Phone 7.5 provided no mechanism to easily accept a self-signed certificate (it had to be installed manually), when the certificate changed there was no easy mechanism to accept the new certificate.

Clients of exchange servers using certificates signed by a Trusted CA fared somewhat better, with Android & Windows Phone devices simply refusing to connect. Android cited a security error, whilst windows phone cited a certificate error. There was no mechanism provided to continue connecting regardless in either case. The iOS devices tested provided a prompt to accept the new certificate, again with no advice and an easily available continue button. It can be seen that the Windows Phone devices fared best against this issue. Showing an error in all cases, refusing to continue with the connection.

CONCLUSION

In summary the issue highlighted is primarily one of authenticity, rather than an issue in the ActiveSync protocol itself. In this case security has been delegated to certificate handling routines, which have proven to be inadequate in many situations. A remote wipe isn't the only functionality possible, stealing credentials would be possible, as would intercepting messages, forcing data sync, etc.