CVE-2021-26855: Microsoft Exchange Server-Side Request Forgery

googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2021/CVE-2021-26855.html

Google Project Zero

Anthony Weems, Michael Weber, Dallas Kaman

The Basics

Disclosure or Patch Date: March 2 2021

Product: Microsoft Microsoft Exchange Server

Advisory: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-26855

Affected Versions: Exchange 2010, 2013, 2016, and 2019 before KB5000871.

First Patched Version: KB5000871

Issue/Bug Report: N/A

Patch CL: N/A

Bug-Introducing CL: N/A

Reporter(s): Volexity, Orange Tsai from DEVCORE research team, and Microsoft Threat Intelligence Center (MSTIC)

The Code

Proof-of-concept: https://github.com/praetorian-inc/proxylogon-exploit

Exploit sample: N/A

Did you have access to the exploit sample when doing the analysis? No

The Vulnerability

Bug class: Server-Side Request Forgery (SSRF)

Vulnerability details:

Note: This analysis relies upon source code obtained by decompiling the various .NET assemblies within Microsoft Exchange 2013.

The Exchange frontend proxy is tricked into sending a request to an arbitrary backend endpoint authenticated via Kerberos as the Exchange server.

The `BEResourceRequestHandler` is used to handle requests for static resources within `/ecp` that pass the `IsResourceRequest` check. This function validates that the provided URL ends with one of many static file extensions (e.g. js, jpg, ico, png, ttf, etc.). Since this handler is within the frontend, it does not validate the full static file path and therefore a random filename with a `.js` suffix will be sent to this handler.

Requests in the frontend proxy are routed to the backend via "anchor mailboxes" (the `AnchoredRoutingTarget` field within `ProxyRequestHandler`). Each resource handler is responsible for returning an anchor mailbox to describe how its request should be routed. The `BEResourceRequestHandler` uses the following pseudo-code for its routing:

```
protected override AnchorMailbox ResolveAnchorMailbox()
{
    string cookie = BEResourceRequestHandler.GetBEResouceCookie(base.ClientRequest);
    if (!string.IsNullOrEmpty(cookie))
    {
        return new ServerInfoAnchorMailbox(BackEndServer.FromString(cookie), this);
    }
    return base.ResolveAnchorMailbox();
}

private static string GetBEResouceCookie(HttpRequest httpRequest)
{
    string result = null;
    HttpCookie httpCookie = httpRequest.Cookies[Constants.BEResource];
    if (httpCookie != null)
    {
        result = httpCookie.Value;
    }
    return result;
}
```

The `BEResourceRequestHandler` uses the `X-BEResource` cookie to construct a `BackEndServer` and then a `ServerInfoAnchorMailbox`. Pseudo-code for `BackEndServer` instantiation is shown below:

```
public static BackEndServer FromString(string input)
{
    string[] array = input.Split(new char[]{'~'});
    int version;
    if (array.Length != 2 || !int.TryParse(array[1], out version))
    {
        throw new ArgumentException("Invalid input value", "input");
    }
    return new BackEndServer(array[0], version);
}
```

Finally, this anchor mailbox is used within the `ProxyRequestHandler`'s `GetTargetBackEndServerUrl` function to resolve the actual `Uri` to use in the backend request. Pseudo-code for this method is shown below:

```
protected virtual Uri GetTargetBackEndServerUrl()
{
    // ...
    UriBuilder clientUrlForProxy = new UriBuilder(this.ClientRequest.Url);
    clientUrlForProxy.Scheme = Uri.UriSchemeHttps;
    clientUrlForProxy.Host = this.AnchoredRoutingTarget.BackEndServer.Fqdn;
    clientUrlForProxy.Port = 444;
    if (this.AnchoredRoutingTarget.BackEndServer.Version < Server.E15MinVersion)
    {
        this.ProxyToDownLevel = true;
        clientUrlForProxy.Port = 443;
    }
    return clientUrlForProxy.Uri;
}
```

The `UriBuilder` implementation in .NET uses simple string concatenation when building the `Uri` string in the last line of `GetTargetBackEndServerUrl`. For example, if the `Host` is set to `example.local/endpoint#`, then the resulting Uri from this method call might be `https://example.local:443/endpoint#/ecp/favicon.eco`. If the `Host` header contains a `:` (e.g. to change the destination port to `444`), the `UriBuilder` class assumes the host must be an IPv6 address and surrounds it with `[]`. To resolve this issue, the desired host must be prefixed with an `@` symbol, which causes the leading `[` to be treated as a username.

It seems likely that this vulnerability arose due to incorrect assumptions about `UriBuilder` validation.

Patch analysis:

The patch adds hostname validation in two locations:

```
---
a/Microsoft.Exchange.Data.ApplicationLogic/Exchange/Data/ApplicationLogic/Cafe/BackEnd

+++
b/Microsoft.Exchange.Data.ApplicationLogic/Exchange/Data/ApplicationLogic/Cafe/BackEnd

@@ -34,7 +34,7 @@ namespace Microsoft.Exchange.Data.ApplicationLogic.Cafe
         '~'
     });
     int version;
-    if (array.Length != 2 || !int.TryParse(array[1], out version))
+    if (array.Length != 2 || !int.TryParse(array[1], out version) ||
UriHostNameType.Dns != Uri.CheckHostName(array[0]))
     {
         throw new ArgumentException("Invalid input value", "input");
     }
---
a/Exchange2013/Microsoft.Exchange.FrontEndHttpProxy/HttpProxy/ProxyRequestHandler.cs
+++
b/Exchange2013/Microsoft.Exchange.FrontEndHttpProxy/HttpProxy/ProxyRequestHandler.cs
@@ -923,7 +923,10 @@ namespace Microsoft.Exchange.HttpProxy
     try
     {
         Uri uri = this.GetTargetBackEndServerUrl();
-        bool proxyKerberosAuthentication = this.ProxyKerberosAuthentication;
+        if (!this.ProxyKerberosAuthentication && !string.Equals(uri.Host,
this.AnchoredRoutingTarget.BackEndServer.Fqdn, StringComparison.OrdinalIgnoreCase))
+        {
+            throw new HttpException(503, "Service Unavailable");
+        }
         bool flag2 = false;
```

Additionally, the patch overrides `ShouldBackendRequestBeAnonymous` in `BEResourceRequestHandler` to return true.

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

It seems plausible that this vulnerability was found through code auditing of the frontend proxy and reviewing connections between the frontend and backend.

(Historical/present/future) context of bug:

In March 2021, Microsoft published that "multiple 0-day exploits [were] being used to attack on-premises versions of Microsoft Exchange Server in limited and targeted attacks". Microsoft credited Volexity for discovering the active exploitation and Volexity published their analysis on the same day.

The Exploit

(The terms *exploit primitive*, *exploit strategy*, *exploit technique*, and *exploit flow* are defined here.)

Exploit strategy (or strategies):

The SSRF allows an attacker to submit arbitrary requests to backend `/ecp` endpoints. The frontend proxy authenticates to the backend via Kerberos as the Exchange server. However, this user is unlikely to have access provisioned for the application itself. For example, when attempting to access the DDIService via the SSRF, it responds with the error:

```
The user "XYZ" isn't assigned to any management roles.
```

As a result, an authentication bypass is also needed to make use of this vulnerability. The ECP backend exposes an endpoint that is used for authentication between proxied components. The `/ecp/proxyLogon.ecp` endpoint uses request headers and a serialized XML request body to initialize an `EcpIdentity` within the `Microsoft.Exchange.Management.ControlPanel.RbacSettings` class. If a valid user SID is provided, it creates this identity and sets the `ASP.NET_SessionId` and `msExchEcpCanary` cookies needed to authenticate to `/ecp`. An example request is to `/ecp/proxyLogon.ecp` via the SSRF is shown below:

```
POST /ecp/favicon.ico HTTP/1.1
Host: example.local
Cookie: X-BEResource=@backend.example.local:444/ecp/proxyLogon.ecp#~1941962753;
msExchLogonMailbox: S-1-5-21-1234567890-123456789-1234567890-500

<r at="" ln=""><s>S-1-5-21-1234567890-123456789-1234567890-500</s></r>

HTTP/1.1 241
Cache-Control: private
Server: Microsoft-IIS/8.0
request-id: 00000000-0000-0000-0000-000000000000
X-CalculatedBETarget: example.local
X-Content-Type-Options: nosniff
X-DiagInfo: example
X-BEServer: example
X-FEServer: example
Set-Cookie: ASP.NET_SessionId=11111111-1111-1111-1111-111111111111; path=/; HttpOnly
Set-Cookie:
msExchEcpCanary=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 path=/ecp
Content-Length: 0
```

Exploit flow:

The active exploitation in the wild used this SSRF as the starting point for a full remote code execution chain against Microsoft Exchange. The general exploit flow is as follows:

- Leak the backend hostname
- Leak a user SID

- Authenticate with proxyLogon via SSRF (CVE-2021-26855)
- Use one of three remote code execution vulnerabilities via SSRF (CVE-2021-26857, CVE-2021-26858, or CVE-2021-27065)
    - *As an example, CVE-2021-27065 has the following flow (all via SSRF):*
    - List OABVirtualDirectory objects via the DDIService
    - Modify the OABVirtualDirectory to inject ASP code into the ExternalUrl (typically a webshell)
    - "Reset" the OABVirtualDirectory, which writes all properties to disk at a user-controlled path
    - Access this webshell externally (optionally using the SSRF)

Known cases of the same exploit flow:

- There have been other authenticated remote code execution gadgets within Exchange, such as CVE-2020-16875 and its bypass CVE-2020-171324.
- The authentication bypass via proxyLogon appears to be unknown prior to these exploits.

Part of an exploit chain? This vulnerability was used as part of the observed HAFNIUM exploitation as described by Microsoft.

The Next Steps

Variant analysis

Areas/approach for variant analysis (and why):

> Audit overrides of `GetTargetBackEndServerUrl` for similar mistakes in URI routing. Several request handlers override this method.

Found variants: N/A

Structural improvements

What are structural improvements such as ways to kill the bug class, prevent the introduction of this vulnerability, mitigate the exploit flow, make this type of vulnerability harder to exploit, etc.?

Ideas to kill the bug class:

Ideas to mitigate the exploit flow:

Other potential improvements:

0-day detection methods

What are potential detection methods for similar 0-days? Meaning are there any ideas of how this exploit or similar exploits could be detected **as a 0-day**?

Other References

- March 2, 2021: HAFNIUM targeting Exchange Servers with 0-day exploits by Microsoft. This post was the initial public disclosure of in-the-wild exploitation.
- March 2, 2021: Operation Exchange Marauder: Active Exploitation of Multiple Zero-Day Microsoft Exchange Vulnerabilities by Volexity. This post was released concurrently with the Microsoft disclosure.
- March 5, 2021: Proxylogon by DEVCORE, the team who reported the vulnerability to Microsoft.
- March 9, 2021: Reproducing the Microsoft Exchange Proxylogon Exploit Chain by the authors of this RCA describing the process for reproducing this exploit chain using publicly available information.