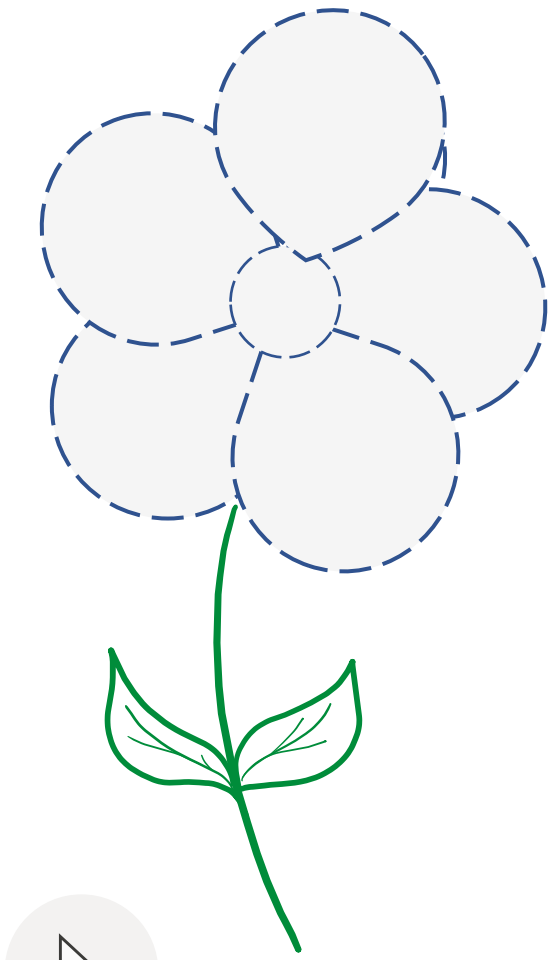
Microsoft 365

Security Detection and threat hunting

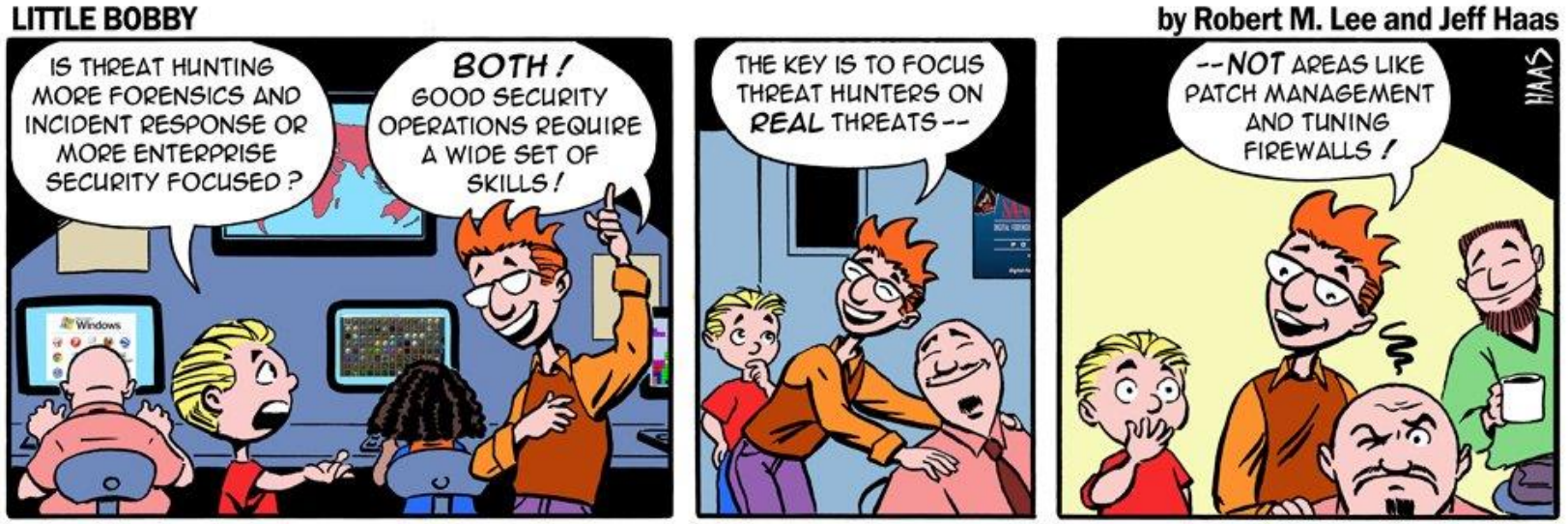
@ali_alwashali

Agenda

1. Threat Detection Maturity
2. TTPs and Attack Simulation
3. Developing Detection rules
4. Hunting with Windows Defender for Endpoints



Threat Hunting?



Threat Detection Maturity

1. Lack visibility
2. Lack of data analytics capabilities
2. Lack of context (intelligence)

Threat Hunting methodology

Hypothesis Generation

Twitter

- Follow Threat Hunters and Detection Engineers

Community rules and Queries

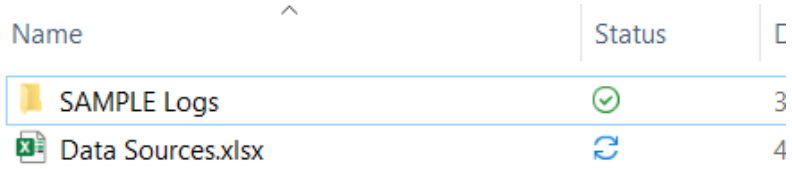
- KQL queries github repos, FalconFriday, reprise99
- Sigma rules

Threat research/Threat intel Blog posts

- Red canary blogs
- The DFIRReports
- Elastic blog
- US-CERT
- Threat research articles

Data Source Identification

- **EDR**
- **NDR**
- **AV**
- **SIEM**
- **PAM**
- **Firewall**
- **..etc**



Data Source	Focus	Not useful /exclude
API - Tenable SecurityCenter		
Syslog - Symantec Endpoint Server	Destinaion port 8080,80,445,443,21 ..etc there is Hash values Subject field (jucey data) Future use cases to include subject	Exclude Classification:information it contains notification about successful log delivery Exclude Classification:Other Audit Success Contains info about successful scan completion Classification Startup and Shutdown
AVI - WAF	USER agent URL Threat Name Signature	
Flat File - Microsoft ActiveSync 2010		
Flat File - Microsoft IIS W3C File	HTTP logs	
Flat File - Microsoft Windows 2012 DNS		
Flat File - MS Exchange 2016 Message Tracking Log	2 log sources Recipient Sender Subject Domain (Impacted) Status (incoming-orginating)	

Data Cleaning

- **Data analytics skills (love data and scripting)**
- **Love Pandas**

```
[94]: dfc.loc[dfc['cases'].str.contains("logon"), 'tag'] = 'Logon'
      dfc.loc[dfc['cases'].str.contains("login"), 'tag'] = 'Logon'
      dfc.loc[dfc['cases'].str.contains("malware"), 'tag'] = 'AV'
      dfc.loc[dfc['cases'].str.contains("defender"), 'tag'] = 'AV'
      dfc.loc[dfc['cases'].str.contains("symantec"), 'tag'] = 'AV'
      dfc.loc[dfc['cases'].str.contains("authentication failure"), 'tag'] = 'Logon'
      dfc.loc[dfc['cases'].str.contains("openssl"), 'tag'] = 'OpenSSL'
      dfc.loc[dfc['cases'].str.contains("injection"), 'tag'] = 'SQLInjection'
      dfc.loc[dfc['cases'].str.contains("scan"), 'tag'] = 'PortScan'
      dfc.loc[dfc['cases'].str.contains("port"), 'tag'] = 'PortScan'
      dfc.loc[dfc['cases'].str.contains("geo"), 'tag'] = 'Logon'
      dfc.loc[dfc['cases'].str.contains("waf"), 'tag'] = 'WAF'
      dfc.loc[dfc['cases'].str.contains("sama"), 'tag'] = 'MaliciousIP'
      dfc.loc[dfc['cases'].str.contains("nca"), 'tag'] = 'MaliciousIP'
      dfc.loc[dfc['cases'].str.contains("suspicious communication"), 'tag'] = 'MaliciousIP'
      dfc.loc[dfc['cases'].str.contains("user added"), 'tag'] = 'UserEvents'
      dfc.loc[dfc['tag'].isnull() , 'tag'] = 'Other'
```

Unstructured Data

```
let timeframe=1h;
let CobaltStrikeDefaults= dynamic([@"msagent_", @"MSSE-", @"postex_", @"status_", @"mypipe-f", @"mypipe-h",@"ntsvcs_",@"scerpc_", @"m
let CobaltStrikeMallable= dynamic([@"win_svc", @"ntsvcs", @"scerpc", @"status_", @"SearchTextHarvester", @"DserNamePipe",@"wkssvc_",@
DeviceEvents
| where Timestamp >= ago(timeframe)
| where ActionType == "NamedPipeEvent"
| extend AdditionalFields=parse_json(AdditionalFields)
| extend ThreadId=tostring(AdditionalFields.ThreadId)
| extend PipeName=tostring(AdditionalFields.PipeName)
// creating string based variants of the processIDs for matching several times later
| extend InitiatingPID=tostring(InitiatingProcessId)
| extend InitiatingParentPID=tostring(InitiatingProcessParentId)
// Customer specific whitelist
// End customer specific whitelist
| where PipeName has_any (CobaltStrikeDefaults) or
// mojo is generated by Chrome(ium) browsers and teams and have distinct pattern including the (parent)ProcessId and ThreadId plus a
       (PipeName matches regex @"\\mojo\.\d+\.\d+\." and not(PipeName matches regex @"\\mojo\.\d+\.\d+\.\d+$" or PipeName has Initia
// chrome(ium) browsers sync processes have distinct pattern including the (parent)ProcessId and ThreadId plus a random character str
       (PipeName matches regex @"\\(edge|chrome)\.sync\.\d+\.\d+\." and not(PipeName matches regex @"\\(edge|chrome|edge\.sync|chrom
// PSHost is generated by PowerShell and has a distinct pattern including the (parent)ProcessId
       (PipeName matches regex @"\\PSHost\.\d+\." and not(PipeName matches regex @"\\PSHost\.\d+\.\d+\." or PipeName has InitiatingP
// crashpad pipes have a distinct pattern including the ProcessId and a string of upper case characters
       (PipeName matches regex @"\\crashpad_" and not(PipeName matches regex @"\\crashpad_\d+_[A-Z]+" or PipeName has InitiatingPID
// firefox pipes have a distinct pattern including the ProcessId and 1-3 digits which are sequential for each new pipe
       (PipeName matches regex @"\\cubeb-pipe-" and not(PipeName matches regex @"\\cubeb-pipe-\d+_[0-9]{1-3}+" or PipeName has Initi
// based on a list of public mallable profiles and a suffix that is a random HEX string
       (PipeName has_any (CobaltStrikeMallable) and PipeName matches regex @"[a-fA-F0-9]{2,10}$") or
       (PipeName matches regex @"\\pipe\\[0-9a-f]{7,10}" or PipeName matches regex @"\\pipe\\[0-9a-f]{8}")
```

Parsing on spot

Hunting Suspicious Named Pipes

Data Cleaning

- **Security Control API interaction**
- **Hunting for threats in "Alerts"**

[3]:
```
import requests
import pandas as pd
```

[228]:
```
offenseURL = 'https://ip/api/siem/offenses'
crURL = 'https://ip/api/siem/offense_closing_reasons'

headers=  {
           'Range': 'items=0-1500','Version': '16.0',
           'Accept': 'application/json',
           'SEC': 'Token'
          }
```

[ ]:
```
#offenses
ofR = requests.get(url = URL,verify=False, headers=headers)
data = ofR.json()

# Closing reason
crR = requests.get(url = crURL,verify=False, headers=headers)
closeingReason = crR.json()
```

[123]:
```
offenses = pd.DataFrame(data)
```

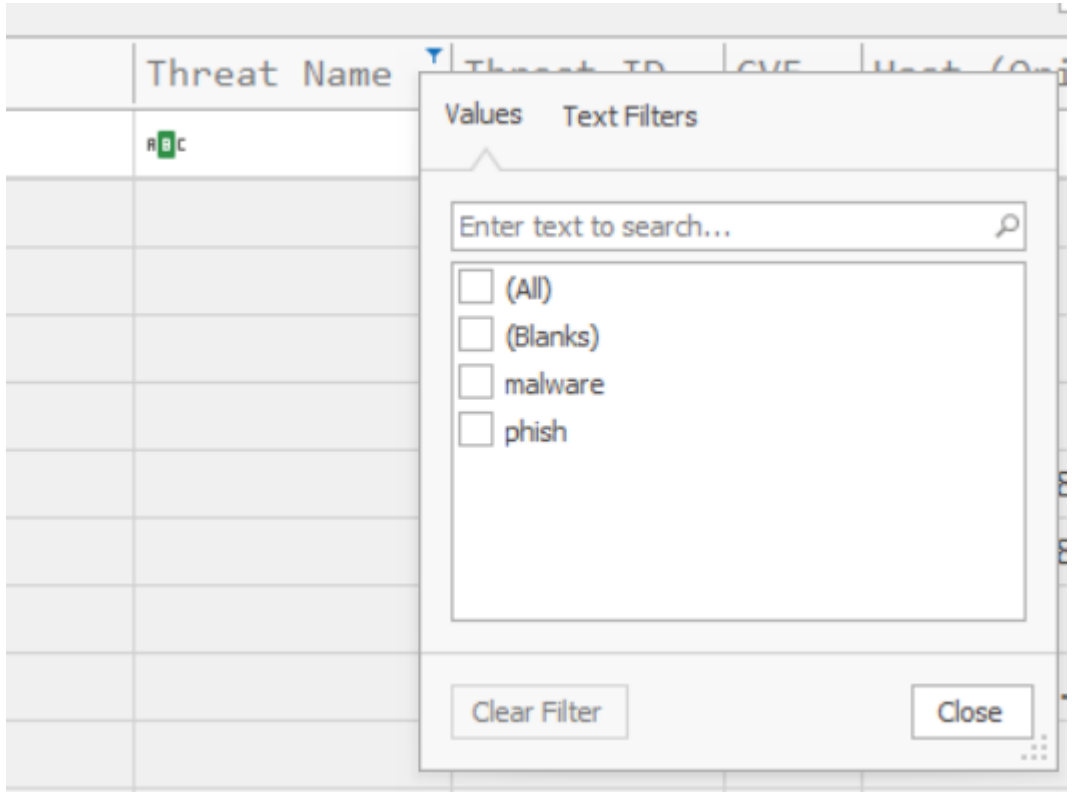
[124]:
```
offenses['offense_time'] = pd.to_datetime(offenses['start_time'],unit='ms')
```

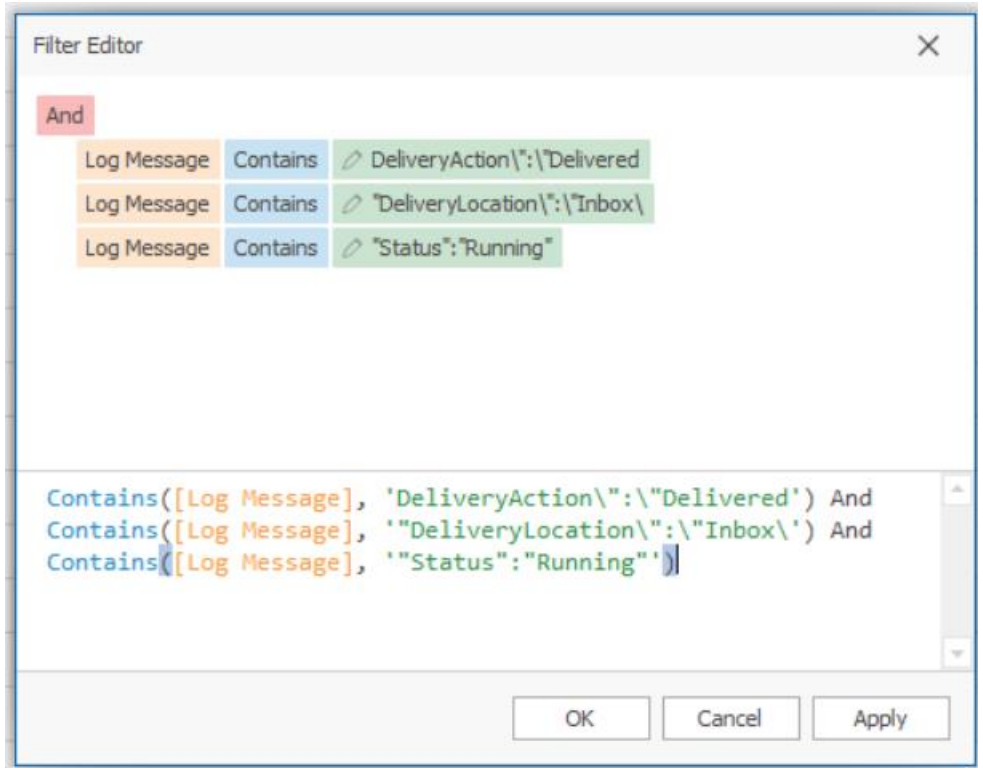
[ ]:
```
offenses.set_index('offense_time')
```

[126]:
```
#offenses.loc[offenses['closing_reason_id'] == 1, 'closing_reason'] = "Non-issue"
#offenses.loc[offenses['closing_reason_id'] == 154, 'closing_reason'] = "Escalated"
#offenses.loc[offenses['closing_reason_id'] == 55, 'closing_reason'] = "False_Positive"
#offenses.loc[offenses['closing_reason_id'] == 104, 'closing_reason'] = "To be monitored"
#offenses.loc[offenses['closing_reason_id'] == 155, 'closing_reason'] = "Reported"
#offenses.loc[offenses['closing_reason_id'] == 2, 'closing_reason'] = "False-Positive, Tuned"
```

Data Cleaning

- **Can be as simple as using excel or Timeline Explorer**

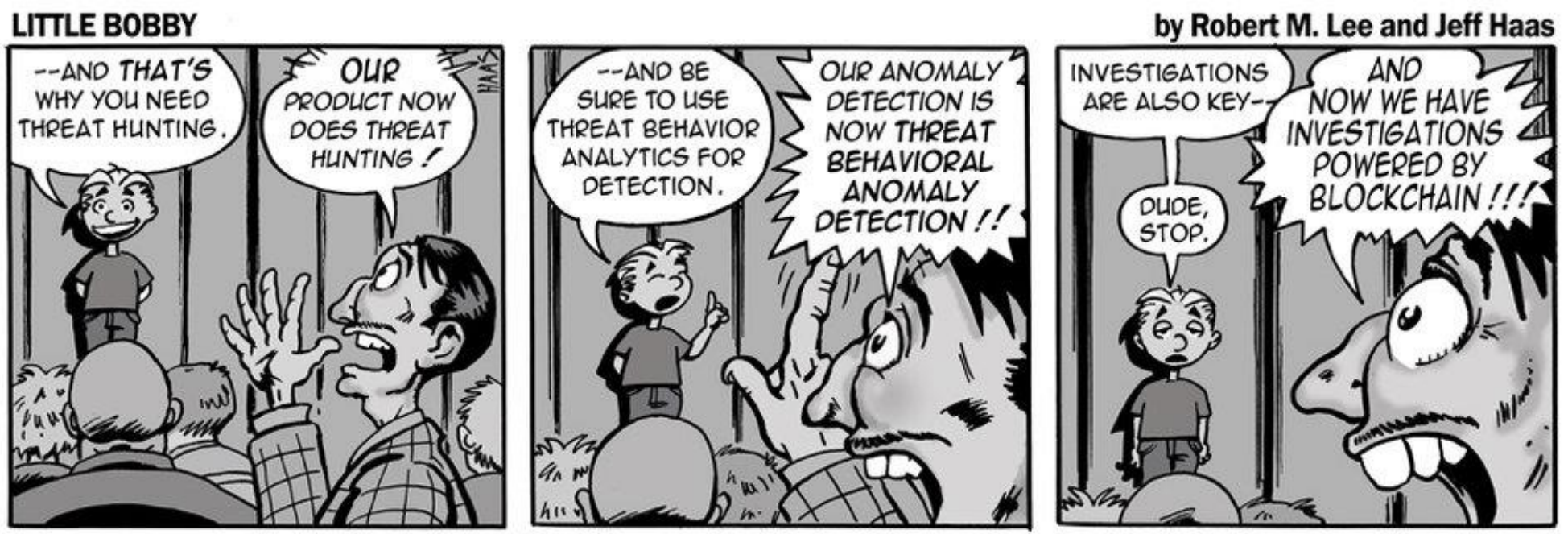




Analysis and threat identification

1. Solid understanding of attackers techniques

2. Up to date knowledge about attack campaigns and 0day vulnerabilities



Security Enhancements

Threat Detection life cycle should result in either **detecting a threat** or **security enhancement**

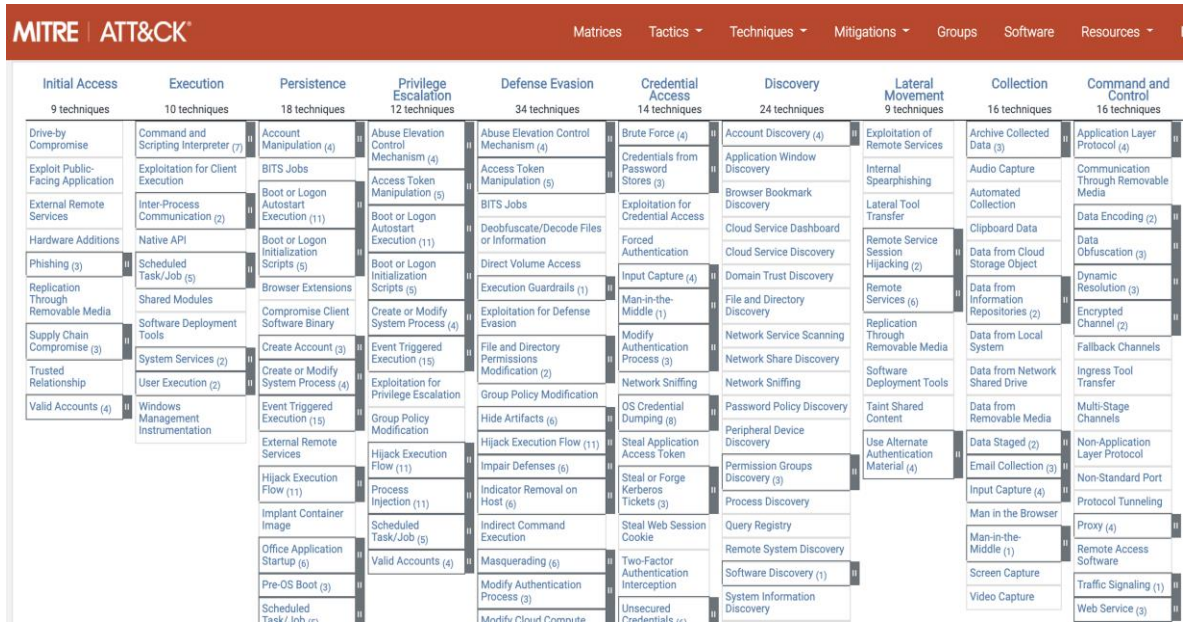
NO.	Attack Technique	Can be prevented	Can be Restriction	Could be monitored
1	Command and Scripting Interpreter		✔	✔
2	Schedule Task Creation			✔
3	Remote Hash Retrieval		✔	✔
4	Account Discovery			✔
5	Accessing Hidden SMB shares		✔	✔

3.1. Use case development

- Remote hash retrieval use case was developed to trigger an alert for same activity.
- Account Discovery behaviour could be monitored if the required data feeds mentioned above are available in SIEM.
- Access to hidden admin shares baseline can be monitored by creating a baseline for the legitimate users.

- Blocking Workstation to Workstation SMB traffic is effective restriction against remote hash retrieval and unauthorize SMB admin share access.
- Use application control where possible.
- On Windows 10, enable Attack Surface Reduction (ASR) rules to prevent Visual Basic and JavaScript scripts from executing potentially malicious downloaded content

TTPs and Attack Simulation



1 **T1059** →
Command and Scripting Interpreter (53.4% of customers affected)

2 **T1218** →
Signed Binary Proxy Execution (34.8%)

3 **T1047** →
Windows Management Instrumentation (15.4%)

4 **T1003** →
Credential Dumping (18.3%)

5 **T1105** →
Ingress Tool Transfer (20.4%)

6 **T1055** →
Process Injection (21.7%)

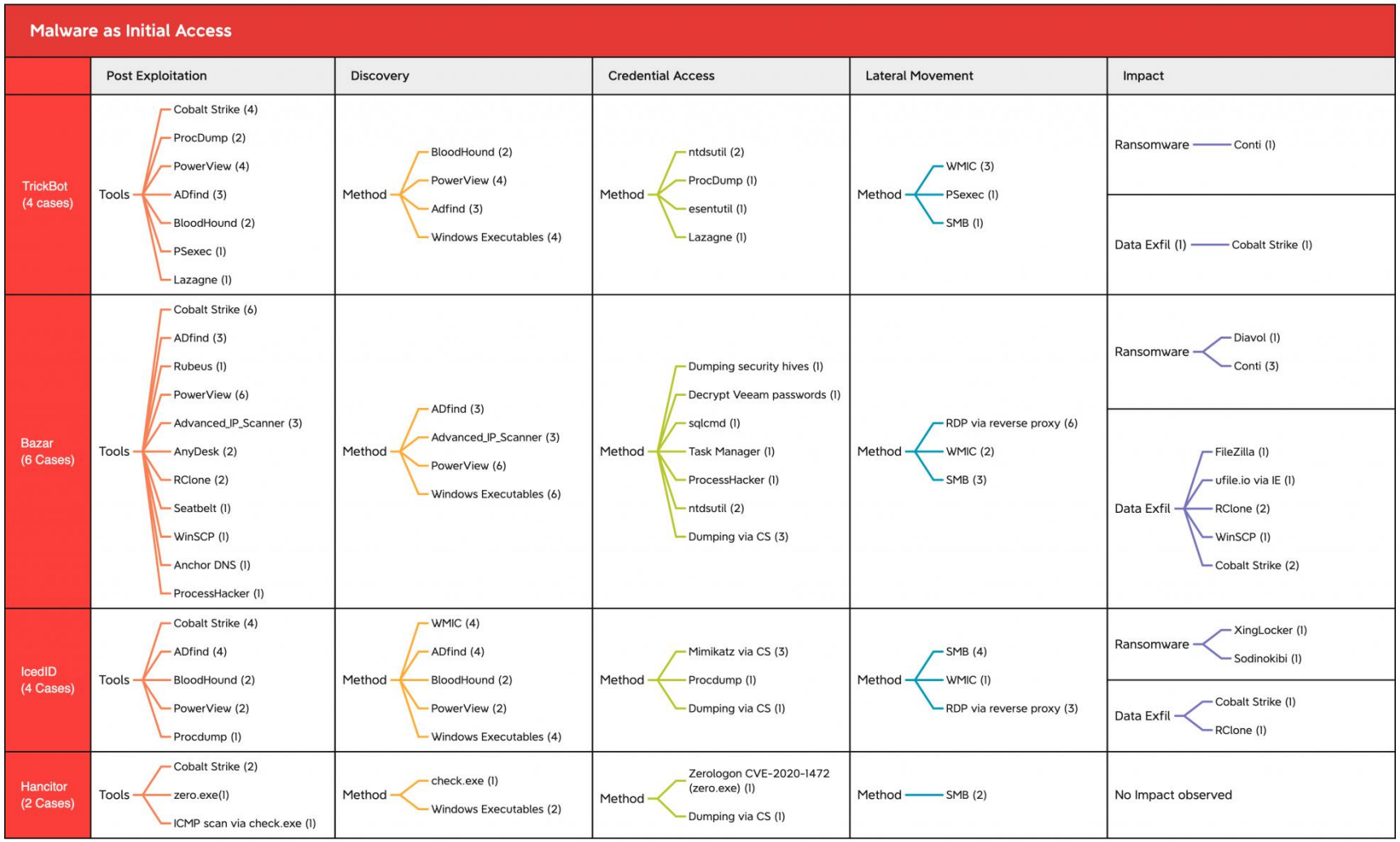
7 **T1053** →
Scheduled Task/Job (14.7%)

8 **T1027** →
Obfuscated Files or Information (19.4%)

9 **T1036** →
Masquerading (22.1%)

10 **T1574** →
Hijack Execution Flow (8.4%)

TTPs and Attack Simulation

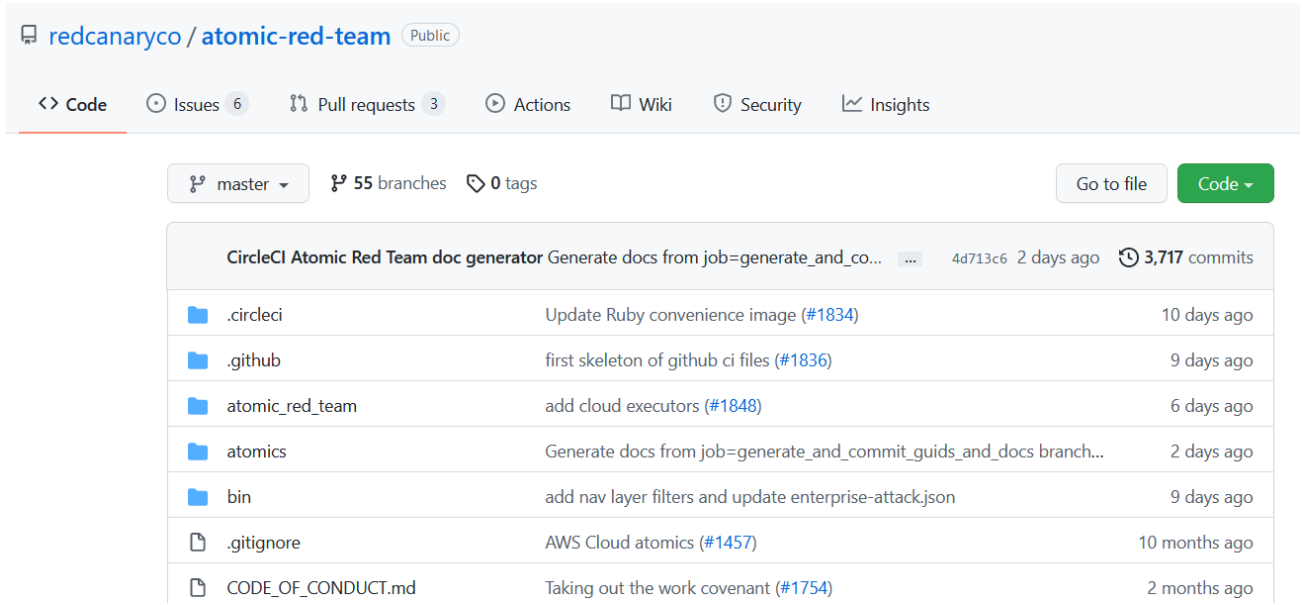


https://thedfirreport.com/2022/03/07/2021-year-in-review/

Detection Lab

1. Telemetry naming convention

2. Test your queries



Developing Detection Rules

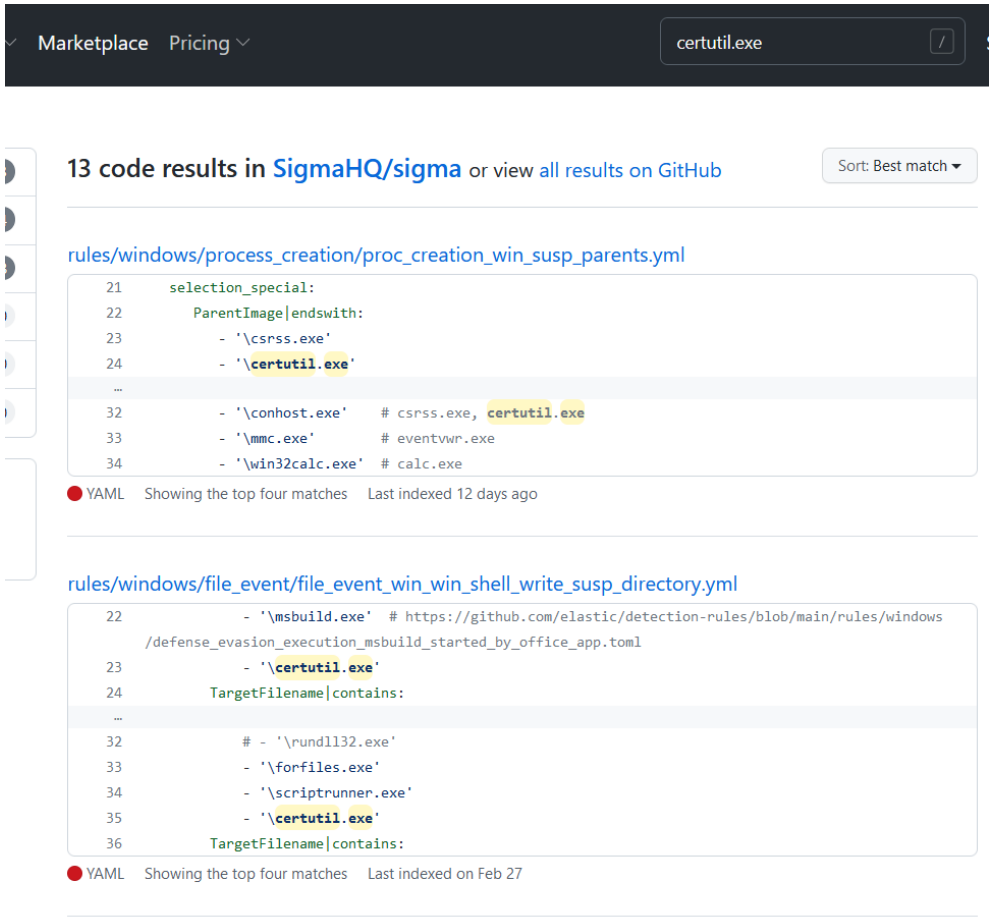
Where to get use cases from

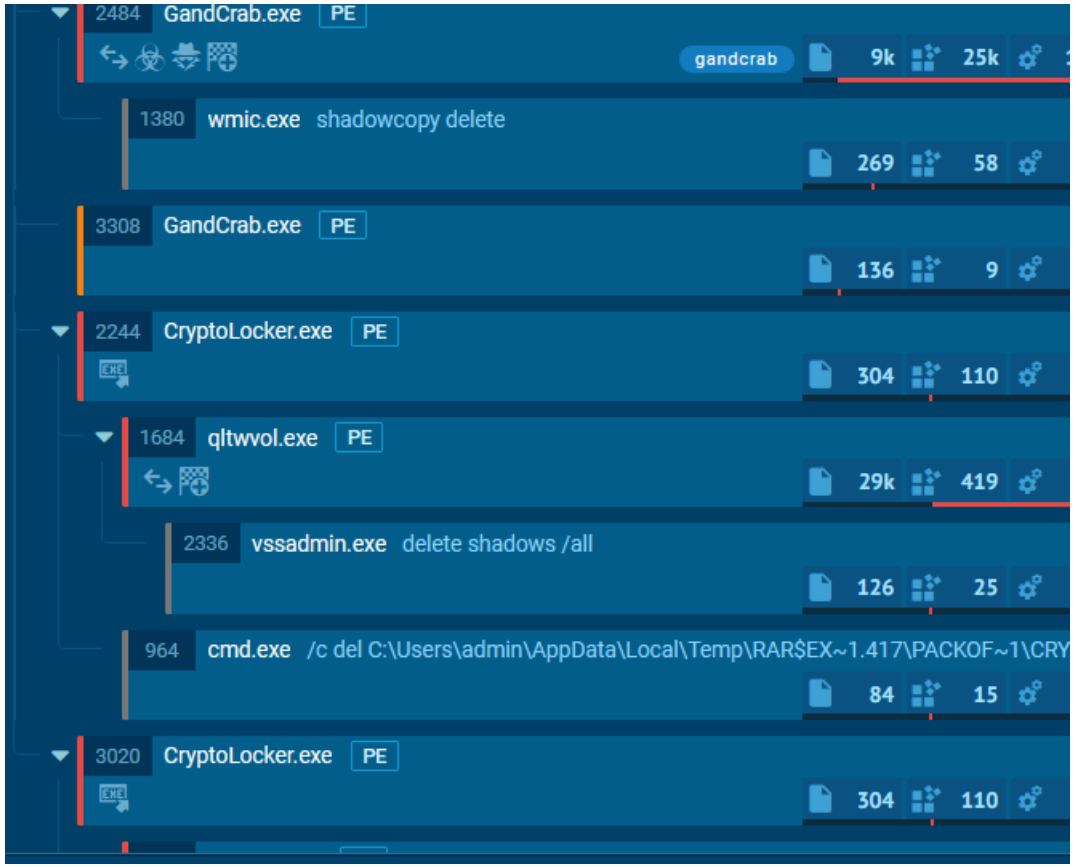
- **Malware Sandbox**
  - **tria.ge,**
  - **https://any.run**
- **Atomic Red teaming**
- **Living Off The Land Binaries**
- **Malicious Command-Line (MAL-CL)**
- **Intelligence blogs**

Detection Rules building blocks
Alert Name
Data sources
Rule Logic
False positive
Playbook
Status
References

Developing Detection Rules

- **When in doubt, search for the TTP behavior !!**

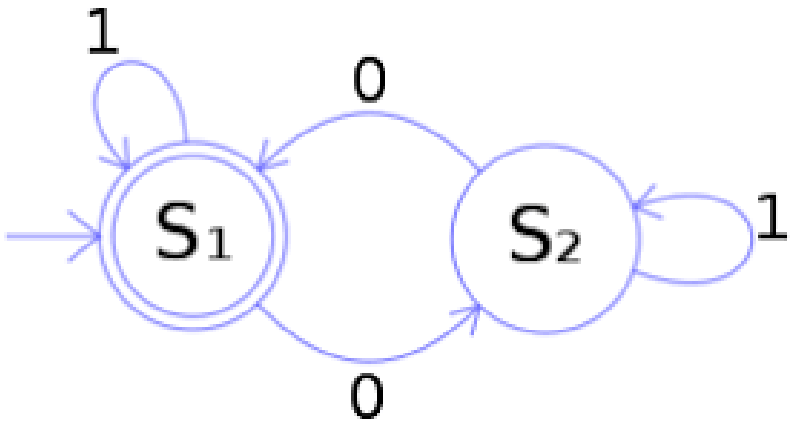


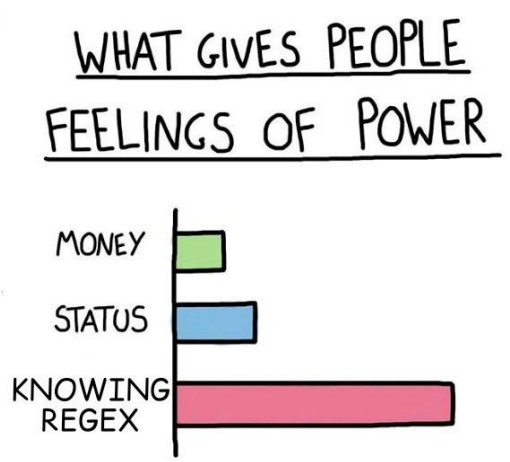


Developing Detection Rules

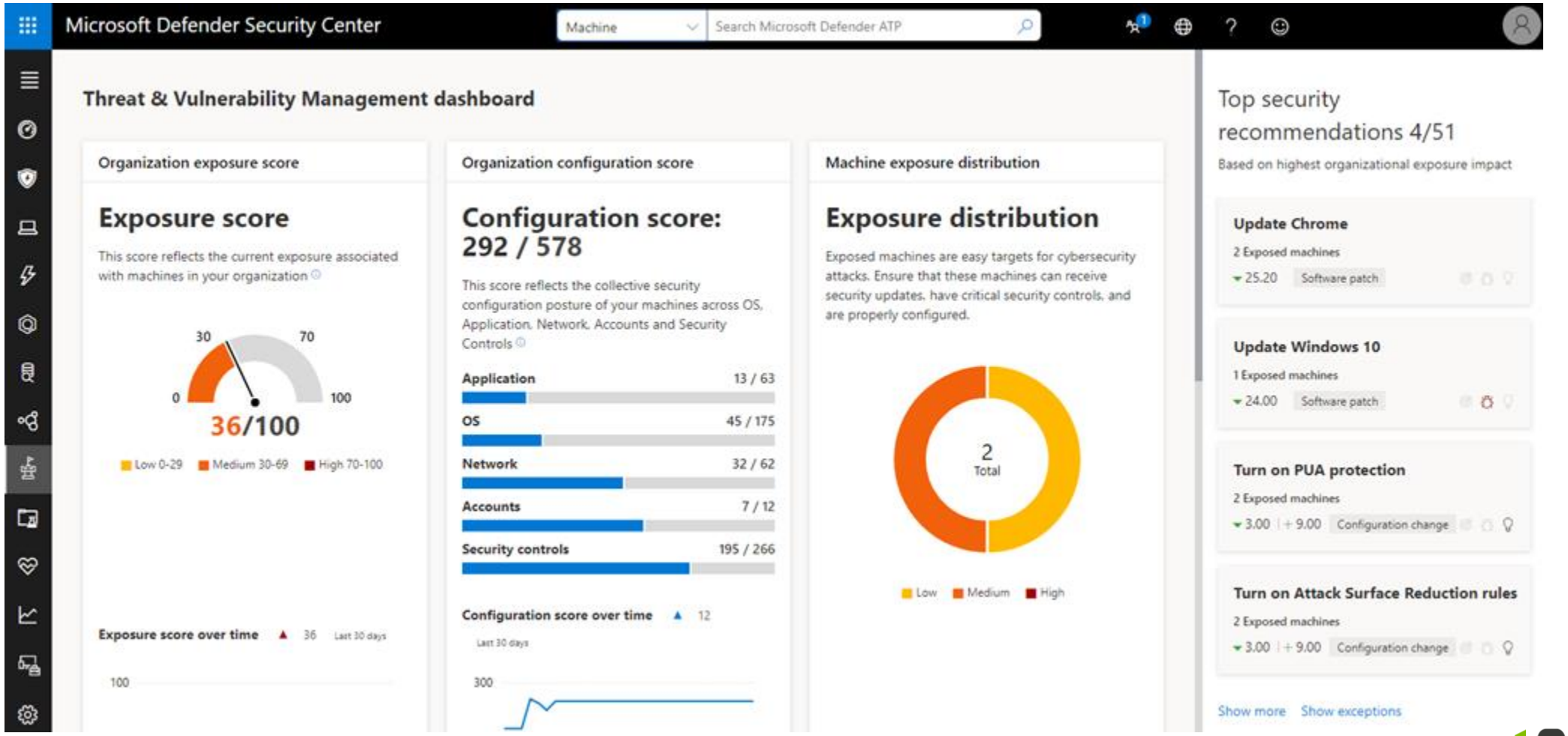
Detection Engineer skills

- **Regex**
- **Data analytics**
- **Attack Simulation**





Windows Defender For Endpoints



Windows Defender For Endpoints

Microsoft Defender
for Endpoint

https://security.microsoft.com
hood@hackdefendlabs.com
MSEDR@123

Windows Defender For Endpoints

EDR VS sysmon

- Live response
- Temper protection
- prevention
- Isolation
- Threat hunting
- Investigation package

https://m365internals.com/2021/05/14/using-microsoft-defender-for-endpoint-during-investigation/amp/

Onbaording

settings

Grouping and settings

- Tagging
- Time Zone
- Tenant ID, Org ID
- Where groups are used
  - web filtering
  - KQL
  - Indicators

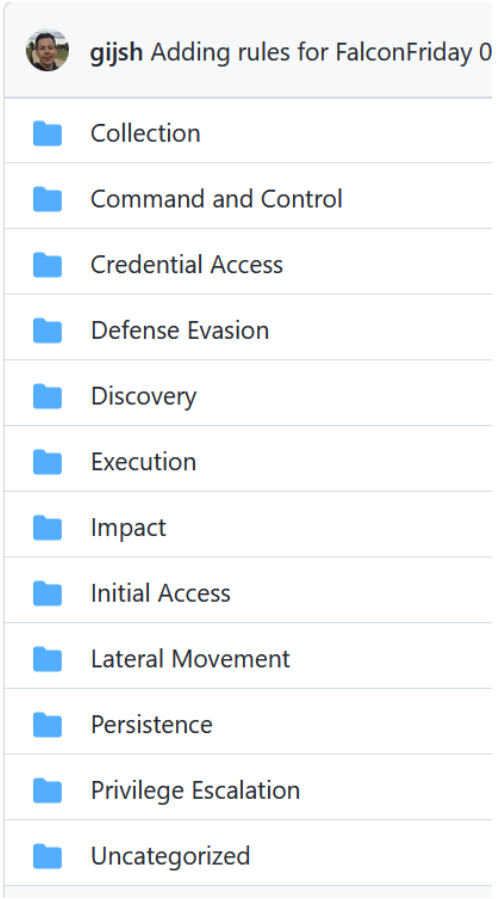
Indicators

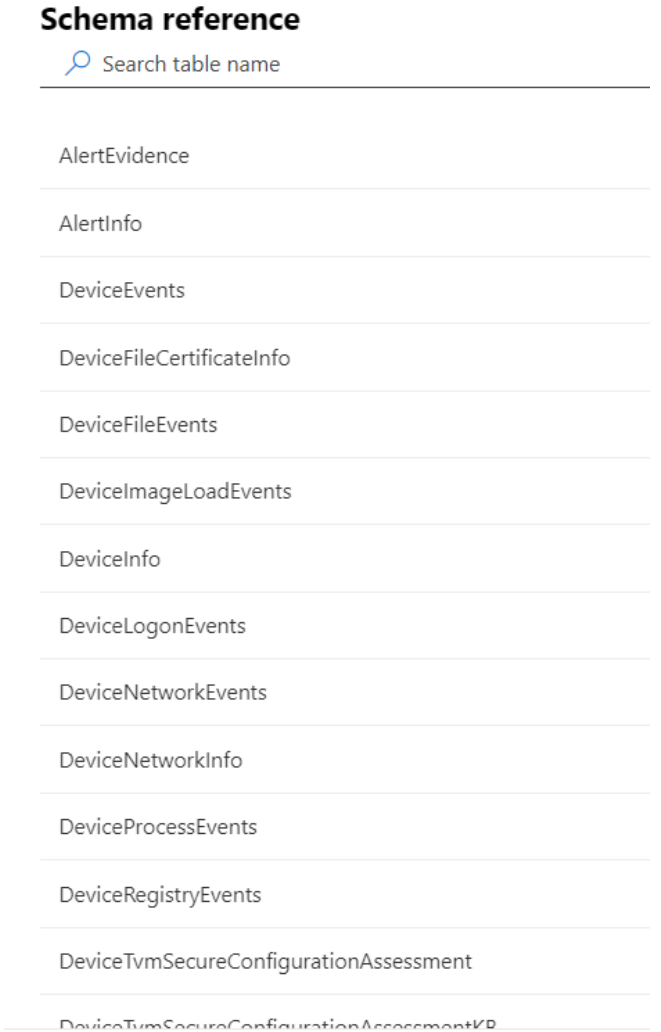
Device Discovery

Live Response

- Upload file to library
- Command history

Threat Hunting wit KQL





https://github.com/FalconForceTeam/FalconFriday

KQL

Kusto Query Language (KQL) overview

Article • 03/07/2022 • 2 minutes to read • 9 contributors

Kusto Query Language is a powerful tool to explore your data and discover patterns, identify anomalies and outliers, create statistical modeling, and more. The query uses schema entities that are organized in a hierarchy similar to SQL's: databases, tables, and columns.

https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/

Advanced data processing

```
DeviceProcessEvents
| where ProcessCommandLine contains "powershell" or InitiatingProcessCommandLine contains "powershell"
| where ProcessCommandLine contains "-enc" or ProcessCommandLine contains "-encodedcommand" or
InitiatingProcessCommandLine contains "-enc" or InitiatingProcessCommandLine contains "-encodedcommand"
| extend EncodedCommand = extract(@'\s+([A-Za-z0-9+/]{20}\S+$)', 1, ProcessCommandLine)
| where EncodedCommand != ""
| extend DecodedCommand = base64_decode_tostring(EncodedCommand)
| where DecodedCommand != ""
| project TimeGenerated, DeviceName, InitiatingProcessAccountName, InitiatingProcessCommandLine, ProcessCommandLine,
EncodedCommand, DecodedCommand
```

```
SecurityAlert
| where AlertName in ("Impossible travel activity", "Atypical Travel", "Anonymous IP address", "Anomalous Token")
| parse Entities with * 'AadUserId": "' aadid_ '",' *
| extend ep_ = parse_json(ExtendedProperties)
| extend s = tostring(ep_["IP Addresses"])
| extend ipv4_ = extract_all(@"(([\d]{1,3}\.){3}[\d]{1,3})", dynamic([1]), s)
| extend ipv4Add_ = translate('["]', '', tostring(ipv4_))
| extend ipv6_ = extract_all(@"(([\d|\w]{1,4}\:){7}[\d|\w]{1,4})", dynamic([1]), s)
| extend ipv6Add_ = translate('["]', '', tostring(ipv6_))
| project TimeGenerated, AlertName, ipv4Add_, ipv6Add_, CompromisedEntity
```

https://learnsentinel.blog/2022/02/28/detecting-malware-kill-chains-with-defender-and-microsoft-sentinel/

Advanced Detection Rules

KQL capabilities enable analyst to think out of the box and detect real threats

```
DeviceEvents
| where TimeGenerated > ago(30d) and TimeGenerated < ago(1h)
| where ActionType == "ScheduledTaskCreated"
| extend ScheduledTaskName = tostring(AdditionalFields.TaskName)
| distinct ScheduledTaskName
| join kind=rightanti
    (DeviceEvents
    | where TimeGenerated > ago(1h)
    | where ActionType == "ScheduledTaskCreated"
    | extend ScheduledTaskName = tostring(AdditionalFields.TaskName)
    | project TimeGenerated, DeviceName, ScheduledTaskName, InitiatingProcessAccountName)
    on ScheduledTaskName
| project TimeGenerated, DeviceName, InitiatingProcessAccountName, ScheduledTaskName
```

RITA Beacon Analyzer

Author: Cyb3rMonk (Medium, Twitter)

Link to Original Post: Medium

Language: Azure KQL

Products: Azure Sentinel

Required: VMConnection

https://learnsentinel.blog/2022/02/28/detecting-malware-kill-chains-with-defender-and-microsoft-sentinel/

https://github.com/Cyb3r-Monk/Threat-Hunting-and-Detection/blob/main/Command%20and%20Control/RITA%20Beacon%20Analyzer.md

Power of functions

FilePorfile()

Kusto Copy

```
DeviceFileEvents
| where ActionType == "FileCreated" and Timestamp > ago(1d)
| project CreatedOn = Timestamp, FileName, FolderPath, SHA1
| invoke FileProfile("SHA1", 500)
| where GlobalPrevalence < 15
```

```
//https://github.com/fireeye/red_team_tool_countermeasures/blob/master/rules/PGF/supplemental/hxioc/ushata.dll Hijack (Methodology).i
//Identifies possible DLL search order hijacking of ushata.dll based on image loads from unexpected locations.
DeviceImageLoadEvents
| where FileName =~ "ushata.dll"
| where not(FolderPath has_any (@"\Program Files (x86)\Kaspersky Lab\Kaspersky",
    @"\Program Files\Kaspersky Lab\Kaspersky",
    @"\Program Files (x86)\LANDesk\LDClient\antivirus\"
))
| invoke FileProfile(SHA1)
```

https://docs.microsoft.com/en-us/microsoft-365/security/defender/advanced-hunting-fileprofile-function?view=o365-worldwide

Power of functions

materialize()
set_has_element()
Parse_command_line()
ipv4_is_private()

```
// removing any potential command line obfuscation
| extend CleanProcessCommandLine=parse_command_line(ProcessCommandLine, "windows")
// search for de-obfuscated commands used
| where CleanProcessCommandLine has_any ("decode", "encode", "verify","url")
// urlcache is the documented attribute, only url is also accepted
// verifyctl is the documented attribute, only verify is also accepted
| order by Timestamp
| project Timestamp, CleanProcessCommandLine, ProcessCommandLine, SHA1
```

```
//
// The following list of lolbins is used to include all results which have a high reputation, but are lolbins
let lolbins = dynamic(["At.exe", "Atbroker.exe", "Bash.exe", "Bitsadmin.exe", "CertReq.exe", "Certutil.exe", "Cmd.exe", "Cmdkey.exe",
// First we want to get all the networkevents triggered by services.exe
let networkEvents = materialize(DeviceNetworkEvents
| where InitiatingProcessFileName in~ ("services.exe")
| where ActionType == "InboundConnectionAccepted"
| project-rename TimestampNetworkAct=Timestamp);
// Next we want to get the list of childprocesses created by services.exe
```

Power of functions

Externaldata()

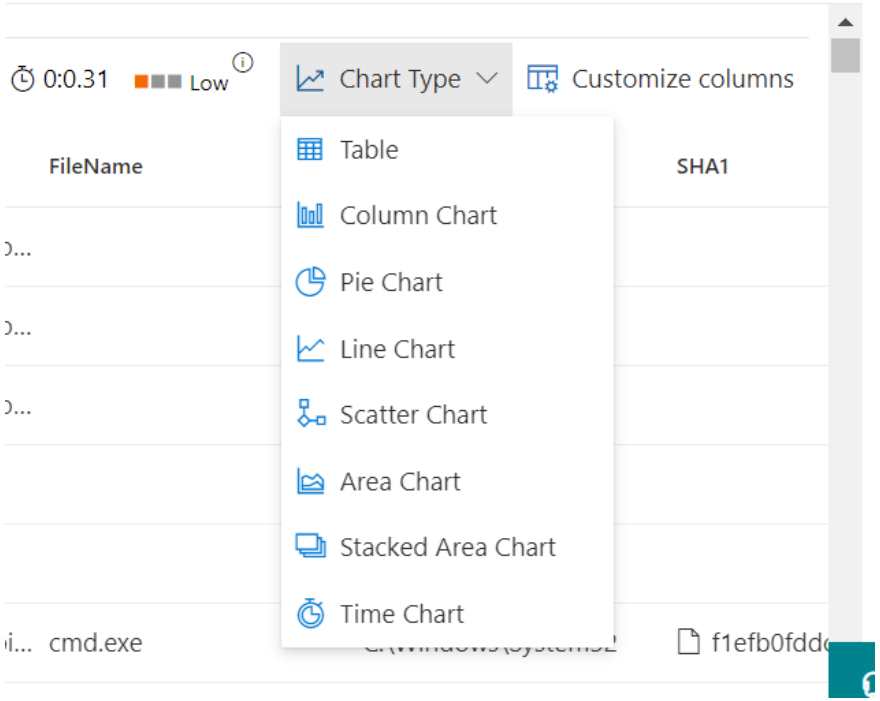
```
let KEV=
externaldata(cveID: string, vendorProject: string, product: string, vulnerabilityName: s
[
h@'https://www.cisa.gov/sites/default/files/csv/known_exploited_vulnerabilities.csv'
]
with(format='csv',ignorefirstrecord=true);
DeviceTvmSoftwareVulnerabilities
| project DeviceName, OSPlatform, cveID=CveId
| join kind=inner KEV on cveID
| summarize ['Vulnerabilities']=make_set(cveID) by DeviceName
| extend ['Count of Known Exploited Vulnerabilities'] = array_length(['Vulnerabilities']
| sort by ['Count of Known Exploited Vulnerabilities']
```

Power of functions

Create Your Own Function()

```
let MakeFolderPathVogonPoetry = (SourceData:(DeviceName:string, FolderPath:string)) {
    };
 DeviceProcessEvents
 | invoke MakeFolderPathVogonPoetry()
```

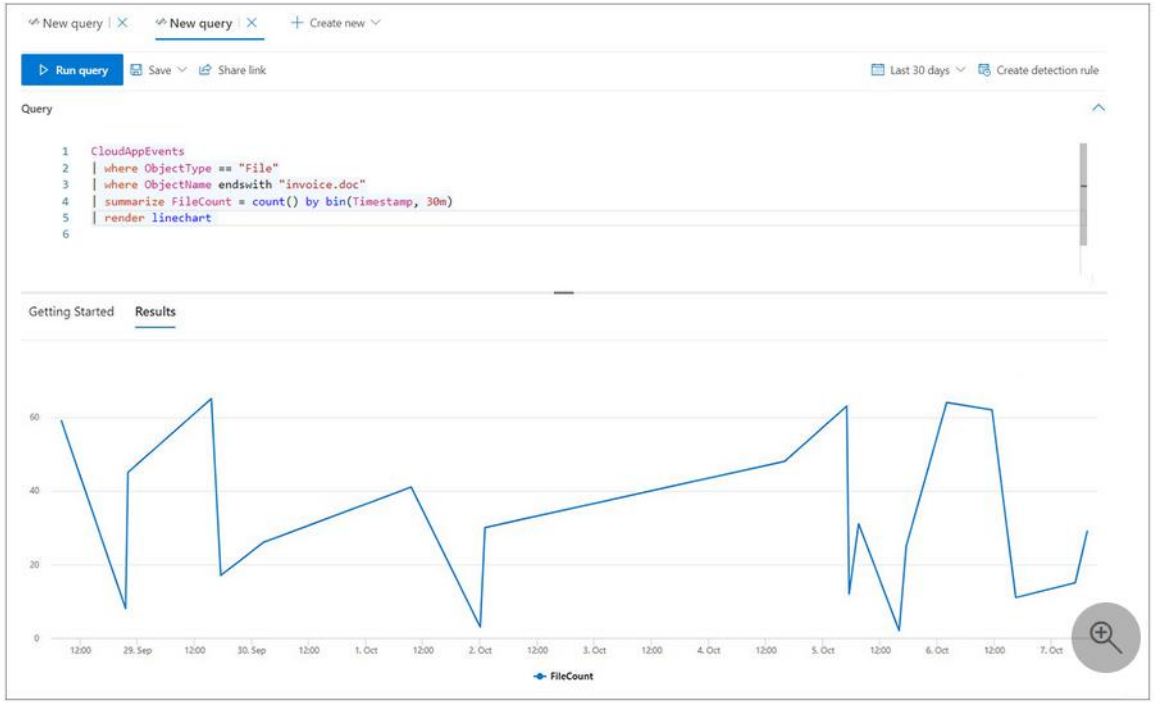
Visualization



Kusto Copy

```
CloudAppEvents
| union DeviceFileEvents
| where FileName == "invoice.doc"
| summarize FileCount = count() by bin(Timestamp, 30m)
```

The line chart below clearly highlights time periods with more activity involving `invoice.doc`:



Simple Use Case

Avoid filtering custom detections using the Timestamp column. The data used for custom detections is pre-filtered based on the detection frequency.

Timestamp and the corresponding ReportId should be part of any returned result for custom detection

```
DeviceProcessEvents
| where FileName == "whoami.exe" and ProcessCommandLine contains "priv"
| project TimeGenerated, DeviceName, InitiatingProcessAccountName, FileName,
InitiatingProcessCommandLine, ProcessCommandLine
```

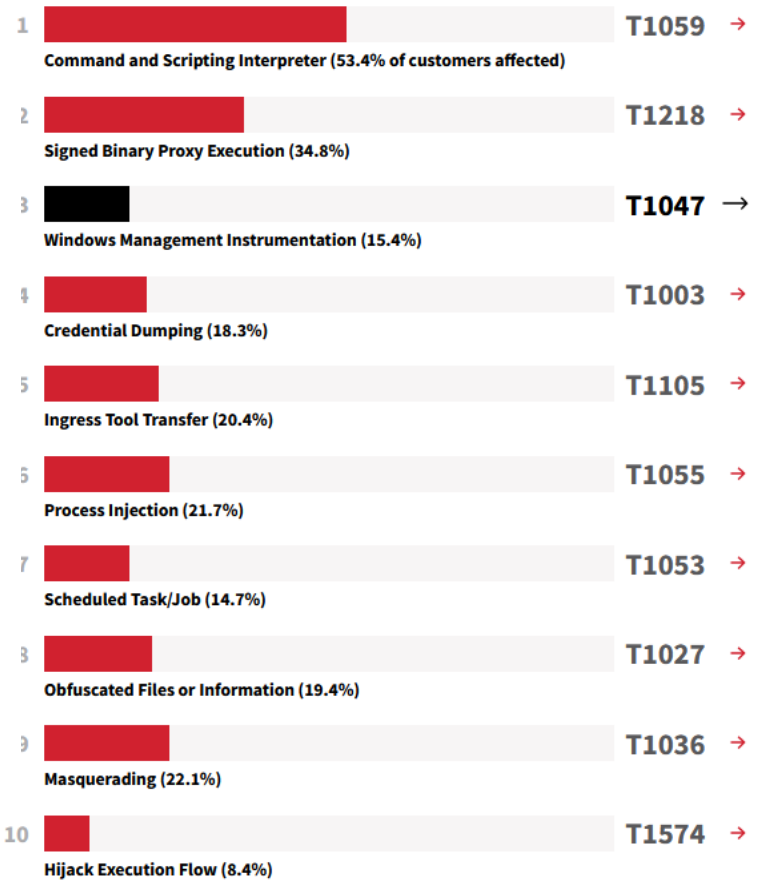
eventcreate /t information /id 7070 /so
HoodEventGenerated /l security /d "Hood test rule"

Detecting common TTPs and Known threats

Top techniques

The purpose of this section is to help you detect malicious activity in its early stages so you don't have to deal with the consequences of a serious security incident.

The following chart represents the most prevalent **MITRE ATT&CK®** techniques observed in confirmed threats across the Red Canary customer base in 2021. To briefly summarize what's **explained in detail in the Methodology section**, we have a library of roughly 3,000 detection analytics that we use to surface potentially malicious and suspicious activity across our customers' environments. These are mapped to corresponding MITRE ATT&CK techniques whenever possible, allowing us to associate the behaviors that comprise a confirmed threat detection with the industry standard for classifying adversary activity.



Hunting Ideas

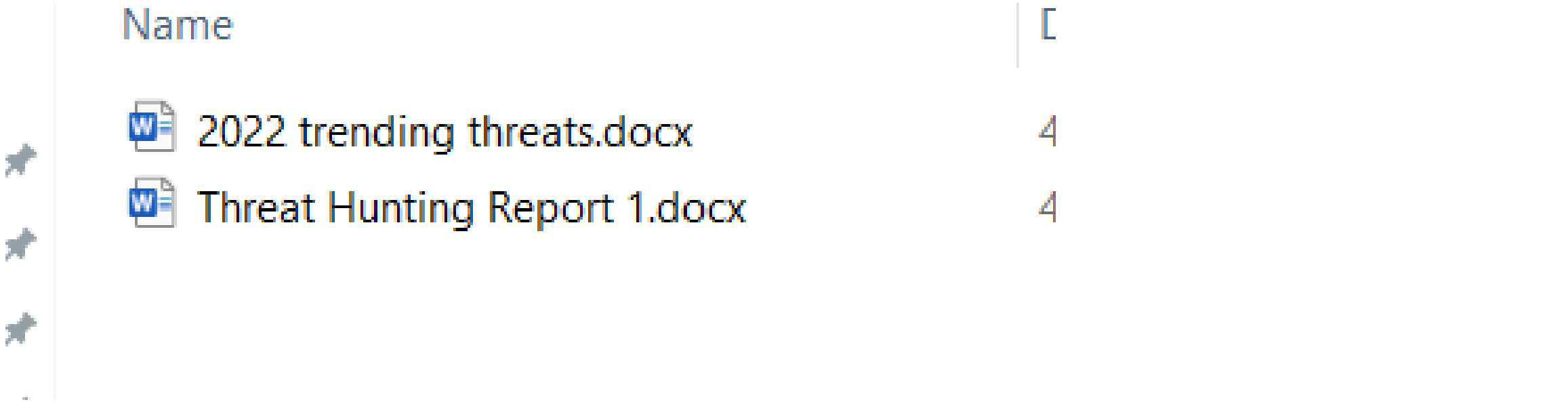
1. Backup deletion/system resotre
2. Spelling mistake
3. Passwords managers and file named password
4. creation of zip files from commandline utility
5. ConsoleHost_history.txt
6. File Age
7. PST file access/browser credential store

```
DeviceFileEvents | where Timestamp > ago(7d) | where FileName
endswith '.pfx' or FileName endswith '.pfn' or FileName endswith '.p12'
```

Trending Threat

Welcome to Red Canary's 2022 Threat Detection Report. Based on in-depth analysis of over 30,000 confirmed threats detected across our customers' environments, this research arms security leaders and their teams with actionable insight into the threats we observe, techniques adversaries most commonly leverage, and trends that help you understand what is changing and why. This is our most expansive report to date, but our intention remains the same: The Threat Detection Report exists to help you understand and detect threats.

Sample Threat Hunting Reports



Harmless Malware

Name	Type	Compressed size	Pa
harmlessmalware.exe	Application	1,493 KB	No
harmlessmsg.exe	Application	762 KB	No