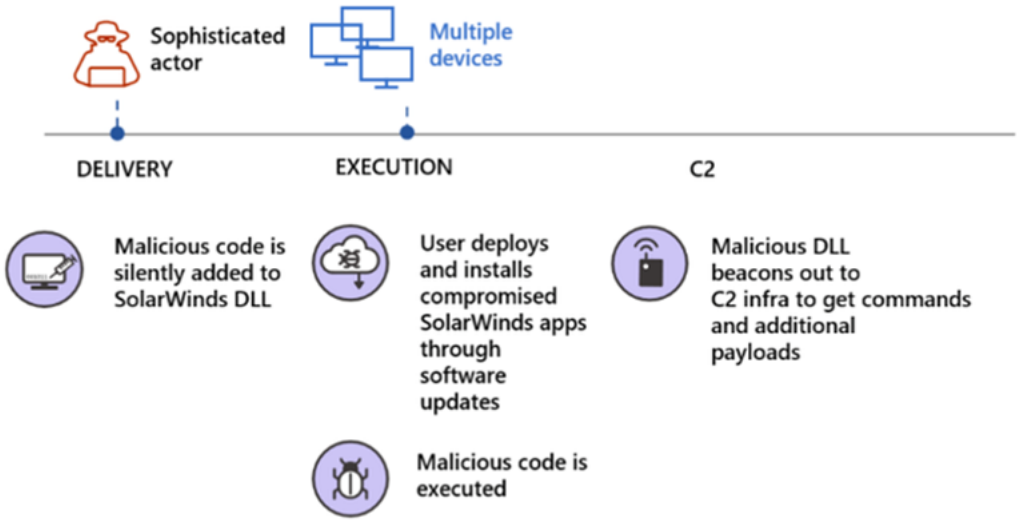
SolarWinds Post-Compromise Hunting with Azure Sentinel

techcommunity.microsoft.com/t5/azure-sentinel/solarwinds-post-compromise-hunting-with-azure-sentinel/ba-p/1995095

December 16, 2020



Solorigate supply chain attack diagram

MSTIC has released a number of new hunting and detection queries for Azure Sentinel based on additional observations as well as research released by partners and the wider community. In addition, the SolarWinds post compromise hunting workbook has been updated to include a number of new sections.

Blog sections have been marked with **Updated** and include the date they were last updated.

Microsoft recently blogged about the Recent Nation-State Cyber Attacks that has impacted high value targets both across the government and private sector. This attack is also known as Solorigate or Sunburst. This threat actor is believed to be highly sophisticated and motivated. Relevant security data required for hunting and investigating such a complex attack is produced in multiple locations - cloud, on-premises and across multiple security tools and product logs. Being able to analyze all the data from a single point makes it easier to spot trends and attacks. Azure Sentinel has made it easy to collect data from multiple data sources across different environments both on-prem and cloud with the goal of connecting that data together more easily. This blog post contains guidance and generic approaches to hunt for attacker activity (TTPs) in data that is available by default in Azure Sentinel or can be onboarded to Azure Sentinel.

Associated content details:

Updated 12/18/2020

Currently known in depth **attack details** have been provided by the M365 and MSTIC teams via the deep dive analysis blog**.**

Updated 12/21/2020

Current **advice for incident responders** on recovery from systemic identity compromises has been provided by Microsoft Detection and Response Team.

Updated 12/22/2020

Azure AD Identity admins who want to gain further visibility and understanding related to the identity implications of this attack on their environment can use the newly released Sensitive Operations Report workbook.

Updated 12/26/2020

For Identity Vendors and their customers to understand the Solorigate identity related attack components the Identity team at Microsoft has produced a blog has been created to walk thru this information.

Updated 12/28/2020

Users of Microsoft 365 Defender can also hunt and detect on similar items in this blog, but tailored towards investigation using Microsoft 365 Defender to protect against Solorigate.

The goal of this article is post-compromise investigation strategies and is focused on TTPs and not focused on specific IOCs. Azure Sentinel customers are encouraged to review advisories and IOC's shared by Microsoft MSRC and security partners to search on specific IOC's in their environment using Azure Sentinel. Links to these IOC's are listed in the reference section at the end.

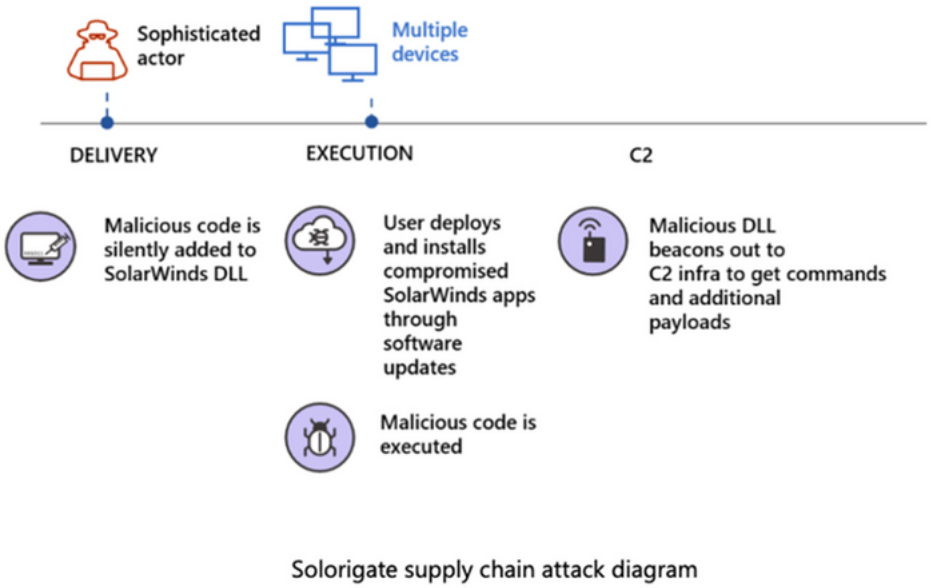
To make it easier for security teams to visualize and monitor their environments for this attack the MSTIC team has shared a SolarWinds Post Compromise hunting workbook via Azure Sentinel and Azure Sentinel GitHub. There are many things in this workbook that threat hunters would find useful and the workbook is complimentary to the hunting methods shared below. Importantly, if you have recently rotated ADFS key material this workbook can be useful in identifying attacker logon activity if they logon with old key material. Security teams should leverage this hunting workbook as part of their workflow in investigating this attack.

Thanks to the **MSTIC** and **M365** teams for collaborating to deliver this content in a timely manner. **Special thanks to** @aprakash13, @Ashwin_Patil, @Pete Bryan, @ItsReallyNick, Chris Glyer, @Cyb3rWard0g, @Tim Burrell (MSTIC), Rob Mead, @TomMcElroy, @Elia Florio, @Corina Feuerstein, Ramin Nafisi, Michael Matonis.

Please note that since Azure Sentinel and the M365 Advanced Hunting portal share the same query language and share similar data types, all of the referenced queries can be used directly or slightly modified to work in both.

Gaining a foothold

As shared in Microsoft's technical blog – Customer Guidance on Recent Nation-state Cyber Attacks - attackers might have compromised the internal build systems or the update distribution systems of SolarWinds Orion software then modified a DLL component in the legitimate software and embedded backdoor code that would allow these attackers to remotely perform commands or deliver additional payloads. Below is a representation of various attack stages which you can also see in Microsoft Threat Protection (MTP) portal. Note that if you do not have Microsoft Threat Protection this link will not work for you.



Solorigate supply chain attack diagram

To hunt for similar TTPs used in this attack, a good place to start is to build an inventory of the machines that have SolarWinds Orion components. Organizations might already have a software inventory management system to indicate hosts where the SolarWinds application is installed. Alternatively, Azure Sentinel could be leveraged to run a simple query to gather similar details. Azure Sentinel collects data from multiple different logs that could be used to gather this information. For example, through the recently released Microsoft 365 Defender connector, security teams can now easily ingest Microsoft 365 raw data into Azure Sentinel. Using the ingested data, a simple query like below can be written that will pull the hosts with SolarWinds process running in last 30 days based on Process execution either via host on boarded to Sentinel or on boarded via Microsoft Defender for Endpoints (MDE). The query also leverages the Sysmon logs that a lot of customers are collecting from their environment to surface the machines that have SolarWinds running on them. Similar queries that leverage M365 raw data could also be run from the M365's Advanced hunting portal.

SolarWinds Inventory check query

Spoiler

```
let timeframe = 30d;
(union isfuzzy=true
(
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == '4688'
| where tolower(NewProcessName) has 'solarwinds'
| extend MachineName = Computer , Process = NewProcessName
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated), MachineCount = dcount(MachineName), AccountCount = dcount
),
(
DeviceProcessEvents
| where TimeGenerated >= ago(timeframe)
| where tolower(InitiatingProcessFolderPath) has 'solarwinds'
| extend MachineName = DeviceName , Process = InitiatingProcessFolderPath, Account = AccountName
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated), MachineCount = dcount(MachineName), AccountCount = dcount
),
(
Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 1
| extend Image = EventDetail.[4].["#text"]
| where tolower(Image) has 'solarwinds'
| extend MachineName = Computer , Process = Image, Account = UserName
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated), MachineCount = dcount(MachineName), AccountCount = dcount
)
)
```

Updated 12/30/2020

On systems where the malicious SolarWinds DLL (SolarWinds.Orion.Core.BusinessLayer.dll) is running, it is known that the attacker used a hardcoded named pipe '583da945-62af-10e8-4902-a8f205c72b2e' to conduct various checks as well as to ensure only one instance of the backdoor was running. The use of named pipes by malware is not uncommon as it provides a mechanism for communication between processes. This activity by the malware can be detected if you are collecting Sysmon (Event Id 17/18) or Security Event Id 5145 in your Azure Sentinel workspace. The Solorigate Named Pipe detection should not be considered reliable on its own as the creation of just the hardcoded named pipe does not indicate that the malicious code was completely triggered, and the machine beaconed out or received additional commands. However, presence of this is definitely suspicious and should warrant further in-depth investigation.

Spoiler

```
let timeframe = 1d;
(union isfuzzy=true
(Event
| where TimeGenerated >= ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where EventID in (17,18)
```

```
| extend EvData = parse_xml(EventData)
| extend EventDetail = EvData.DataItem.EventData.Data
| extend NamedPipe = EventDetail.[5].["#text"]
| extend ProcessDetail = EventDetail.[6].["#text"]
| where NamedPipe contains '583da945-62af-10e8-4902-a8f205c72b2e'
| extend Account = UserName
| project-away EventDetail, EvData
),
(
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == '5145'
| where AccessList has '%%4418' // presence of CreatePipeInstance value
| where RelativeTargetName contains '583da945-62af-10e8-4902-a8f205c72b2e'
)
)
| extend timestamp = TimeGenerated, AccountCustomEntity = Account, HostCustomEntity = Computer
```

Privilege Escalation

Once the adversary acquires an initial foothold on a system thru the SolarWinds process they will have System account level access, the attacker will then attempt to elevate to domain admin level access to the environment. The Microsoft Threat Intelligence Center (MSTIC) team has already delivered multiple queries into Azure Sentinel that identify similar TTPs and many are also available in M365. These methodologies are not specific to just this threat actor or this attack but have been seen in various attack campaigns.

Identifying abnormal logon activities or additions to privileged groups is one way to identify privilege escalation.

Updated 12/17/2020

- Checking for hosts with new logons to identify potential lateral movement by the attacker.
- Look for any new account being created and added to built-in administrators group.
- Look for any user account added to privileged built in domain local or global groups, including adding accounts to a domain privileged group such as Enterprise Admins, Cert Publishers or DnsAdmins.
- Monitor for rare activity by a high-value account carried out on a system or service.

Related to this attack, in some environments service account credentials had been granted administrative privileges. The above queries can be modified to remove the condition of focusing “User” accounts by commenting the query to include service accounts in the scope where applicable:

//| where AccountType == "User"

Please see the Azure Sentinel Github for additional queries and hunting ideas related to Accounts under the Detections and Hunting Queries sections for AuditLogs, and SecurityEvents

Microsoft 365 Defender team has also shared quite a few sample queries for use in their advanced hunting portal that could be leveraged to detect this part of the attack. Additionally, the logic for many of the Azure Sentinel queries can also be transformed to equivalent queries for Microsoft 365 Defender, that could be run in their Advanced Hunting Portal.

Microsoft 365 Defender has an upcoming complimentary blog that will be updated here once available.

Certificate Export

The next step in the attack was stealing the certificate that signs SAML tokens from the federation server (ADFS) called a Token Signing Cert (TSC). SAML Tokens are basically XML representations of claims. You can read more about ADFS in What is federation with Azure AD? | Microsoft Docs and SAML at Azure Single Sign On SAML Protocol - Microsoft identity platform | Microsoft Docs. The process is as follows:

1. A client requests a SAML token from an ADFS Server by authenticating to that server using Windows credentials.
2. The ADFS server issues a SAML token to the client.
3. The SAML token is signed with a certificate associated with the server.
4. The client then presents the SAML token to the application that it needs access to.
5. The signature over the SAML token tells the application that the security token service issued the token and grants access to the client.

ADFS Key Extraction

Updated 01/15/2021

The implication of stealing the Token Signing Cert (TSC) is that once the certificate has been acquired, the actor can forge SAML (Security Assertions Markup Language) tokens with whatever claims and lifetime they choose, then sign it with the certificate that has been acquired. Microsoft continues to strongly recommend securing your AD FS (Active Directory Federation Service) TSC because if these TSC's are acquired by a bad actor, this then enables the actor to forge SAML tokens that impersonate highly privileged accounts. There are publicly available pen-testing tools like ADFSDump and ADFSpoof that help with extracting required information from the AD FS configuration database to generate the forged security tokens. While we have not confirmed these specific tools were used in this attack, they are useful for simulating the attack behavior or executing a similar attack and therefore, Microsoft has created a high-fidelity detection related to this for M365 Defender:

ADFS private key extraction which detects ADFS private key extraction patterns from tools such as ADFSDump.

Note: Any M365 Defender alert can be seen in Azure Sentinel Security Alerts or in the M365 security portal.

Updated 01/15/2021

The TTP (tactics, techniques, and procedures) observed in the Solorigate attack is the creation of a legitimate SAML token used to authenticate as any user. One way an attacker could achieve this is by compromising FS key material. Microsoft has a new detection for this as stated above and for Azure Sentinel has also created a Windows Event Log based detection that indicates an ADFS DKM Master Key Export. As part of the update for this query to the Azure Sentinel GitHub, there is a detailed write up for why this is interesting along with a subsequent addition providing clarity on how to get 4662 events to fire. This detection should not be considered reliable on its own but can identify suspicious activity that warrants further investigation.

Updated 01/15/2021

Spoiler

```
(union isfuzzy=true (SecurityEvent
| where EventID == 4662 // You need to create a SACL on the ADFS Policy Store DKM group for this event to be created.
| where ObjectServer == 'DS'
| where OperationType == 'Object Access'
//| where ObjectName contains '<GUID of ADFS Policy Store DKM Group object' This is unique to the domain. Check description for more details.
| where ObjectType contains '5cb41ed0-0e4c-11d0-a286-00aa003049e2' // Contact Class
| where Properties contains '8d3bca50-1d7e-11d0-a081-00aa006c33ed' // Picture Attribute - Ldap-Display-Name: thumbnailPhoto
| extend timestamp = TimeGenerated, HostCustomEntity = Computer, AccountCustomEntity = SubjectAccount),
(DeviceEvents
| where ActionType =~ "LdapSearch"
| where AdditionalFields.AttributeList contains "thumbnailPhoto"
| where AdditionalFields.DistinguishedName contains "CN=ADFS,CN=Microsoft,CN=Program Data" // Filter results to show only hits related to the ADFS AD container
| extend timestamp = TimeGenerated, HostCustomEntity = DeviceName, AccountCustomEntity = InitiatingProcessAccountName)
)
```

Updated 12/19/2020

MSTIC has developed another detection for ADFS server key export events. This detection leverages the visibility provided by Sysmon and provides a more reliable detection method than that covered in the Windows Event Log detection. For this detection to be effective you must be collecting Sysmon Event IDs 17 and 18 into your Azure Sentinel workspace.

Spoiler

```
// Adjust this to use a longer timeframe to identify ADFS servers
let lookback = 6d;
// Adjust this to adjust the key export detection timeframe
let timeframe = 1d;
// Start be identifying ADFS servers to reduce FP chance
let ADFS_Servers = (
Event
| where TimeGenerated > ago(timeframe+lookback)
| where Source == "Microsoft-Windows-Sysmon"
| extend EventData = parse_xml(EventData).DataItem.EventData.Data
| mv-expand bagexpansion=array EventData
| evaluate bag_unpack(EventData)
| extend Key=tostring(['@Name']), Value=['#text']
| evaluate pivot(Key, any(Value), TimeGenerated, Source, EventLog, Computer, EventLevel, EventLevelName, EventID, UserName, RenderedDescription, MG, ManagementGroupName, Type, _ResourceId)
| extend process = split(Image, '\\', -1)[-1]
| where process =~ "Microsoft.IdentityServer.ServiceHost.exe"
```

```
| summarize by Computer);
// Look for ADFS servers where Named Pipes event are present
Event
| where TimeGenerated > ago(timeframe)
| where Source == "Microsoft-Windows-Sysmon"
| where Computer in~ (ADFS_Servers)
| extend RenderedDescription = tostring(split(RenderedDescription, ":")[0])
| extend EventData = parse_xml(EventData).DataItem.EventData.Data
| mv-expand bagexpansion=array EventData
| evaluate bag_unpack(EventData)
| extend Key=tostring(['@Name']), Value=['#text']
| evaluate pivot(Key, any(Value), TimeGenerated, Source, EventLog, Computer, EventLevel, EventLevelName, EventID, UserName,
RenderedDescription, MG, ManagementGroupName, Type, _ResourceId)
| extend RuleName = column_ifexists("RuleName", ""), TechniqueId = column_ifexists("TechniqueId", ""), TechniqueName =
column_ifexists("TechniqueName", "")
| parse RuleName with * 'technique_id=' TechniqueId ',' * 'technique_name=' TechniqueName
| where EventID in (17,18)
// Look for Pipe related to querying the WID
| where PipeName == "\\MICROSOFT##WID\\tsql\\query"
| extend process = split(Image, '\\', -1)[-1]
// Exclude expected processes
| where process !in ("Microsoft.IdentityServer.ServiceHost.exe", "Microsoft.Identity.Health.Adfs.PshSurrogate.exe", "AzureADConnect.exe",
"Microsoft.Tri.Sensor.exe", "wsmprovhost.exe","mmc.exe", "sqlservr.exe")
| extend Operation = RenderedDescription
| project-reorder TimeGenerated, EventType, Operation, process, Image, Computer, UserName
| extend HostCustomEntity = Computer, AccountCustomEntity = UserName
```

Outside of directly looking for tools, this adversary may have used custom tooling so looking for anomalous process executions or anomalous accounts logging on to our ADFS server can give us some clue when such attacks happen. Azure Sentinel provides queries that can help to:

- Find rare anomalous process in your environment.
- Also look for rare processes run by service accounts
- Or uncommon processes that are in the bottom 5% of all the process.
- In some instances, there is a rare command line syntax related to DLL loading, you can adjust these queries to also look at rarity on the command line.

Every environment is different and some of these queries being generic could be noisy. So, in the first step a good approach would be to limit this kind of hunting to our ADFS server.

Azure Active Directory Hunting

Having gained a significant foothold in the on prem environment, the actor also targeted the Azure AD of some of the compromised organizations and made modifications to Azure AD settings to facilitate long term access. Microsoft has shared many relevant queries through the Azure Sentinel GitHub to identify these actions.

Updated 01/15/2021

One such activity is related to modifying domain federation trust settings. A federation trust signifies the establishment of authentication and authorization trust between two organizations so that users located in partner organizations can send authentication and authorization requests successfully.

> While not specifically seen in this attack, tracking federation trust modifications is important. The Azure Sentinel query for domain federation trust settings modification will alert when a user or application modifies the federation settings on the domain particularly when a new Active Directory Federated Service (ADFS) Trusted Realm object, such as a signing certificate, is added to the domain or there is an update to domain authentication from **managed** to **federated.** Modification to domain federation settings should be rare and this should be treated as a high-fidelity alert that Azure AD and Azure Sentinel users should follow up on.

Updated 01/15/2021

The original purpose of the STSRefreshTokenModification low severity, hunting-only query was to demonstrate an event that has token validity time periods in it and demonstrate how one could monitor for anomalous/edited tokens. We have determined this event will only fire on the manual expiration of the StsRefreshToken by an admin (or the user). These types of events are most often generated when legitimate administrators troubleshoot frequent AAD (Azure AD) user sign-ins. To avoid any confusion with Solorigate investigation and hunting, we have removed this section from the blog.

Another such activity is adding access to the Service Principal or Application. If a threat actor obtains access to an Application Administrator account, they may configure alternate authentication mechanisms for direct access to any of the scopes and services available to the Service Principal. With these privileges, the actor can add alternative authentication material for direct access to resources using this credential.

> Identify where the verify KeyCredential has **been updated** with New access credential added to Application or Service Principal.

Updated 12/20/2020

> Identify where the verify KeyCredential **was not present** and has now has had its First access credential added to Application or Service Principal where no credential was present.

Spoiler

New access credential added to Application or Service Principal

```
let auditLookback = 1h;
AuditLogs
| where TimeGenerated > ago(auditLookback)
| where OperationName has_any ("Add service principal", "Certificates and secrets management") // captures "Add service principal", "Add service principal credentials", and "Update application – Certificates and secrets management" events
| where Result =~ "success"
| mv-expand target = TargetResources
| where tostring(InitiatedBy.user.userPrincipalName) has "@" or tostring(InitiatedBy.app.displayName) has "@"
| extend targetDisplayName = tostring(TargetResources[0].displayName)
| extend targetId = tostring(TargetResources[0].id)
| extend targetType = tostring(TargetResources[0].type)
| extend keyEvents = TargetResources[0].modifiedProperties
| mv-expand keyEvents
| where keyEvents.displayName =~ "KeyDescription"
| extend new_value_set = parse_json(tostring(keyEvents.newValue))
| extend old_value_set = parse_json(tostring(keyEvents.oldValue))
| where old_value_set != "[]"
| extend diff = set_difference(new_value_set, old_value_set)
| where isnotempty(diff)
| parse diff with * "KeyIdentifier=" keyIdentifier:string ",KeyType=" keyType:string ",KeyUsage=" keyUsage:string ",DisplayName=" keyDisplayName:string "]" *
| where keyUsage == "Verify" or keyUsage == ""
| extend UserAgent = iff(AdditionalDetails[0].key == "User-Agent",tostring(AdditionalDetails[0].value),"")
| extend InitiatingUserOrApp = iff(isnotempty(InitiatedBy.user.userPrincipalName),tostring(InitiatedBy.user.userPrincipalName), tostring(InitiatedBy.app.displayName))
| extend InitiatingIpAddress = iff(isnotempty(InitiatedBy.user.ipAddress), tostring(InitiatedBy.user.ipAddress), tostring(InitiatedBy.app.ipAddress))
// The below line is currently commented out but Azure Sentinel users can modify this query to show only Application or only Service Principal events in their environment
//| where targetType =~ "Application" // or targetType =~ "ServicePrincipal"
| project-away diff, new_value_set, old_value_set
| project-reorder TimeGenerated, OperationName, InitiatingUserOrApp, InitiatingIpAddress, UserAgent, targetDisplayName, targetId, targetType, keyDisplayName, keyType, keyUsage, keyIdentifier, CorrelationId, TenantId
| extend timestamp = TimeGenerated, AccountCustomEntity = InitiatingUserOrApp, IPCustomEntity = InitiatingIpAddress
```

First access credential added to Application or Service Principal where no credential was present

```
let auditLookback = 1h;
AuditLogs
| where TimeGenerated > ago(auditLookback)
| where OperationName has_any ("Add service principal", "Certificates and secrets management") // captures "Add service principal", "Add service principal credentials", and "Update application – Certificates and secrets management" events
| where Result =~ "success"
| mv-expand target = TargetResources
| where tostring(InitiatedBy.user.userPrincipalName) has "@" or tostring(InitiatedBy.app.displayName) has "@"
| extend targetDisplayName = tostring(TargetResources[0].displayName)
| extend targetId = tostring(TargetResources[0].id)
| extend targetType = tostring(TargetResources[0].type)
| extend keyEvents = TargetResources[0].modifiedProperties
| mv-expand keyEvents
| where keyEvents.displayName =~ "KeyDescription"
| extend new_value_set = parse_json(tostring(keyEvents.newValue))
```

```
| extend old_value_set = parse_json(tostring(keyEvents.oldValue))
| where old_value_set == "[]"
| parse new_value_set with * "KeyIdentifier=" keyIdentifier:string ",KeyType=" keyType:string ",KeyUsage=" keyUsage:string ",DisplayName="
keyDisplayName:string "]" *
| where keyUsage == "Verify" or keyUsage == ""
| extend UserAgent = iff(AdditionalDetails[0].key == "User-Agent",tostring(AdditionalDetails[0].value),"")
| extend InitiatingUserOrApp = iff(isnotempty(InitiatedBy.user.userPrincipalName),tostring(InitiatedBy.user.userPrincipalName),
tostring(InitiatedBy.app.displayName))
| extend InitiatingIpAddress = iff(isnotempty(InitiatedBy.user.ipAddress), tostring(InitiatedBy.user.ipAddress),
tostring(InitiatedBy.app.ipAddress))
// The below line is currently commented out but Azure Sentinel users can modify this query to show only Application or only Service Principal
events in their environment
//| where targetType =~ "Application" // or targetType =~ "ServicePrincipal"
| project-away new_value_set, old_value_set
| project-reorder TimeGenerated, OperationName, InitiatingUserOrApp, InitiatingIpAddress, UserAgent, targetDisplayName, targetId,
targetType, keyDisplayName, keyType, keyUsage, keyIdentifier, CorrelationId, TenantId
| extend timestamp = TimeGenerated, AccountCustomEntity = InitiatingUserOrApp, IPCustomEntity = InitiatingIpAddress
```

Updated 12/19/2020

This threat actor has been observed using applications to read users mailboxes within a target environment. To help identify this activity MSTIC has created a hunting query that looks for applications that have been granted mailbox read permissions followed by consent to this application. Whilst this may uncover legitimate applications hunters should validate applications granted mail read permissions genuinely require them.

Spoiler

```
AuditLogs
| where Category =~ "ApplicationManagement"
| where ActivityDisplayName =~ "Add delegated permission grant"
| where Result =~ "success"
| where tostring(InitiatedBy.user.userPrincipalName) has "@" or tostring(InitiatedBy.app.displayName) has "@"
| extend props = parse_json(tostring(TargetResources[0].modifiedProperties))
| mv-expand props
| extend UserAgent = tostring(AdditionalDetails[0].value)
| extend InitiatingUser = tostring(parse_json(tostring(InitiatedBy.user)).userPrincipalName)
| extend UserIPAddress = tostring(parse_json(tostring(InitiatedBy.user)).ipAddress)
| extend DisplayName = tostring(props.displayName)
| extend Permissions = tostring(parse_json(tostring(props.newValue)))
| where Permissions has_any ("Mail.Read", "Mail.ReadWrite")
| extend PermissionsAddedTo = tostring(TargetResources[0].displayName)
| extend Type = tostring(TargetResources[0].type)
| project-away props
| join kind=leftouter(
AuditLogs
| where ActivityDisplayName has "Consent to application"
| extend AppName = tostring(TargetResources[0].displayName)
| extend AppId = tostring(TargetResources[0].id)
| project AppName, AppId, CorrelationId) on CorrelationId
| project-reorder TimeGenerated, OperationName, InitiatingUser, UserIPAddress, UserAgent, PermissionsAddedTo, Permissions, AppName,
AppId, CorrelationId
| extend timestamp = TimeGenerated, AccountCustomEntity = InitiatingUser, IPCustomEntity = UserIPAddress
```

It's also advised to hunt for application consents for unexpected applications, particularly where they provide offline access to data or other high value access;

- Suspicious application consent similar to O365 Attack Toolkit
- Suspicious application consent for offline access

Updated 12/17/2020 (moved location)

In addition to Azure AD pre-compromise logon hunting it is also possible to monitor for logons attempting to use invalid key material. This can help identify attempted logons using stolen key material made after key material has been rotated. This can be done by querying SigninLogs in Azure Sentinel where the ResultType is 5000811. **Please note** that if you roll your token signing certificate, there will be expected activity when searching on the above.

Recon and Remote Execution

Updated 12/27/2020

The adversary will often attempt to access on-prem systems to gain further insight and mapping of the environment. As described in the **Resulting hands-on-keyboard attack** section of the Analyzing Solorigate blog by Microsoft, attackers renamed windows administrative tools like adfind.exe which were then used for domain enumeration. An example of the process execution command like can look like this:

```
C:\Windows\system32\cmd.exe /C csrss.exe -h breached.contoso.com -f (name=”Domain Admins”) member -list | csrss.exe -h
breached.contoso.com -f objectcategory=* > .\Mod\mod1.log
```

We have provided a query in the Azure Sentinel Github which will help in detecting the command line patterns related to ADFind usage. You can customize this query to look at your specific DC/ADFS servers.

Spoiler

```
let startdate = 1d;
let lookupwindow = 2m;
let threshold = 3; //number of commandlines in the set below
let DCADFSServersList = dynamic (["DCServer01", "DCServer02", "ADFSServer01"]); // Enter a reference list of hostnames for your DC/ADFS
servers
let tokens = dynamic(["objectcategory","domainlist","dcmodes","adinfo","trustdmp","computers_pwdnotreqd","Domain Admins",
"objectcategory=person", "objectcategory=computer", "objectcategory=*"]);
SecurityEvent
//| where Computer in (DCADFSServersList) // Uncomment to limit it to your DC/ADFS servers list if specified above or any pattern in
hostnames (startswith, matches regex, etc).
| where TimeGenerated between (ago(startdate) .. now())
| where EventID == 4688
| where CommandLine has_any (tokens)
| where CommandLine matches regex "(.*)>(.*)"
| summarize Commandlines = make_set(CommandLine), LastObserved=max(TimeGenerated) by bin(TimeGenerated, lookupwindow),
Account, Computer, ParentProcessName, NewProcessName
| extend Count = array_length(Commandlines)
| where Count > threshold
```

On the remote execution side, there is a pattern that can be identified related to using alternate credentials than the currently logged on user, such as when using the RUN AS feature on a Windows system and passing in explicit credentials. We have released a query that will identify when execution is occurring via multiple explicit credentials against remote targets. This requires that Windows Event 4648 is being collected as part of Azure Sentinel.

Spoiler

```
let WellKnownLocalSIDs = "S-1-5-[0-9][0-9]$";
let protocols = dynamic(['cifs', 'ldap', 'RPCSS', 'host' , 'HTTP', 'RestrictedKrbHost', 'TERMSRV', 'msomsdksvc', 'mssqlsvc']);
SecurityEvent
| where TimeGenerated >= ago(1d)
| where EventID == 4648
| where SubjectUserSid != 'S-1-0-0' // this is the Nobody SID which really means No security principal was included.
| where not(SubjectUserSid matches regex WellKnownLocalSIDs) //excluding system account/service account as this is generally normal
| where TargetInfo has '/' //looking for only items that indicate an interesting protocol is included
| where Computer !has tostring(split(TargetServerName,'$')[0])
| where TargetAccount !~ tostring(split(SubjectAccount,'$')[0])
| extend TargetInfoProtocol = tolower(split(TargetInfo, '/')[0]), TargetInfoMachine = toupper(split(TargetInfo, '/')[1])
| extend TargetAccount = tolower(TargetAccount), SubjectAccount = tolower(SubjectAccount)
| extend UncommonProtocol = case(not(TargetInfoProtocol has_any (protocols)), TargetInfoProtocol, 'NotApplicable')
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated), AccountsUsedCount = dcount(TargetAccount),
AccountsUsed = make_set(TargetAccount), TargetMachineCount = dcount(TargetInfoMachine),
TargetMachines = make_set(TargetInfoMachine), TargetProtocols = dcount(TargetInfoProtocol), Protocols = make_set(TargetInfoProtocol),
Processes = make_set(Process) by Computer, SubjectAccount, UncommonProtocol
| where TargetMachineCount > 1 or UncommonProtocol != 'NotApplicable'
| extend ProtocolCount = array_length(Protocols)
| extend ProtocolScore = case(
Protocols has 'rpcss' and Protocols has 'host' and Protocols has 'cifs', 10, //observed in Solorigate and depending on which are used together
the higher the score
Protocols has 'rpcss' and Protocols has 'host', 5,
Protocols has 'rpcss' and Protocols has 'cifs', 5,
Protocols has 'host' and Protocols has 'cifs', 5,
Protocols has 'ldap' or Protocols has 'rpcss' or Protocols has 'host' or Protocols has 'cifs', 1, //ldap is more commonly seen in general, this was
```

also seen with Solorigate but not usually to the same machines as the others above
UncommonProtocol != 'NotApplicable', 3,
0 //other protocols may be of interest, but in relation to observations for enumeration/execution in Solorigate they receive 0
)
| extend Score = ProtocolScore + ProtocolCount + AccountsUsedCount
| where Score >= 9 or (UncommonProtocol != 'NotApplicable' and Score >= 4) // Score must be 9 or better as this will include 5 points for atleast 2 of the interesting protocols + the count of protocols (min 2) + the number of accounts used for execution (min 2) = min of 9 OR score must be 4 or greater for an uncommon protocol
| extend TimePeriod = EndTime - StartTime //This identifies the time between start and finish for the use of the explicit credentials, shorter time period may indicate scripted executions
| project-away UncommonProtocol
| extend timestamp = StartTime, AccountCustomEntity = SubjectAccount, HostCustomEntity = Computer
| order by Score desc

Data Access

Accessing confidential data is one of the primary motives of this attack. Data access for the attacker here relied on leveraging minted SAML tokens to access user files/email stored in the cloud via compromised AppIds. One way to detect this is when a user or application signs in using Azure Active Directory PowerShell to access non-Active Directory resources.

Microsoft Graph is one way that the attacker may be seen accessing resources and can help find what the attacker may have accessed using the Service principal Azure Active Directory sign-in logs. If you have data in your Log analytics you could easily plot a chart to see what anomalous activity is happening in your environment that is leveraging the graph.

Updated 12/17/2020

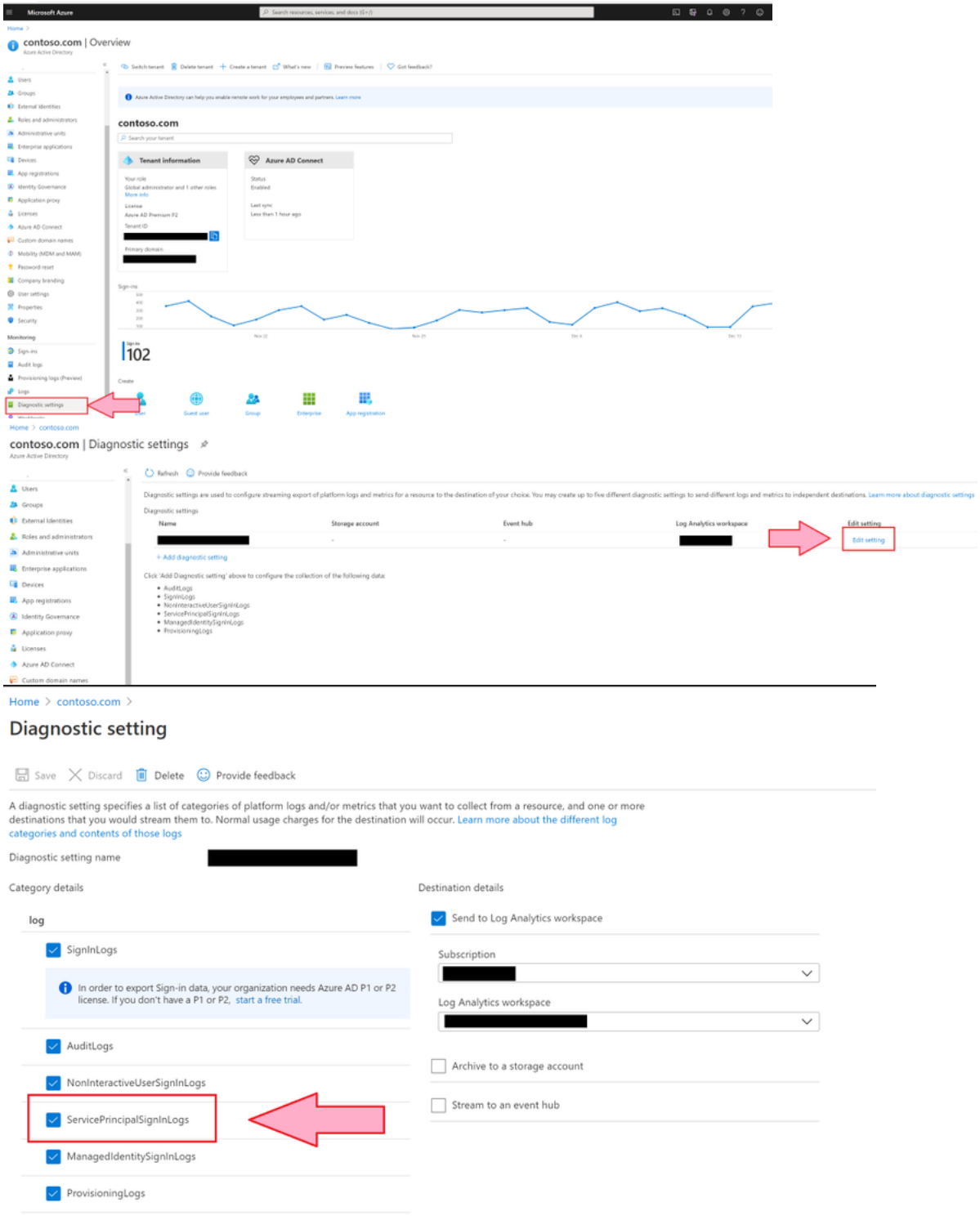
Note that this data type in Azure Sentinel below is only available when additional Diagnostic Logging is enabled on the workspace. Please see the instructions in the expandable section below.

Spoiler

The AADServicePrincipalSigninLogs datatype will not be available in Azure Sentinel unless it is configured under Diagnostic Settings. Please see screenshots below the query.

AADServicePrincipalSignInLogs
| where TimeGenerated > ago(90d)
| where ResourceDisplayName == "Microsoft Graph"
| where ServicePrincipalId == "524c43c4-c484-4f7a-bd44-89d4a0d8aeab"
| summarize count() by bin(TimeGenerated, 1h)
| render timechart

To enable Service Principal Signin Logging, do the following:



Updated 12/21/2020

Additionally, below is a sample query that brings out some of the logons to Azure AD where multi factor authentication was satisfied by token based logons versus MFA via phone auth or the like. It is possible this could produce many results, so additional tuning is suggested for your environment.

Spoiler

```
SigninLogs
| where TimeGenerated > ago(30d)
| where ResultType == 0
| extend additionalDetails = tostring(Status.additionalDetails)
| summarize make_set(additionalDetails), min(TimeGenerated), max(TimeGenerated) by IPAddress, UserPrincipalName
```

```
| where array_length(set_additionalDetails) == 2
| where (set_additionalDetails[1] == "MFA requirement satisfied by claim in the token" and set_additionalDetails[0] == "MFA requirement satisfied by claim provided by external provider") or (set_additionalDetails[0] == "MFA requirement satisfied by claim in the token" and set_additionalDetails[1] == "MFA requirement satisfied by claim provided by external provider")
//| project IPAddress, UserPrincipalName, min_TimeGenerated, max_TimeGenerated
```

UPDATED 12/17/2020

This attack also used Virtual Private Servers (VPS) hosts to access victim networks and can be used in conjunction with the query above. Both MSTIC and FireEye have reported attacker logon events coming from network ranges associated with VPS providers. In order to highlight these logons, MSTIC has created a new hunting query - Signins From VPS Providers - that looks for successful signins from network ranges associated with VPS providers. This is joined with the above query, the new query looks for IPs that only display sign-ins based on tokens and not other MFA options, although this could be removed if wanted. The list of VPS ranges in the query is not comprehensive and there is significant potential for false positives so results should be investigated before responding, however it can provide very effective signal. Combining the query below with data that list VPS server ranges will make this a high-confidence hunting query.

In relation to the VPS servers section above, the previously mentioned workbook has a section that shows successful user signins from VPS (Virtual Private Server) providers where only tokens were used to authenticate. This uses the new KQL operator ipv4_lookup to evaluate if a login came from a known VPS provider network range. This operator can alternatively be used to look for all logons not coming from known ranges should your environment have a common logon source.

Data Exfiltration

Updated 12/20/2020

Email data has been observed as a target for the Solorigate attackers, one way to monitor for potential suspicious access is to look for anomalous MailItemsAccessed volumes. MSTIC has created a specific hunting query to identify Anomolous User Accessing Other Users Mailbox which can help to identify malicious activity related to this attack. Additionally, MSTIC previously created a more generic detection - Exchange workflow MailItemsAccessed operation anomaly - which looks for time series based anomalies in MailItemsAccessed events in the OfficeActivity log.

Spoiler

Anomalous access to other user's mailboxes

```
let timeframe = 14d;
let user_threshold = 1;
let folder_threshold = 5;
OfficeActivity
| where TimeGenerated > ago(timeframe)
| where Operation =~ "MailItemsAccessed"
| where ResultStatus =~ "Succeeded"
| mv-expand parse_json(Folders)
| extend folders = tostring(Folders.Path)
| where tolower(MailboxOwnerUPN) != tolower(UserId)
| extend ClientIP = iif(Client_IPAddress startswith "[", extract("\\[([^\\]]*)", 1, Client_IPAddress), Client_IPAddress)
| summarize make_set(folders), make_set(ClientInfoString), make_set(ClientIP), make_set(MailboxGuid), make_set(MailboxOwnerUPN) by UserId
| extend folder_count = array_length(set_folders)
| extend user_count = array_length(set_MailboxGuid)
| where user_count > user_threshold or folder_count > folder_threshold
| sort by user_count desc
| project-reorder UserId, user_count, folder_count, set_MailboxOwnerUPN, set_ClientIP, set_ClientInfoString, set_folder
```

Exchange workflow MailItemsAccessed operation anomaly

```
let starttime = 14d;
let endtime = 1d;
let timeframe = 1h;
let scorethreshold = 1.5;
let percentthreshold = 50;
// Preparing the time series data aggregated hourly count of MailItemsAccessd Operation in the form of multi-value array to use with time series anomaly function.
let TimeSeriesData =
OfficeActivity
| where TimeGenerated between (startofday(ago(starttime))..startofday(ago(endtime)))
| where OfficeWorkload=~ "Exchange" and Operation =~ "MailItemsAccessed" and ResultStatus =~ "Succeeded"
| project TimeGenerated, Operation, MailboxOwnerUPN
```

```
| make-series Total=count() on TimeGenerated from startofday(ago(starttime)) to startofday(ago(endtime)) step timeframe;
let TimeSeriesAlerts = TimeSeriesData
| extend (anomalies, score, baseline) = series_decompose_anomalies(Total, scorethreshold, -1, 'linefit')
| mv-expand Total to typeof(double), TimeGenerated to typeof(datetime), anomalies to typeof(double), score to typeof(double), baseline to typeof(long)
| where anomalies > 0
| project TimeGenerated, Total, baseline, anomalies, score;
// Joining the flagged outlier from the previous step with the original dataset to present contextual information
// during the anomalyhour to analysts to conduct investigation or informed decisions.
TimeSeriesAlerts | where TimeGenerated > ago(2d)
// Join against base logs since specified timeframe to retrive records associated with the hour of anomoly
| join (
OfficeActivity
| where TimeGenerated > ago(2d)
| where OfficeWorkload=~ "Exchange" and Operation =~ "MailItemsAccessed" and ResultStatus =~ "Succeeded"
) on TimeGenerated
```

Updated 12/19/2020

Targeting of email data has also been observed by other industry members including Volexity who reported attackers using PowerShell commands to export on premise Exchange mailboxes and then hosting those files on OWA servers in order to exfiltrate them.

MSTIC has created detections to identify this activity at both the OWA server and attacking host level through IIS logs, and PowerShell command line logging.

OWA exfiltration:

Spoiler

```
let excludeIps = dynamic(["127.0.0.1", "::1"]);
let scriptingExt = dynamic(["aspx", "ashx", "asp"]);
W3CIISLog
| where csUriStem contains "/owa/"
//The actor pulls a file back but won't send it any URI params
| where isempty(csUriQuery)
| extend file_ext = tostring(split(csUriStem, ".")[-1])
//Giving your file a known scripting extension will throw an error
//rather than just serving the file as it will try to interpret the script
| where file_ext !in~ (scriptingExt)
//The actor was seen using image files, but we go wider in case they change this behaviour
//| where file_ext in~ ("jpg", "jpeg", "png", "bmp")
| extend file_name = tostring(split(csUriStem, "/")[-1])
| where file_name != ""
| where cIP !in~ (excludeIps)
| project file_ext, csUriStem, file_name, Computer, cIP, sIP, TenantId, TimeGenerated
| summarize dcount(cIP), AccessingIPs=make_set(cIP), AccessTimes=make_set(TimeGenerated), Access=count() by TenantId, file_name, Computer, csUriStem
//Collection of the exfiltration will occur only once, lets check for 2 accesses in case they mess up
//Tailor this for hunting
| where Access <= 2 and dcount_cIP == 1
```

Host creating then removing mailbox export requests using PowerShell cmdlets:

Spoiler

```
 // Adjust the timeframe to change the window events need to occur within to alert

 let timeframe = 1h;

 SecurityEvent

 | where Process in ("powershell.exe", "cmd.exe")

 | where CommandLine contains 'New-MailboxExportRequest'

 | summarize by Computer, timekey = bin(TimeGenerated, timeframe), CommandLine, SubjectUserName

 | join kind=inner (SecurityEvent

 | where Process in ("powershell.exe", "cmd.exe")
```

```
| where CommandLine contains 'Remove-MailboxExportRequest'

| summarize by Computer, timekey = bin(TimeGenerated, timeframe), CommandLine, SubjectUserName) on Computer, timekey, SubjectUserName

| extend commands = pack_array(CommandLine1, CommandLine)

| summarize by timekey, Computer, tostring(commands), SubjectUserName

| project-reorder timekey, Computer, SubjectUserName, ['commands']

| extend HostCustomEntity = Computer, AccountCustomEntity = SubjectUserName
```

Updated 12/28/2020

Email Delegation and later delegate access is another tactic that has been observed to gain access to user's mailboxes. We have a previously created a method to discover Non-owner mailbox login activity that can be applied here to help identify when delegates are inappropriately access email.

Spoiler

```
let timeframe = 1d;
OfficeActivity
| where TimeGenerated >= ago(timeframe)
| where Operation == "MailboxLogin" and Logon_Type != "Owner"
| summarize count(), min(TimeGenerated), max(TimeGenerated) by Operation, OrganizationName, UserType, UserId, MailboxOwnerUPN, Logon_Type
| extend timestamp = min_TimeGenerated, AccountCustomEntity = UserId
```

Domain Hunting

Updated 12/17/2020

Domain specific

MSTIC has collated network based IoCs from MSTIC, FireEye and Volexity to create a network based IoC detection - Solorigate Network Beacon - that leverage multiple network focused data sources within Azure Sentinel.

Spoiler

```
let domains =
dynamic(["incomeupdate.com","zupertech.com","databasegalore.com","panhardware.com","avsvmcloud.com","digitalcollege.org","freescanonline.c
let timeframe = 6h;
(union isfuzzy=true
(CommonSecurityLog
| where TimeGenerated >= ago(timeframe)
| parse Message with * '(' DNSName ')' *
| where DNSName in~ (domains) or DestinationHostName has_any (domains) or RequestURL has_any(domains)
| extend AccountCustomEntity = SourceUserID, HostCustomEntity = DeviceName, IPCustomEntity = SourceIP
),
(DnsEvents
| where TimeGenerated >= ago(timeframe)
| extend DNSName = Name
| where isnotempty(DNSName)
| where DNSName in~ (domains)
| extend IPCustomEntity = ClientIP
),
(VMConnection
| where TimeGenerated >= ago(timeframe)
| parse RemoteDnsCanonicalNames with * '["' DNSName '"]' *
| where isnotempty(DNSName)
| where DNSName in~ (domains)
| extend IPCustomEntity = RemoteIp
),
(DeviceNetworkEvents
| where TimeGenerated >= ago(timeframe)
| where isnotempty(RemoteUrl)
| where RemoteUrl has_any (domains)
| extend DNSName = RemoteUrl
```

```
| extend IPCustomEntity = RemoteIP
| extend HostCustomEntity = DeviceName
)
)
```

Domain DGA

The avsvmcloud[.]com has been observed by several organizations as making DGA like subdomain queries as part of C2 activities. MSTIC have generated a hunting query - Solorigate DNS Pattern - to look for similar patterns of activity from other domains that might help identify other potential C2 sources.

Spoiler

```
let cloudApiTerms = dynamic(["api", "east", "west"]);
DnsEvents
| where IPAddresses != "" and IPAddresses != "127.0.0.1"
| where Name endswith ".com" or Name endswith ".org" or Name endswith ".net"
| extend domain_split = split(Name, ".")
| where tostring(domain_split[-5]) != "" and tostring(domain_split[-6]) == ""
| extend sub_domain = tostring(domain_split[0])
| where sub_domain !contains "-"
| extend sub_directories = strcat(domain_split[-3], " ", domain_split[-4])
| where sub_directories has_any(cloudApiTerms)
//Based on sample communications the subdomain is always between 20 and 30 bytes
| where strlen(domain_split) < 32 or strlen(domain_split) > 20
| extend domain = strcat(tostring(domain_split[-2]), ".", tostring(domain_split[-1]))
| extend subdomain_no = countof(sub_domain, @"(\d)", "regex")
| extend subdomain_ch = countof(sub_domain, @"([a-z])", "regex")
| where subdomain_no > 1
| extend percentage_numerical = toreal(subdomain_no) / toreal(strlen(sub_domain)) * 100
| where percentage_numerical < 50 and percentage_numerical > 5
| summarize count(), make_set(Name), FirstSeen=min(TimeGenerated), LastSeen=max(TimeGenerated) by Name
| order by count_ asc
```

Encoded Domain

In addition we have another query - Solorigate Encoded Domain in URL- that takes the encoding pattern the DGA uses, encodes the domains seen in signin logs and then looks for those patterns in DNS logs. This can help identify other C2 domains using the same encoding scheme.

Spoiler

```
let dictionary =
dynamic(["r","q","3","g","s","a","l","t","6","u","1","i","y","f","z","o","p","5","7","2","d","4","9","b","n","x","8","c","v","m","k","e","w","h","j"]);
let regex_bad_domains = SigninLogs
//Collect domains from tenant from signin logs
| where TimeGenerated > ago(1d)
| extend domain = tostring(split(UserPrincipalName, "@", 1)[0])
| where domain != ""
| summarize by domain
| extend split_domain = split(domain, ".")
//This cuts back on domains such as na.contoso.com by electing not to match on the "na" portion
| extend target_string = iff(strlen(split_domain[0]) <= 2, split_domain[1], split_domain[0])
| extend target_string = split(target_string, "-")
| mv-expand target_string
//Rip all of the alphanumeric out of the domain name
| extend string_chars = extract_all(@"([a-z0-9])", tostring(target_string))
//Guid for tracking our data
| extend guid = new_guid()
//Expand to get all of the individual chars from the domain
| mv-expand string_chars
| extend chars = tostring(string_chars)
//Conduct computation to encode the domain as per actor spec
| extend computed_char = array_index_of(dictionary, chars)
| extend computed_char = dictionary[(computed_char + 4) % array_length(dictionary)]
| summarize make_list(computed_char) by guid, domain
| extend target_encoded = tostring(strcat_array(list_computed_char, ""))
//These are probably too small, but can be edited (expect FP's when going too small)
| where strlen(target_encoded) > 5
```

```
| distinct target_encoded
| summarize make_set(target_encoded)
//Key to join to DNS
| extend key = 1;
DnsEvents
| where TimeGenerated > ago(1d)
| summarize by Name
| extend key = 1
//For each DNS query join the malicious domain list
| join kind=inner (
regex_bad_domains
) on key
| project-away key
//Expand each malicious key for each DNS query observed
| mv-expand set_target_encoded
//IndexOf allows us to fuzzy match on the substring
| extend match = indexof(Name, set_target_encoded)
| where match > -1
```

Security Service Tampering

Updated 01/19/2021

There has been additional indication that security services are being tampered with to hinder detection and investigation. While this is a common tactic, we felt that we should include this reference. The query is currently written specifically for Potential Microsoft security services tampering, but can easily be adapted to identify other security services.

Spoiler

```
let includeProc = dynamic(["sc.exe","net1.exe","net.exe", "taskkill.exe", "cmd.exe", "powershell.exe"]);
let action = dynamic(["stop","disable", "delete"]);
let service1 = dynamic(['sense', 'windefend', 'mssecflt']);
let service2 = dynamic(['sense', 'windefend', 'mssecflt', 'healthservice']);
let params1 = dynamic(["-DisableRealtimeMonitoring", "-DisableBehaviorMonitoring" ,"-DisableIOAVProtection"]);
let params2 = dynamic(["sgrmbroker.exe", "mssense.exe"]);
let regparams1 = dynamic(['reg add "HKLM\\SOFTWARE\\Policies\\Microsoft\\Windows Defender"', 'reg add
"HKLM\\SOFTWARE\\Policies\\Microsoft\\Windows Advanced Threat Protection"']);
let regparams2 = dynamic(['ForceDefenderPassiveMode', 'DisableAntiSpyware']);
let regparams3 = dynamic(['sense', 'windefend']);
let regparams4 = dynamic(['demand', 'disabled']);
let regparams5 = dynamic(['reg add "HKLM\\SYSTEM\\CurrentControlSet\\services\\HealthService"', 'reg add
"HKLM\\SYSTEM\\CurrentControlSet\\Services\\Sense"', 'reg add "HKLM\\SYSTEM\\CurrentControlSet\\Services\\WinDefend"', 'reg add
"HKLM\\SYSTEM\\CurrentControlSet\\Services\\MsSecFlt"', 'reg add "HKLM\\SYSTEM\\CurrentControlSet\\Services\\DiagTrack"', 'reg add
"HKLM\\SYSTEM\\CurrentControlSet\\Services\\SgrmBroker"', 'reg add "HKLMSYSTEM\\CurrentControlSet\\Services\\SgrmAgent"', 'reg add
"HKLM\\SYSTEM\\CurrentControlSet\\Services\\AATPSensorUpdater"' , 'reg add
"HKLM\\SYSTEM\\CurrentControlSet\\Services\\AATPSensor"', 'reg add "HKLM\\SYSTEM\\CurrentControlSet\\Services\\mpssvc"']);
let regparams6 = dynamic(['/d 4','/d "4"','/d 0x00000004']);
let regparams7 = dynamic(['/d 1','/d "1"','/d 0x00000001']);
let timeframe = 1d;
(union isfuzzy=true
(
SecurityEvent
| where TimeGenerated >= ago(timeframe)
| where EventID == 4688
| extend ProcessName = tostring(split(NewProcessName, '\\')[-1])
| where ProcessName in~ (includeProc)
| where (CommandLine has_any (action) and CommandLine has_any (service1))
or (CommandLine has_any (params1) and CommandLine has 'Set-MpPreference' and CommandLine has '$true')
or (CommandLine has_any (params2) and CommandLine has "/IM")
or (CommandLine has_any (regparams5) and CommandLine has 'Start' and CommandLine has_any (regparams6))
or (CommandLine has_any (regparams1) and CommandLine has_any (regparams2) and CommandLine has_any (regparams7))
or (CommandLine has "start" and CommandLine has "config" and CommandLine has_any (regparams3) and CommandLine has_any
(regparams4))
| project TimeGenerated, Computer, Account, AccountDomain, ProcessName, ProcessNameFullPath = NewProcessName, EventID, Activity,
CommandLine, EventSourceName, Type
),
```

```
(
Event
| where TimeGenerated >= ago(timeframe)
| where Source =~ "Microsoft-Windows-SENSE"
| where EventID == 87 and ParameterXml in ("<Param>sgrmbroker</Param>", "<Param>WinDefend</Param>")
| project TimeGenerated, Computer, Account = UserName, EventID, Activity = RenderedDescription, EventSourceName = Source, Type
),
(
DeviceProcessEvents
| where TimeGenerated >= ago(timeframe)
| where InitiatingProcessFileName in~ (includeProc)
| where (InitiatingProcessCommandLine has_any(action) and InitiatingProcessCommandLine has_any (service2) and
InitiatingProcessParentFileName != 'cscript.exe')
or (InitiatingProcessCommandLine has_any (params1) and InitiatingProcessCommandLine has 'Set-MpPreference' and
InitiatingProcessCommandLine has '$true')
or (InitiatingProcessCommandLine has_any (params2) and InitiatingProcessCommandLine has "/IM")
or ( InitiatingProcessCommandLine has_any (regparams5) and InitiatingProcessCommandLine has 'Start' and InitiatingProcessCommandLine
has_any (regparams6))
or (InitiatingProcessCommandLine has_any (regparams1) and InitiatingProcessCommandLine has_any (regparams2) and
InitiatingProcessCommandLine has_any (regparams7))
or (InitiatingProcessCommandLine has_any("start") and InitiatingProcessCommandLine has "config" and InitiatingProcessCommandLine
has_any (regparams3) and InitiatingProcessCommandLine has_any (regparams4))
| extend Account = iff(isnotempty(InitiatingProcessAccountUpn), InitiatingProcessAccountUpn, InitiatingProcessAccountName), Computer =
DeviceName
| project TimeGenerated, Computer, Account, AccountDomain, ProcessName = InitiatingProcessFileName, ProcessNameFullPath =
FolderPath, Activity = ActionType, CommandLine = InitiatingProcessCommandLine, Type, InitiatingProcessParentFileName
)
)
| extend timestamp = TimeGenerated, AccountCustomEntity = Account, HostCustomEntity = Computer
```

Microsoft M365 Defender + Azure Sentinel detection correlation

In addition we have created a query in Azure Sentinel - Solorigate Defender Detections - to collate the range of Defender detections that are now deployed. This query can be used to get an overview of such alerts and the hosts they relate to.

Spoiler

```
DeviceInfo
| extend DeviceName = tolower(DeviceName)
| join (SecurityAlert
| where ProviderName =~ "MDATP"
| extend ThreatName = tostring(parse_json(ExtendedProperties).ThreatName)
| where ThreatName has "Solarigate"
| extend HostCustomEntity = tolower(CompromisedEntity)
| take 10) on $left.DeviceName == $right.HostCustomEntity
| project TimeGenerated, DisplayName, ThreatName, CompromisedEntity, PublicIP, MachineGroup, AlertSeverity, Description,
LoggedOnUsers, DeviceId, TenantId
| extend timestamp = TimeGenerated, IPCustomEntity = PublicIP
```

Conclusion

Additionally, as a cloud native SIEM Azure Sentinel can not only collect raw data from various disparate logs but it also gets alerts from various security products. For example, M365 Defender has a range of alerts for various attack components like SolarWinds malicious binaries, network traffic to the compromised domains, DNS queries for known patterns associated with SolarWinds compromise that can flow into Sentinel. Combining these alerts with other raw logs and additional data sources provides the security team with additional insights as well as a complete picture of nature and the scope of attack.

Appendix

Many of these queries have been incorporated into the related hunting workbook.

List of all Azure Sentinel Queries from each section

Updated 01/15/2021

Spoiler

Gaining a foothold	
SolarWinds Inventory check query	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/MultipleDataSources/SolarWinds...
Solorigate Name Pipe	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SecurityEvent/SolorigateNamedPipe.yam...
Privilege Escalation	
Hosts with new logons	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/HostsWithNewLogo...
New user created and added to the built-in administrators group	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SecurityEvent/UserCreatedAddedToBuilt...
User account added to built in domain local or global group	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SecurityEvent/UserAccountAddedToPrivl...
Tracking Privileged Account Rare Activity	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/MultipleDataSources/TrackingPr...
ADFS Key Extraction	
ADFS DKM Master Key Export	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/MultipleDataSources/ADFS-DKM-MasterKe...
ADFS Key Export (Sysmon)	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SecurityEvent/ADFSKeyExportSysmon.yam...
Entropy for Processes for a given Host	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/ProcessEntropy.y...
Rare processes run by Service accounts	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/RareProcbyServic...
Uncommon processes - bottom 5%	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/uncommon_process...
Azure Active Directory	
Modified domain federation trust settings	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/ADFSDomainTrustMods.yaml
New access credential added to Application or Service Principal	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/NewAppOrServicePrincipalCre...
First access credential added to Application or Service Principal where no credential was present	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/FirstAppOrServicePrincipalC...
Mail.Read Permissions Granted to Application	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/MailPermissionsAddedToAppli...
Suspicious application consent similar to O365 Attack Toolkit	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/MaliciousOAuthApp_O365Attac...
Suspicious application consent for offline access	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/AuditLogs/SuspiciousOAuthApp_OfflineA...
Recon and Remote Execution	
Suspicious enumeration using Adfind tool	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/Suspicious_enume...
Multiple explicit credential usage - 4648 events	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/MultipleExplicit...
Data Access	
Azure Active Directory PowerShell accessing non-AAD resources	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SigninLogs/AzureAADPowerShellAnomaly....
Signins From VPS Providers	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SigninLogs/Signins-From-VPS-Pr...
Data Exfiltration	

Anomalous access to other user's mailboxes	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/OfficeActivity/AnomolousUserAc...
Exchange workflow MailItemsAccessed operation anomaly	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/OfficeActivity/MailItemsAccessedTimeS...
Suspect Mailbox Export on IIS/OWA	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/W3CIISLog/SuspectedMailBoxExpo...
Host Exporting Mailbox and Removing Export	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/SecurityEvent/HostExportingMai...
Non-owner mailbox login activity	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/OfficeActivity/nonowner_Mailbo...
Domain Hunting	
Solorigate Network Beacon	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/MultipleDataSources/Solorigate-Networ...
Solorigate DNS Pattern	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/DnsEvents/Solorigate-DNS-Patte...
Solorigate Encoded Domain in URL	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/DnsEvents/Solorigate-Encoded-D...
Security Service Tampering	
Potential Microsoft security services tampering	https://github.com/Azure/Azure-Sentinel/blob/master/Hunting%20Queries/MultipleDataSources/PotentialM...
M365+Sentinel	
Solorigate Defender Detections	https://github.com/Azure/Azure-Sentinel/blob/master/Detections/SecurityAlert/Solorigate-Defender-Det...

References

Recent Nation-State Cyber Attacks

Behavior:Win32/Solorigate.C!dha threat description - Microsoft Security Intelligence

Customer guidance on recent nation-state cyberattacks

FireEye Advisory: Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple G...

FireEye GitHub page: Sunburst Countermeasures

DHS Directive

SolarWinds Security Advisory

FalconFriday – Fireeye Red Team Tool Countermeasures KQL Queries

Microsoft-365-Defender-Hunting-Queries: Sample queries for Advanced hunting in Microsoft 365 Defende...

Azure Sentinel SolarWinds Post Compromise Hunting Workbook

Azure Sentinel SolarWinds Post Compromise Notebook

Updated 12/18/2020

New Threat analytics report shares the latest intelligence on recent nation-state cyber attacks - Mi...

Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Mic...

Updated 12/28/2020

Using Microsoft 365 Defender to protect against Solorigate - Microsoft Security