Iran-linked DEV-0343 targeting defense, GIS, and maritime sectors

microsoft.com/security/blog/2021/10/11/iran-linked-dev-0343-targeting-defense-gis-and-maritime-sectors/

October 11, 2021

DEV-0343 is a new activity cluster that the Microsoft Threat Intelligence Center (MSTIC) first observed and began tracking in late July 2021. MSTIC has observed DEV-0343 conducting extensive password spraying against more than 250 Office 365 tenants, with a focus on US and Israeli defense technology companies, Persian Gulf ports of entry, or global maritime transportation companies with business presence in the Middle East. Less than 20 of the targeted tenants were successfully compromised, but DEV-0343 continues to evolve their techniques to refine its attacks. MSTIC noted that Office 365 accounts with multifactor authentication (MFA) enabled are resilient against password sprays.

Microsoft uses DEV-#### designations as a temporary name given to an unknown, emerging, or a developing cluster of threat activity, allowing MSTIC to track it as a unique set of information until they can reach high confidence about the origin or identity of the actor behind the operation. Once it meets the criteria, a DEV is converted to a named actor. As with any observed nation state actor activity, Microsoft has directly notified customers that have been targeted or compromised, providing them with the information they need to secure their accounts.

Targeting in this DEV-0343 activity has been observed across defense companies that support United States, European Union, and Israeli government partners producing military-grade radars, drone technology, satellite systems, and emergency response communication systems. Further activity has targeted customers in geographic information systems (GIS), spatial analytics, regional ports of entry in the Persian Gulf, and several maritime and cargo transportation companies with a business focus in the Middle East.

This activity likely supports the national interests of the Islamic Republic of Iran based on pattern-of-life analysis, extensive crossover in geographic and sectoral targeting with Iranian actors, and alignment of techniques and targets with another actor originating in Iran. Microsoft assesses this targeting supports Iranian government tracking of adversary security services and maritime shipping in the Middle East to enhance their contingency plans. Gaining access to commercial satellite imagery and proprietary shipping plans and logs could help Iran compensate for its developing satellite program. Given Iran's past cyber and military attacks against shipping and maritime targets, Microsoft believes this activity increases the risk to companies in these sectors, and we encourage our customers in these industries and geographic regions to review the information shared in this blog to defend themselves from this threat.

DEV-0343 conducts extensive password sprays emulating a Firefox browser and using IPs hosted on a Tor proxy network. They are most active between Sunday and Thursday between 7:30 AM and 8:30 PM Iran Time (04:00:00 and 17:00:00 UTC) with significant drop-offs in activity before 7:30 AM and after 8:30 PM Iran Time. They typically target dozens to hundreds of accounts within an organization, depending on the size, and enumerate each account from dozens to thousands of times. On average, between 150 and 1,000+ unique Tor proxy IP addresses are used in attacks against each organization.

DEV-0343 operators typically target two Exchange endpoints – Autodiscover and ActiveSync – as a feature of the enumeration/password spray tool they use. This allows DEV-0343 to validate active accounts and passwords, and further refine their password spray activity.

Observed behaviors

DEV-0343 uses an elaborate series of Tor IP addresses to obfuscate their operational infrastructure. Because of this, there are no static set of indicators of compromise (IOCs) for us to share tied to this activity. The list below provides a series of behaviors and tactics we have observed being used by the attackers. We encourage our customers to use this information to look for similar patterns in logs and network activity to identify areas for further investigation.

- Extensive inbound traffic from Tor IP addresses for password spray campaigns
- Emulation of FireFox (most common) or Chrome browsers in password spray campaigns
- Enumeration of Exchange ActiveSync (most common) or Autodiscover endpoints

- Use of enumeration/password spray tool similar to the ‘o365spray’ tool hosted at https://github.com/0xZDH/o365spray
- Use of Autodiscover to validate accounts and passwords
- Observed password spray activity commonly peaking between 04:00:00 and 11:00:00 UTC

Recommended defenses

The following guidance can mitigate the techniques described in the threat activity:

- Enable multifactor authentication to mitigate compromised credentials.
  - For Office 365 users, see multifactor authentication support.
  - For Consumer and Personal email accounts, see how to use two step verification.
- Microsoft strongly encourages all customers to download and use passwordless solutions like Microsoft Authenticator to secure accounts.
- Review and enforce recommended Exchange Online access policies.
  Block ActiveSync clients from bypassing Conditional Access policies.
- Block all incoming traffic from anonymizing services where possible.

Advanced hunting queries

Microsoft 365 Defender

To locate related activity, run the following advanced hunting queries in Microsoft 365 Defender:

```
AlertInfo
| where Title in~('Unusual sequence of failed logons to Exchange services',
'Unusual sequence of failed logons',
'Password spraying')
| join AlertEvidence on AlertId
```

Azure Sentinel

Azure Sentinel customers can use the following detection queries to look for this activity:

The query below identifies evidence of password sprays activity where ClientAppUsed is either Exchange ActiveSync or Autodiscover and emulated browser is Chrome or Firefox. The query is leveraging Azure AD data to look for failures from multiple accounts from the same IP address within a time window. Details on whether there were successful authentications by the IP address within the time window are also included. This can be an indicator that an attack was successful. The default failure account threshold is 5 and the default time window for failures is 20m.

```
let timeRange = 3d;
let lookBack = 7d;
let authenticationWindow = 20m;
let authenticationThreshold = 5;
let isGUID = "[0-9a-z]{8}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{12}";
let failureCodes = dynamic([50053, 50126]); // invalid password, account is
locked - too many sign ins, expired password
let successCodes = dynamic([0, 50055, 50057, 50155, 50105, 50133, 50005,
50076, 50079, 50173, 50158, 50072, 50074, 53003, 53000, 53001, 50129]);
let ClientApps = dynamic(["AutoDiscover","Exchange ActiveSync"]);
let BrowserList = dynamic(["Chrome","Firefox "]);
// Lookup up resolved identities from last 7 days
let aadFunc = (tableName:string){
let identityLookup = table(tableName)
| where TimeGenerated >= ago(lookBack)
| where not(Identity matches regex isGUID)
| where isnotempty(UserId)
| summarize by UserId, lu_UserDisplayName = UserDisplayName,
lu_UserPrincipalName = UserPrincipalName, Type;
// collect window threshold breaches
table(tableName)
| where TimeGenerated > ago(timeRange)
| where ResultType in(failureCodes)
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated),
make_set(ClientAppUsed), count() by bin(TimeGenerated,
authenticationWindow), IPAddress, AppDisplayName, UserPrincipalName, Type
| summarize FailedPrincipalCount = dcount(UserPrincipalName) by
bin(TimeGenerated, authenticationWindow), IPAddress, AppDisplayName, Type
| where FailedPrincipalCount >= authenticationThreshold
| summarize WindowThresholdBreaches = count() by IPAddress, Type
| join kind= inner (
// where we breached a threshold, join the details back on all failure data
table(tableName)
| where TimeGenerated > ago(timeRange)
| where ResultType in(failureCodes)
| extend LocationDetails = todynamic(LocationDetails)
| extend FullLocation = strcat(LocationDetails.countryOrRegion,'|',
LocationDetails.state, '|', LocationDetails.city)
| extend DeviceDetail = todynamic(DeviceDetail)
| extend Browser = DeviceDetail.browser
| summarize StartTime = min(TimeGenerated), EndTime = max(TimeGenerated),
make_set(ClientAppUsed), make_set(FullLocation), make_set(Browser),
FailureCount = count() by IPAddress, AppDisplayName, UserPrincipalName,
UserDisplayName, Identity, UserId, Type
// lookup any unresolved identities
| extend UnresolvedUserId = iff(Identity matches regex isGUID, UserId, "")
| join kind= leftouter (
```

```
identityLookup
) on $left.UnresolvedUserId==$right.UserId
| extend UserDisplayName=iff(isempty(lu_UserDisplayName), UserDisplayName,
lu_UserDisplayName)
| extend UserPrincipalName=iff(isempty(lu_UserPrincipalName),
UserPrincipalName, lu_UserPrincipalName)
| summarize StartTime = min(StartTime), EndTime = max(EndTime),
make_set(UserPrincipalName), make_set(UserDisplayName),
make_set(set_ClientAppUsed), make_set(set_Browser),
make_set(set_FullLocation), make_list(FailureCount) by IPAddress,
AppDisplayName, Type
| extend FailedPrincipalCount = arraylength(set_UserPrincipalName)
) on IPAddress
| project IPAddress, StartTime, EndTime, TargetedApplication=AppDisplayName,
FailedPrincipalCount, UserPrincipalNames=set_UserPrincipalName,
UserDisplayNames=set_UserDisplayName, ClientAppUsed=set_set_ClientAppUsed,
Locations=set_set_FullLocation, FailureCountByPrincipal=list_FailureCount,
WindowThresholdBreaches, Type, Browsers = set_set_Browser
| join kind= inner (
table(tableName) // get data on success vs. failure history for each IP
| where TimeGenerated > ago(timeRange)
| where ResultType in(successCodes) or ResultType in(failureCodes) //
success or failure types
| summarize GlobalSuccessPrincipalCount = dcountif(UserPrincipalName,
(ResultType in(successCodes))), ResultTypeSuccesses =
make_set_if(ResultType, (ResultType in(successCodes))),
GlobalFailPrincipalCount = dcountif(UserPrincipalName, (ResultType
in(failureCodes))), ResultTypeFailures = make_set_if(ResultType, (ResultType
in(failureCodes))) by IPAddress, Type
| where GlobalFailPrincipalCount > GlobalSuccessPrincipalCount // where the
number of failed principals is greater than success - eliminates FPs from
IPs who authenticate successfully alot and as a side effect have alot of
failures
) on IPAddress
| project-away IPAddress1
| extend timestamp=StartTime, IPCustomEntity = IPAddress
};
let aadSignin = aadFunc("SigninLogs");
let aadNonInt = aadFunc("AADNonInteractiveUserSignInLogs");
union isfuzzy=true aadSignin, aadNonInt
| where Browsers has_any (BrowserList)
| where ClientAppUsed has_any (ClientApps)
```

One of the results that the query surfaces is the IPAddress field from where the sign-in originated. Customers can leverage their threat intel data that have details about the TOR exit nodes to join with this query and make it even higher fidelity. It is often worthwhile to have a list of all the known TOR exit nodes so that these could be used for matching with

queries of Azure Sentinel, or to block sign-ins from the TOR exit nodes using conditional access. Azure Sentinel also provides playbooks that can leverage third party providers of TOR information like Big Data Cloud to synchronize the list of known TOR exit nodes on an hourly basis. Here is the link to one such playbook: https://github.com/Azure/Azure-Sentinel/blob/master/Playbooks/Update-NamedLocations-TOR/readme.md.

Next, we have another hunting query that identifies instances where a single user account has seen a high incidence of failed attempts from highly volatile IP addresses. Changing the IP address for every password attempt is becoming a more common technique among sophisticated threat groups. Often, threat groups randomize the user agent they are using as well as IP address. This technique has been enabled by the emergence of services providing huge numbers of residential IP addresses. These services are often enabled through malicious browser plugins. This query is best executed over longer timeframes. Results with the highest “IPs”, “Failures” and “DaysWithAttempts” are good candidates for further investigation. This query intentionally does not cluster on UserAgent, IP, etc. This query is clustering on the highly volatile IP behavior.

```
 let timeRange = 14d;
let UnsuccessfulLoginCountryThreshold = 5; // Number of failed countries
attempting to login, good way to filter.
let ClientApps = dynamic(["AutoDiscover","Exchange ActiveSync"]);
let BrowserList = dynamic(["Chrome","Firefox "]);
SigninLogs
| where TimeGenerated > ago(timeRange)
// Limit to username/password failure errors, most common when
bruteforcing/spraying
| where ResultType has_any("50126", "50053")
//Narrowing the result even further to clientapps and browser that are seen
in this attack.
| where ClientAppUsed has_any (ClientApps)
| extend Browser = tostring(DeviceDetail.browser)
| where Browser has_any (BrowserList)
// Find instances where an IP has only been used once
| summarize IPLogins=count(), make_list(TimeGenerated) by IPAddress,
Location, UserPrincipalName
| where IPLogins == 1
// We only keep instances where there is 1 event, so we know there will only
be one datetime in the list
| extend LoginAttemptTime =
format_datetime(todatetime(list_TimeGenerated[0]), 'dd-MM-yyyy')
// So far we've only collected failures, we join back to the log to ensure
there were no successful logins from the IP
| join kind=leftouter (
SigninLogs
| where TimeGenerated > ago(timeRange)
| where ResultType == 0
```

```
| summarize count() by IPAddress, UserPrincipalNameSuccess=UserPrincipalName
) on $left.IPAddress == $right.IPAddress
// Where there have been fewer than 2 successful logins from the IP
| where count_ < 2 or isempty(count_)
// Confirm that the result is for the same account where possible
| where UserPrincipalName == UserPrincipalNameSuccess or
isempty(UserPrincipalNameSuccess)
// Summarize the collected details around the users email address
| summarize IPs=dcount(IPAddress),
UnsuccessfulLoginCountryCount=dcount(Location), make_list(IPAddress),
make_list(Location), DaysWithAttempts=dcount(LoginAttemptTime),
Failures=count() by UserPrincipalName
| project UserPrincipalName, Failures, IPs, UnsuccessfulLoginCountryCount,
DaysWithAttempts, IPAddresses=list_IPAddress,
IPAddressLocations=list_Location
// Join back to get countries the user has successfully authenticated from
to compare with failures
| join kind=leftouter (
SigninLogs
| where TimeGenerated > ago(timeRange)
| where ResultType == 0
// If there is no location make the output pretty
| extend Location = iff(isempty(Location), "NODATA", Location)
| summarize SuccessfulLoginCountries=make_set(Location),
SuccessfulLoginCountryCount=dcount(Location) by UserPrincipalName
) on $left.UserPrincipalName == $right.UserPrincipalName
| project-away UserPrincipalName1
| order by UnsuccessfulLoginCountryCount desc
// Calculate the difference between countries with successful vs. failed
logins
| extend IPIncreaseOnSuccess = UnsuccessfulLoginCountryCount -
SuccessfulLoginCountryCount
// The below line can be removed if the actor is using IPs in one country
| where UnsuccessfulLoginCountryCount > UnsuccessfulLoginCountryThreshold
| project UserPrincipalName, Failures, IPs, DaysWithAttempts,
UnsuccessfulLoginCountryCount, UnuccessfulLoginCountries=IPAddressLocations,
SuccessfulLoginCountries, FailureIPAddresses=IPAddresses
```