



Free AI for Detection Engineers & Security Analysts, powered by Llama 70b

Key features and capabilities guide v1.0

Questions? Ideas? Join us at Discord Uncoder channel <https://discord.gg/yYd47bA2XV>

Live

Q Detection Rules

Threat Report

Microsoft Sentinel Rule ...

Data Schema: Default

GENERATE Behavior Rule

AI Tools

Contribute

Validate

Intelligence

Save As

1 Critical Vulnerabilities in Kubernetes Ingress-NGINX

2 Download

3 History:

4

5 25/03/2025 --- v1.0 -- Initial publication

6 Summary

7 On March 24, 2025, Wiz Research disclosed a set of critical Remote Code Execution vulnerabilities in the Ingress-NGINX Controller for Kubernetes. The vulnerabilities CVE-2025-1097, CVE-2025-1098, CVE-2025-24514, and CVE-2025-1974 can be exploited to gain full cluster access, resulting in a complete compromise of the environment [1,2].

8

9 The vulnerabilities affect a widely used component in Kubernetes environments responsible for routing external traffic to internal services. Clusters with publicly exposed admission webhooks are at immediate risk.

10

11 Technical Details

12 The vulnerability CVE-2025-1097, with a CVSS score of 8.8, allows an unauthenticated remote attacker to inject configuration into nginx using the auth-tls-match-cn Ingress annotation. This can lead to arbitrary code execution in the context of the ingress-nginx controller, and disclosure of Secrets accessible to the controller. (Note that in the default installation, the controller can access all Secrets cluster-wide.)

13

3 "displayName": "Kubernetes Ingress-NGINX Vulnerability Detection",

4 "description": "Detects potential exploitation of critical vulnerabilities in Kubernetes Ingress-NGINX Controller",

5 "severity": "High",

6 "enabled": true,

7 "query": ""

8 let ingress_nginx_vulnerabilities =

9 (

10 // Detect auth-tls-match-cn annotation usage

11 SecurityEvent

12 | where EventID = 4688

13 | where CommandLine contains "auth-tls-match-cn"

14)

15 or

16 (

17 // Detect mirror-target and mirror-host annotations usage

18 SecurityEvent

19 | where EventID = 4688

20 | where CommandLine contains "mirror-target" or CommandLine contains "mirror-host"

21)

22 or

23 (

24 // Detect auth-url annotation usage

0 Hashes 0 Domains 0 URLs 0 IPs 0 Emails 0 Files 0/10000

How it works?





































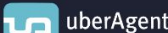







Rule/Query Generation from a Report with AI

Use case development life cycle may be long, sometimes too long. Uncoder AI analyzes the provided threat report and generates a rule/query to detect the described behavior. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure
- Reduces time spent translating threat intelligence into actionable detections
- Enables rapid prototyping of rules
- Allows small teams to cover broader threat landscapes

Rule/Query Generation from a Report with AI

56 languages supported

 ANOMALI	 kafka	 ArcSight	 CORTEX XSIAM	 CORTEX XDR	 AWS Athena	 OpenSearch	 DEVO
 CROWDSTRIKE	 FORTINET	 REGEX	 graylog	 DATADOG	 DNI	 ElastAlert	 Elastic Stack
 HUNTERS	 LOGPOINT	 LogRhythm	 Falco	 CrowdStrike Falcon LogScale	 FIREEYE	 Google Security Operations	 hawksearch
 Radar	 LACEWORK	 LIMA CHARLIE	 LOGIQ	 Microsoft Defender for Endpoint	 Microsoft Sentinel	 NVISIO	 PowerShell
 Qualys	 GREP	 ROOTA	 RSA NETWORKWITNESS	 seconix	 SentinelOne	 snowflake	 splunk
 SQL	 SQLite	 STIX	 StreamAlert	 sumo logic	 Sysmon	 uberAgent	 Carbon Black.
 onum	 Logsign	 SURICATA	 TANIUM	 SOPHOS EDR	 logz.io	 TREND XDR	 exabeam

Live

● Translate functions with Gpt-4o-Mini ⓘ

Detection Rules

Custom Prompt ▼

 GENERATE

1 Write 3 Splunk queries to detect activities associated with APT 29 in an environment where endpoint logging is limited to default Windows audit log

```
3 **Query 1: Detecting suspicious PowerShell activity**
4 ```spl
5 index=windows_security EventID=4688
6 | regex "powershell.exe"
7 | search (CommandLine="*Invoke-Command*" OR CommandLine="*Invoke-Expression*")
8 | stats count as num_events by ComputerName, UserName, CommandLine
9 | where num_events > 5
10 | sort -num_events
11 ```
12 This query looks for PowerShell execution events (Event ID 4688) that contain
   suspicious command lines, such as 'Invoke-Command' or 'Invoke-Expression', which
   are commonly used by APT 29. It then aggregates the results by computer name, user
   name, and command line, filtering out instances with fewer than 5 occurrences.
13
14 **Query 2: Identifying potential credential dumping**
15 ```spl
16 index=windows_security EventID=4663
17 | regex "lsass.exe"
18 | search (ObjectName="*SAM*" OR ObjectName="*LSA*")
19 | stats count as num_events by ComputerName, UserName, ObjectName
20 | where num_events > 2
21 | sort -num_events
```

How it works?



DEBUG CONSOLE (1) ●



Rule/Query Generation with a Custom AI Prompt

The context and requirements when writing detections may be unique. Uncoder AI analyzes the provided custom prompt and generates a rule/query to detect the described behavior taking into account user's instructions. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- All the capabilities of rule generation from threat report
- Custom instructions allow for generating detections tailored to a particular environment or context

Live, all tiers

Q Detection Rules

Elastic Stack Query (EQL) ▼

Select Platform ▼



TRANSLATE ▼

```
1 from logs-endpoint.events.process-* | keep @timestamp, host.os.type, event.type, event.action,
  process.name, process.args, process.command_line, agent.id, host.name | where @timestamp >
  now() - 1 hours | where host.os.type == "linux" and event.type == "start" and event.action
  == "exec" and ( (process.name in ("base64", "base64plain", "base64url", "base64mime",
  "base64pem", "base32", "base16") and process.command_line like "*-d*" ) or (process.name ==
  "openssl" and process.args == "enc" and process.args in ("-d", "-base64", "-a")) or (process
  .name like "python*" and (process.args == "base64" and process.args in ("-d", "-u", "-t"))
  or (process.args == "-c" and process.command_line like "*base64*" and process.command_line
  like "*b64decode*") ) or (process.name like "perl*" and process.command_line like "
  *decode_base64*") or (process.name like "ruby*" and process.args == "-e" and process
  .command_line like "*Base64.decode64*") ) | stats cc = count(), agent_count = count_distinct
  (agent.id), host.name = VALUES(host.name), agent.id = VALUES(agent.id) by process.name,
  process.command_line | where agent_count == 1 and cc < 15 | sort cc asc | limit 100
```

AI-generated Decision Tree

The Elastic Query (EQL) you've provided is designed to detect potential base64 decoding activities in Linux systems. Here's a breakdown of the decision tree embedded within this query:

Initial Filtering

- Time Frame:** The query starts by filtering events from the last hour (`@timestamp > now() - 1 hours`).
- Linux Events:** It then narrows down to only include events where the host operating system is Linux (`host.os.type == "linux"`).
- Event Type and Action:** Further filtering includes only `start` type events with an `exec` action (`event.type == "start" and event.action == "exec"`).

Specific Process Detection

The core of the decision tree involves detecting specific processes that could be involved in base64 decoding, categorized by process name. The conditions are as follows:

- Base64 Utilities:**

- Processes named `base64`, `base64plain`, `base64url`, `base64mime`, `base64pem`, `base32`, or `base16`.
- These processes must have a command line containing `-d*`, suggesting a decode operation.

- OpenSSL:**

- The process name is `openssl`.
- It must be used with the `enc` argument and either `-d` (decode), `-base64`, or `-a` (which in some contexts might imply base64 encoding, but here it's part of decoding conditions).

- Python:**

Rule/Query's Decision Tree summarization with AI

When dealing with complex threat detection logic, clarity is a necessity. Uncoder AI analyzes a query/rule and explains how it works step by step, with all the embeddings, branches and other intricate logic. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Specific concise explanations
- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure

Rule/Query's Decision Tree summarization with AI

48 languages supported in production



In production

Potential Unconstrained Delegation Discovery Via Get-ADComputer - ScriptBlock

```
1 index=* source="WinEventLog:*" AND (ScriptBlockText="*-Properties*TrustedForDelegation*" OR
  ScriptBlockText="*-Properties*TrustedToAuthForDelegation*" OR ScriptBlockText="*-Properties*msDS
  -AllowedToDelegateTo*" OR ScriptBlockText="*-Properties*PrincipalsAllowedToDelegateToAccount*" OR
  ScriptBlockText="*-LDAPFilter*(userAccountControl:1.2.840.113556.1.4.803:=524288)*")
```

Full AI-generated Summary

This Splunk query is designed to search for specific Windows Event Log entries that contain indicators of potential Kerberos delegation abuse or configuration changes. Here's a breakdown of what the query does:

- Index and Source Filtering:** The query starts by specifying `index=*` which means it searches across all available indexes in the Splunk environment. It then narrows down the results to only include logs from sources named "WinEventLog:*", indicating that the focus is on Windows Event Logs.
- ScriptBlockText Conditions:** The main body of the query uses an `AND` operator with a series of `OR` conditions, all related to `ScriptBlockText`. This implies the search is looking for PowerShell script blocks (code snippets) within these event logs that match certain criteria. Specifically, it's searching for any mentions of properties or filters related to Kerberos delegation.
 - TrustedForDelegation:** This condition searches for instances where a user or computer account has been configured as trusted for delegation. In a Kerberos environment, this setting allows the account to impersonate users and access services on their behalf.
 - TrustedToAuthForDelegation:** Similar to the previous condition but specifically related to constrained delegation scenarios where an account is trusted to authenticate for delegation purposes.
 - msDS-AllowedToDelegateTo:** This attribute specifies the services to which a user or computer can present delegated credentials. The query looks for modifications or mentions of this attribute in script blocks, indicating possible configuration changes that could affect delegation paths.
 - PrincipalsAllowedToDelegateToAccount:** This condition targets configurations where specific principals (users, computers, etc.) are explicitly allowed to delegate their credentials to a particular account, which is another aspect of constrained delegation.
 - LDAPFilter with userAccountControl:** The final condition searches for LDAP filters used in script blocks that specifically target accounts with the `userAccountControl` attribute set to `524288`, which corresponds to the "TRUSTED_FOR_DELEGATION" flag. This flag indicates that an account is trusted for delegation, and modifying or searching for this could be part of managing or exploiting Kerberos delegation settings.

In summary, this Splunk query aims to detect potential security-related activities or misconfigurations within Windows environments related to Kerberos delegation. It looks for indicators in PowerShell script blocks captured in the Windows Event Logs that might suggest changes to trusted delegation settings, which could be used by attackers to move laterally within a network or by administrators to manage and audit their environment's delegation configurations.

Modern SOC teams need to process large volumes of detection logic. Much of it is complex, verbose, and time-consuming to understand. Uncoder AI analyzes rule/query and provides you with a detailed yet clear explanation of the detection logic and all the fine points involved. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Clear and structured summary of the detection logic in human language
- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure

48 languages supported in production

 ANOMALI	 kafka	 ArcSight	 CORTEX XSIAM	 CORTEX XDR	 AWS Athena	 OpenSearch	 DEVO
 CROWDSTRIKE	 FORTINET	 RESEX	 graylog	 DATADOG	 DNI	 ElastAlert	 Elastic Stack
 HUNTERS	 LOGPOINT	 LogRhythm	 Falco	 CrowdStrike Falcon LogScale	 FIRE EYE	 Google Security Operations	 hawksearch
 Radar	 LACEWORK	 LIMA CHARLIE	 LOGIQ	 Microsoft Defender for Endpoint	 Microsoft Sentinel	 nviso	 PowerShell
 Qualys	 GREP	 ROOTA	 RSA NETWORKS	 securonix	 SentinelOne	 snowflake	 splunk
 SQL	 SQLite	 STIX	 StreamAlert	 sumo logic	 Sysmon	 uberAgent	 Carbon Black.

Live, all tiers

Q Detection Rules

Microsoft Sentinel Query (Kusto) v

Select Platform v



TRANSLATE



AI Tools v



Save As v



Contribute



Intelligence



```
1 // Some URL are wrapped with a safelink
2 // Let's get the the unwrapped url and clicks
3 AlertInfo
4 | where ServiceSource =~ "Microsoft Defender for Office 365"
5 | join (
6     AlertEvidence
7     | where EntityType == "Url"
8     | project AlertId, RemoteUrl
9 )
10 on AlertId
11 | join (
12     AlertEvidence
13     | where EntityType == "MailMessage"
14     | project AlertId, NetworkMessageId
15 )
16 on AlertId
17 // Get the unique NetworkMessageId for the email containing the Url
18 | distinct RemoteUrl, NetworkMessageId
19 | join EmailEvents on NetworkMessageId
20 // Get the email RecipientEmailAddress and ObjectId from the email
21 | distinct RemoteUrl, NetworkMessageId, RecipientEmailAddress, RecipientObjectId
22 | join kind = inner IdentityInfo on $left.RecipientObjectId == $right.AccountObjectId
23 // get the UserSid of the Recipient
24 | extend OnPremSid = AccountSID
25 | distinct RemoteUrl, NetworkMessageId, RecipientEmailAddress, RecipientObjectId, OnPremSid
26 // Get the Url click event on the recipient device.
27 | join kind = inner
28     (DeviceEvents
29     | where ActionType == "BrowserLaunchedToOpenUrl" | where isnotempty(RemoteUrl)
30     | project UrlDeviceClickTime = Timestamp, UrlClickedByUserSid = RemoteUrl,
31       InitiatingProcessAccountSid, DeviceName, DeviceId, InitiatingProcessFileName
```

AI-generated Query Optimization ⓘ



The provided Microsoft Sentinel Query is quite complex and involves multiple joins. To optimize it for better performance (i.e., faster search speed), I'll provide suggestions to improve its efficiency.

Optimized Query:

```
AlertInfo
| where ServiceSource =~ "Microsoft Defender for Office 365"
| join (
    AlertEvidence
    | where EntityType == "Url"
    | project AlertId, RemoteUrl
) on AlertId
| join (
    AlertEvidence
    | where EntityType == "MailMessage"
    | project AlertId, NetworkMessageId
) on AlertId
| distinct RemoteUrl, NetworkMessageId
| join kind=inner EmailEvents on NetworkMessageId
| project RemoteUrl, NetworkMessageId, RecipientEmailAddress, RecipientObjectId
| join kind=inner IdentityInfo on $left.RecipientObjectId == $right.AccountObjectId
| extend OnPremSid = AccountSID
| distinct RemoteUrl, NetworkMessageId, RecipientEmailAddress, RecipientObjectId, OnPremSid
| join kind=inner (
    DeviceEvents
    | where ActionType == "BrowserLaunchedToOpenUrl" and isnotempty(RemoteUrl)
    | project UrlDeviceClickTime = Timestamp, UrlClickedByUserSid = RemoteUrl,
      InitiatingProcessAccountSid, DeviceName, DeviceId, InitiatingProcessFileName
) on $left.OnPremSid == $right.InitiatingProcessAccountSid and $left.RemoteUrl == $right.UrlClickedByUserSid
| distinct UrlDeviceClickTime, RemoteUrl, NetworkMessageId, RecipientEmailAddress, RecipientObjectId,
  OnPremSid, UrlClickedByUserSid, DeviceName, DeviceId, InitiatingProcessFileName
| sort by UrlDeviceClickTime desc
```


Query performance can become a bottleneck. Uncoder AI analyzes a query and either confirms it's optimal or suggests performance improvements. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Practical and specific suggestions
- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure

48 languages supported in production



In production

AI Tools Save As Contribute Intelligence

Potentially Suspicious WDAC Policy File Creation

```
1 metadata.event_type = "FILE_CREATION" and target.file.full_path =  
  /*\\Windows\\System32\\CodeIntegrity\\.*\\ nocase and ((( not target.process.file.full_path =  
  /*\\Microsoft\\ConfigurationManagement\\.exe$/ nocase and not target.process.file.full_path =  
  /*\\WDAC Wizard\\.exe$/ nocase and not target.process.file.full_path = /*C:\\Program  
  Files\\PowerShell\\7-preview\\pwsh.exe$/ nocase and not target.process.file.full_path = /*C  
  :\\Program Files\\PowerShell\\7\\pwsh.exe$/ nocase and not target.process.file.full_path = /*C  
  :\\Windows\\System32\\dllhost.exe$/ nocase and not target.process.file.full_path = /*C  
  :\\Windows\\System32\\WindowsPowerShell\\v1\\.0\\powershell_ise.exe$/ nocase and not target  
  .process.file.full_path = /*C:\\Windows\\System32\\WindowsPowerShell\\v1\\.0\\powershell.exe$/  
  nocase and not target.process.file.full_path = /*C:\\Windows\\SysWOW64\\dllhost.exe$/ nocase and  
  not target.process.file.full_path = /*C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1\\  
  .0\\powershell_ise.exe$/ nocase and not target.process.file.full_path = /*C  
  :\\Windows\\SysWOW64\\WindowsPowerShell\\v1\\.0\\powershell.exe$/ nocase)) or ((( not target  
  .process.command_line = /*ConvertFrom-CIPolicy -XmlFilePath.*$/ nocase) and ( not target.process  
  .command_line = /*-BinaryFilePath.*$/ nocase)) or not target.process.command_line = /*CiTool  
  --update-policy.*$/ nocase or (( not target.process.command_line = /*Copy-Item -Path.*$/ nocase) and  
  ( not target.process.command_line = /*-Destination.*$/ nocase))))))
```

Short AI-generated Summary


This Google SecOps Query is designed to detect potential malicious activity related to file creation in the Windows System32 folder, specifically within the CodeIntegrity directory. It filters out known legitimate processes and commands that may create files in this location, such as those related to Microsoft Configuration Management, WDAC Wizard, PowerShell, and dllhost. The query aims to identify unauthorized or unexpected file creation events that could indicate a security threat.

Rule/Query Short Summary with AI

When working with third-party content, you often wish you could understand a query/rule at a glance. Uncoder AI transforms detection queries/rules into concise human-readable insights. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- "Executive level" summary of the detection logic
- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure

48 languages supported in production

 ANOMALI	 kafka	 ArcSight	 CORTEX XSIAM	 CORTEX XDR	 AWS Athena	 OpenSearch	 DEVO
 CROWDSTRIKE	 FORTINET	 RESEX	 graylog	 DATADOG	 DNI	 ElastAlert	 Elastic Stack
 HUNTERS	 LOGPOINT	 LogRhythm	 Falco	 CrowdStrike Falcon LogScale	 FIREEYE	 Google Security Operations	 hawksearch
 Radar	 LACEWORK	 LIMA CHARLIE	 LOGIQ	 Microsoft Defender for Endpoint	 Microsoft Sentinel	 nviso	 PowerShell
 Qualys	 GREP	 ROOTA	 RSA NETWORKS	 securonix	 SentinelOne	 snowflake	 splunk
 SQL	 SQLite	 STIX	 StreamAlert	 sumo logic	 Sysmon	 uberAgent	 Carbon Black.

Live

● Translate functions with Gpt-4o-Mini ⓘ

Q Detection Rules

Splunk Query (SPL) ▼

Select Platform ▼



TRANSLATE ▼



```
1 index=* source="WinEventLog:*" EventCode=4625 OR EventCode=4624
2 | bin _time span=5m as minute
3 | rex "Security ID:\\s*\\w*\\s*\\w*\\s*Account Name:\\s*(?<username>.*))\\s*Account Domain:"
4 | stats count(Keywords) as Attempts, count(eval(match(Keywords, "Audit Failure"))) as Failed,
   count(eval(match(Keywords, "Audit Success"))) as Success by minute username
5 | where Failed >= 4
6 | stats dc(username) as Total by minute
7 | where Total > 5
```

Overall Structure: The query is well-structured and follows a logical sequence of commands. It uses pipes (|) to separate each command, which is correct.

Commands and Functions:

- `index=* source="WinEventLog:*"`: Correct usage of index and source specifications.
- `EventCode=4625 OR EventCode=4624`: Correct usage of the `OR` operator for filtering events.
- `bin _time span=5m as minute`: Correct usage of the `bin` command for grouping time intervals.
- `rex "Security ID:\\s*\\w*\\s*\\w*\\s*Account Name:\\s*(?<username>.*))\\s*Account Domain:"`: The regular expression seems complex, but it's correctly formatted and uses a named capture group (`(?<username>.*)`) to extract the username.
- `stats count(Keywords) as Attempts, ...`: Correct usage of the `stats` command for aggregating values.
- `where Failed >= 4` and `where Total > 5`: Correct usage of the `where` command for filtering results.

Field Value Delimiters: The query uses correct field value delimiters, such as double quotes (") around string values and backticks (`) for regex patterns.

Microsoft Sentinel Default Data Schema: The query appears to be using fields that are consistent with the Microsoft Sentinel default data schema, including `EventCode`, `Keywords`, and `_time`.

Performance Considerations:

- The use of `index=*` may lead to performance issues if the index is very large, as it searches all indexes.
- The `rex` command can be computationally expensive, especially with complex patterns.
































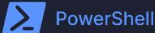
























Rule Syntax and Structure Validation with AI

Everyone makes mistakes, especially in a rapid rule development environment. Uncoder AI analyzes the syntax and structure of a rule/query and flags errors, suggests improvements, or confirms that everything is correct. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Multiple languages supported
- Data doesn't leave SOC Prime's infrastructure
- Reduces time spent manually debugging syntax or structural issues to enable faster iteration and deployment of new detections
- Assists less experienced engineers by offering real-time, contextual feedback and improvement suggestions
- Flags logic flaws (e.g., overly broad conditions, redundant clauses), not just syntax

Rule syntax and structure validation with AI

56 languages supported

Predict ATT&CK Tags in Sigma Rules with ML

```
1 title: llama3.3:70b invoke Keyword
2 id: 63 0508e
3 relate Short Summary ⓘ
4 - id Full Summary ⓘ 922c356f7
5 ty Decision Tree ⓘ
6 status
7 descri Query Optimization ⓘ 16 Base64 encoded powershell 'Invoke-' calls
8 refer Predict ATT&CK tags ⓘ
9 - ht-.../05/09/seo-poisoning-a-gootloader-story/
10 author: pH-T (Nextron Systems), Harjot Singh, @cyb3rjy0t
11 date: 2022-05-20
12 modified: 2023-04-06
13 tags:
14 - attack.t1059.001
15 - attack.t1027
16 logsource:
17 category: process_creation
18 product: windows
19 detection:
20 selection_img:
21 - Image|endswith:
22 - \powershell.exe
23 - \pwsh.exe
24 - OriginalFileName:
25 - PowerShell.EXE
```

Live

Predict ATT&CK Tags in Sigma Rules with ML

MITRE ATT&CK is a widely used framework for detection content, yet mapping rules to it takes time and training. Uncoder AI uses a privately hosted ML model to map a provided Sigma rule to ATT&CK techniques and subtechniques.

- Data doesn't leave SOC Prime's infrastructure
- The model has been trained on over 20,000 Sigma rules, being the largest manually created dataset in existence
- SOC Prime has unique way of tagging Sigma rules as we have've invented this approach in 2018 and advocated it since then
- Reduces manual effort in mapping detections to ATT&CK
- Ensuring that detections are systematically aligned to ATT&CK:
 - Improves visibility into technique coverage and gaps
 - Facilitates better correlation with threat intel, red team findings, and adversary emulation plans
 - Helps in structured reporting

Open Beta for all

Detection Rules

Threat Report

Select Platform

GENERATE
IOC Query

1 Introduction:

2

3 Security researchers at Seqrite Labs have recently uncovered two distinct campaigns carried out by the APT group "Kimsuky," also known as "Black Banshee." This group has been actively targeting South Korea using evolving tactics. In these campaigns, the threat actors delivered two South Korean government-themed documents as lures, specifically targeting government entities within South Korea

4

5 In this blog, we will delve into the technical details of the campaigns uncovered during our analysis. We will examine the various stages of infection, starting with a phishing email containing an LNK (shortcut) file attachment. The LNK file was designed to drop an obfuscated VBA (Visual Basic for Applications) script, After de-obfuscating the script, we found that it was responsible for dropping two additional files: One Pdf file and One ZIP file The ZIP file contained four malicious files: two log files (1.log and 2.log), one VBA script (1.vba), and one PowerShell script (1.ps1). Both campaigns involved the same set of malicious files.

6

7 Infection Chain:

8

9

10 Fig .1 infection chain

11 Initial Findings:

1 Hashes 0 Domains 0 URLs 0 IPs 0 Emails 15 Files 16/10000

Attack Flow

Action - T1566.001 Phishing:
Spearphishing Attachment : Victims receive spear phishing emails with malicious zip files (like .zip or .rar).
Confidence: Certainty

leads_to

Action - Execution of obfuscated VBA script
Download and execution of PowerShell script

leads_to

Action - Creation of registry entries for persistence
Use of "1.log" and "1.vbs" files

leads_to

Action - Deobfuscation/Decoding of files or information

Confidential and Proprietary. Do not distribute without consent. © SOC Prime 2025. All rights reserved.

Attack Flow Generation with AI

Visualization can be a great help in understanding an attack. Uncoder AI analyzes the provided threat report or other description of malicious activities and visualizes it in the form of Attack Flow. For this purpose, Uncoder AI uses Llama 3.3 customized for detection engineering and threat intelligence processing, hosted at SOC Prime SOC 2 Type II private cloud for maximum security, privacy, and IP protection.

- Inspired by the open-source [Attack Flow](#) project to help defenders move from tracking individual adversary behaviors to tracking the sequences of behaviors that adversaries employ to move towards their goals
- Data doesn't leave SOC Prime's infrastructure
- Reduces the time to understand the attack. On average, generation takes about 2 minutes
- Visualized attack flows can directly inform detection rule logic by identifying TTP chains, enabling proactive defense without relying on IOCs
- When linked to existing telemetry or detection rules, it helps prioritize threats that map to known gaps or current alerts
- Machine-readable MMD export for easier integration with detection engineering workflows
- Gives engineers a visual depiction that aids communication with non-technical stakeholders, management, and executives

Live for paid,
final QA for free

Detection Rules

Microsoft Sentinel Query (Kusto)

Splunk Query (SPL)

TRANSLATE

AI Tools Save As Contribute

Sigma Splunk Query (SPL)

Save As

```
1 SAPBTPAuditLog_CL
2 | where Message has "malware"
3 | extend MessageData = parse_json(tostring(Message.data))
4 | extend
5 |   ClusterID = tostring(MessageData.clusterID),
6 |   WorkspaceID = tostring(MessageData.wsID),
7 |   Message = tostring(MessageData.message)
8 | parse Message with * 'user: ' User '.The following issues were detected: ' Malware ',' *
9 | extend
10 |   AccountName = tostring(split(User, '@')[0]),
11 |   UPNSuffix = tostring(split(User, '@')[1])
12 | project
13 |   UpdatedOn,
14 |   ClusterID,
15 |   WorkspaceID,
16 |   Message@,
17 |   User,
18 |   Malware,
19 |   Tenant,
20 |   SpaceId,
21 |   Category,
22 |   CloudApp = "SAP BTP",
23 |   AccountName,
24 |   UPNSuffix
```

```
1 source=WinEventLog:* AND Message="*malware*" | spath input=Message.data output=MessageData | eval
  ClusterID = coalesce(MessageData.clusterID, ""), WorkspaceID = coalesce(MessageData.wsID, ""),
  Message = coalesce(MessageData.message, "") | rex field=Message "user: (?<User>.*?)" | eval AccountName = split(User, "@")[0], UPNSuffix = split
  (User, "@")[1] | table UpdatedOn, ClusterID, WorkspaceID, Message, User, Malware, Tenant, SpaceId,
  Category, CloudApp="SAP BTP", AccountName, UPNSuffix
2
3 *** uuid: aacd3c35-862d-40b0-8788-07718b85a247 ***
4 *** author: not defined in query/rule ***
5 *** licence: DRL 1.1 ***
```

How it works?

DEBUG CONSOLE (1)

[08:49:33] UNMAPPED FIELDS IN SOURCE

Message

Translating content across security platform languages can become a nightmare, especially as part of SIEM migration. Uncoder AI translates across platform-native languages:

- 10 source languages and 21 target languages supported
- Basic query logic translated natively by Uncoder. Advanced function translation generated by third-party AI (OpenAI's GPT-4o-mini model)
- Opt-in use of third-party AI, only advanced functions are sent as part of prompt (we're in the process of transitioning to a locally hosted Llama model to further improve on privacy and speed)
- Any unmapped fields and source parts that are not supported by target are listed for manual review
- Sigma rule generated for each translation to capture the basic logic

AI-Assisted Cross-Platform Translation

21 languages supported as target

ANOMALI



AWS Athena

OpenSearch

CORTEX XSIAM

CORTEX XDR

CROWDSTRIKE

FORTINET

graylog

ElastAlert

Elastic Stack

HUNTERS



Google Security Operations

LogRhythm

Falco

CrowdStrike
Falcon LogScale

Radar



Microsoft
Defender
for Endpoint



Microsoft
Sentinel

SentinelOne

Sigma

splunk>

10 languages
supported as source



AWS Athena

OpenSearch

CROWDSTRIKE

Elastic Stack

CrowdStrike
Falcon LogScale



Google Security Operations

Radar



Microsoft
Defender
for Endpoint



Microsoft
Sentinel

splunk>

Translate from Sigma into 48 Languages

In production

Q Detection Rules
Sigma
↔
Splunk Alert (SPL)

 TRANSLATE

AI Tools
 Save As
 Contribute
 Validate
 Intelligence

Save As

```

1 title: PowerShell Base64 Encoded Invoke Keyword
2 id: 6385697e-9f1b-40bd-8817-f4a91f40508e
3 related:
4   - id: fd6e2919-3936-40c9-99db-0aa922c356f7
5     type: obsolete
6 status: test
7 description: Detects UTF-8 and UTF-16 Base64 encoded powershell 'Invoke-' calls
8 references:
9   - https://thedfirreport.com/2022/05/09/se0-poisoning-a-gootloader-story/
10 author: pH-T (Nextron Systems), Harjot Singh, @cyb3rjy0t
11 date: 2022-05-20
12 modified: 2023-04-06
13 tags:
14   - attack.execution
15   - attack.t1059.001
16   - attack.defense-evasion
17   - attack.t1027
18 logsource:
19   category: process_creation
20   product: windows
21 detection:
22   selection_img:
23     - Image|endswith:
24       - \powershell.exe
25       - \pwsh.exe
26     - OriginalFileName:
27       - PowerShell.EXE
28       - pwsh.dll
29   selection_cli_enc:
30     CommandLine|contains: '-e'
31   selection_cli_invoke:
```

```

1 [PowerShell Base64 Encoded Invoke Keyword]
2 alert.severity = 4
3 description = Detects UTF-8 and UTF-16 Base64 encoded powershell 'Invoke-' calls (Rule ID:
4   6385697e-9f1b-40bd-8817-f4a91f40508e) Reference: https://tdm.socprime.com/tdm/info/
5 cron_schedule = 0 * * * *
6 disabled = 1
7 is_scheduled = 1
8 is_visible = 1
9 dispatch.earliest_time = -60m@m
10 dispatch.latest_time = now
11 search = index=* source=WinEventLog:Microsoft-Windows-Sysmon/Operational AND
12   (((NewProcessName="*\powershell.exe" OR NewProcessName="*\pwsh.exe") OR
13   (OriginalFileName="PowerShell.EXE" OR OriginalFileName="pwsh.dll")) AND CommandLine="-
14   -e*" AND (CommandLine="*$QBvAHYAbwBrAGUALQ*" OR CommandLine="*kAbgB2AG8AwBlAC0A*" OR
15   CommandLine="*JAG4AdgBVAgSAZQAAtA*" OR CommandLine="*SW52b2tll*" OR CommandLine="
16   *ludm9rZS*" OR CommandLine="*JbnZva2Ut*))
17 alert.suppress = 0
18 alert.track = 1
19 actions = risk,notable
20 action.risk = 1
21 action.risk.param._risk_object_type = user
22 action.risk.param._risk_score = 75
23 action.correlationsearch = 0
24 action.correlationsearch.enabled = 1
25 action.notable.param.rule_title = PowerShell Base64 Encoded Invoke Keyword
26 action.notable.param.rule_description = Detects UTF-8 and UTF-16 Base64 encoded
27   'Invoke-' calls (Rule ID: 6385697e-9f1b-40bd-8817-f4a91f40508e)
28 action.correlationsearch.label = PowerShell Base64 Encoded Invoke Keyword
29 rule.base.base.base.base.base.base.base.base.base.base.base.base.base.base.base.base
```

Splunk

[How it works?](#)

Translate from Sigma into 48 Languages

Don't get into a vendor lock-in with your security platform. Uncoder AI natively translates Sigma rules into multiple SIEM, EDR, XDR, and Data Lake languages.


- Detection portability & scalability across heterogeneous environments
- #1 translation engine for Sigma rules – by users, by languages, by features
- Security vendor agnosticism with Sigma as a single source of truth
- Removing translation overhead
- Leveraging large Sigma community for open-source detections
- Full use case life cycle support when combined with Threat Detection Marketplace

Translate from Sigma into 48 Languages

48 languages supported in production

 ANOMALI	 kafka	 ArcSight	 CORTEX XSIAM	 CORTEX XDR	 AWS Athena	 OpenSearch	 DEVO
 CROWDSTRIKE	 FORTINET	 RESEX	 graylog	 DATADOG	 DNI	 ElastAlert	 Elastic Stack
 HUNTERS	 LOGPOINT	 LogRhythm	 Falco	 CrowdStrike Falcon LogScale	 FIREEYE	 Google Security Operations	 hawksearch
 Radar	 LACEWORK	 LIMA CHARLIE	 LOGIQ	 Microsoft Defender for Endpoint	 Microsoft Sentinel	 nviso	 PowerShell
 Qualys	 GREP	 ROOTA	 RSA NETWORKS	 securonix	 SentinelOne	 snowflake	 splunk
 SQL	 SQLite	 STIX	 StreamAlert	 sumo logic	 Sysmon	 uberAgent	 Carbon Black.

In production

From Splunk Alert (SPL) 

SUPERCHARGE



```
1 [PowerShell Base64 Encoded Invoke Keyword]
2 alert.severity = 4
3 description = Detects UTF-8 and UTF-16 Base64 encoded powershell 'Invoke-' calls (Rule ID: 6385697e-9f1b-40bd-8817-f4a91f40508e) Reference: https://tdm.socprime.com/tdm/info/
4 cron_schedule = 0 * * * *
5 disabled = 1
6 is_scheduled = 1
7 is_visible = 1
8 dispatch.earliest_time = -60m@m
9 dispatch.latest_time = now
10 search = index=* source="WinEventLog:Microsoft-Windows-Sysmon/Operational" AND (((NewProcessName="*\\powershell.exe" OR NewProcessName="*\\pwsh.exe") OR (OriginalFileName="PowerShell.EXE" OR OriginalFileName="pwsh.dll")) AND
    Command line="* -e*" AND (Command line="*SQRuAHVhwrRdGll!l0*" OR Command line="*k4hnR2dCRAswR1dCRd*" OR Command line="*7dC4AdnRvAEcA70dA+*" OR Command line="*SWC3h2!1+*" OR Command line="*ludm0r7S+*" OR Command line="*7bn7uz2!t+
```

To  Roota

Enhance With More Queries

Save As 

```
20 logsource: {}
21 timeline: ''
22 - false-positives: |-
23   Possible False-Positives or Benign Activities for PowerShell Base64 Encoded Invoke Keyword
24   1. Legitimate administrative scripts that utilize Base64 encoding for obfuscation.
25   2. Third-party applications that use PowerShell for automation and may encode commands in Base64.
26   3. Security tools that leverage PowerShell for legitimate purposes, such as endpoint protection or monitoring.
27   4. User-initiated PowerShell scripts that are encoded for ease of transfer or storage.
28   5. Scheduled tasks or cron jobs that execute Base64 encoded PowerShell commands for routine maintenance.
29
30   Recommendations to Avoid False-Positives or Benign Activities
31   1. Implement a whitelist of known legitimate scripts and applications that use Base64 encoding.
32   2. Monitor the context in which the Base64 encoded commands are executed, including user identity and execution time.
33   3. Analyze the content of the decoded Base64 strings to determine if they align with known benign activities.
34   4. Correlate PowerShell execution events with other logs (e.g., user activity, file access) to establish a clearer picture of intent.
35   5. Educate users on the risks of using Base64 encoding in scripts and encourage the use of clear, readable code.
36 - triage_recommendations: |-
37   Possible Actions for Validating and Investigating Malicious Activity
38
39   1. Review logs for PowerShell execution events to identify any suspicious 'Invoke-' calls.
40   2. Analyze the Base64 encoded strings to determine their decoded content and intent.
41   3. Check for unusual patterns or anomalies in the PowerShell command execution history.
42   4. Correlate the identified PowerShell activity with known threat intelligence to assess potential risks.
43   5. Investigate the source of the PowerShell execution to determine if it originated from a legitimate user or process.
```


Turn a platform-specific rule or query into a Roota rule and enrich it with metadata using SOC Prime's proprietary algorithms and AI.

- AI input on possible false positives and triage recommendations (only metadata is used for prompting)
- Adding possible log sources if they were not specified in the original content. Additionally, the audit section is filled that specifies what logging service should be enabled to have the logs required and how to enable it
- Prediction of relevant MITRE ATT&CK techniques and sub-techniques with a machine learning model

Search Detection Rules

Sigma

↑

📄

🌟 AI Tools

⚙️

💾 Save As

⚙️ Contribute

🔍 Validate

⚙️ Intelligence

🔄

📄

🗑️

```
1 title: Rorschach Ransomware Execution Activity
2 id: 0e9e6c63-1350-48c4-9fa1-7ccb235edc68
3 status: test
4 description: Detects Rorschach ransomware execution activity
5 references:
6   - https://research.checkpoint.com/2023/rorschach-a-new-sophisticated-and-fast-ransomware/
7 author: X__Junior (Nextron Systems)
8 date: 2023-04-04
9 modified: 2023-04-22
10 tags:
11   - attack.execution
12   - attack.t1059.003
13   - command and scr
14   - attack.attack.t1059 #Command and Scripting Interpreter: ATT&CK
15   - detectiattack.t1059.001 #Command and Scripting Interpreter: PowerShell: ATT&CK
16 logsource: attack.t1059.002 #Command and Scripting Interpreter: AppleScript: ATT&CK
17 category: attack.t1059.003 #Comma... and Scripting Interpreter: Windows Command She... ATT&CK
18 product: attack.t1059.004 #Command and Scripting Interpreter: Unix Shell: ATT&CK
19 detection: attack.t1059.005 #Command and Scripting Interpreter: Visual Basic: ATT&CK
20 selection attack.t1059.006 #Command and Scripting Interpreter: Python: ATT&CK
21   Image|attack.t1059.007 #Command and Scripting Interpreter: JavaScript: ATT&CK
22     - \bcdedit.exe
23     - \net.exe
24     - \net1.exe
25     - \netsh.exe
26     - \wevtutil.exe
27     - \vssadmin.exe
28     CommandLine|contains: '11111111'
29 condition: selection
30 falsepositives:
31   - Unlikely
```

In production

Uncoder AI is an IDE for detection engineering. Any IDE starts with a convenient code editor.

- Language-specific syntax highlighting
- Automatic language detection
- Sigma and Rooter templates
- Upload from file
- Code autocomplete including MITRE ATT&CK and log sources from all Sigma rules in Threat Detection Marketplace

In production

Detection Rules

✕

Platform Repositories

Select All

SOC Prime

Roota

Threat Bounty

SigmaHQ/sigma

Azure/Azure-Sentinel

Community

Active Threat IOCs

Blitzkrieg/sigma-rules

The-DFIR-Report/Sigma-Rules

The-DFIR-Report/Sigma-Rules

Platform Repos

My Repos

HOT OSINT Indicators

Possible RDP Resource Redirection Patterns (via file_event)

Author: SOC Prime Team

Possible Rogue RDP via Outlook Attachment (via file_event)

Author: SOC Prime Team

Possible Msedge Dynamic Library Side-Loading Attempt (via image_load)

Author: SOC Prime Team

Possible Screen Capture (via powershell)

Author: SOC Prime Team

Possible Microsoft Console File Created In Unusual Folder (via file_event)

Author: SOC Prime Team

Possible EDR Disablement Attempt Using WDAC (via file_event)

Author: SOC Prime Team

1-10 of 82 10 ▾

< 1 2 3 4 5 ... 9 >

Search Threat Detection Marketplace

Search Threat Detection Marketplace content right from Uncoder AI. Supports both platform and custom repositories. Threat Detection Marketplace is the world's largest repository of public, private and Threat Bounty-made detection rules and queries, indexing over 15 open-source repos, and sporting over 500,000 detection rules in 48 languages. Every rule that exists is in Threat Detection Marketplace, as long as its author's license permitted that.

- Easily find detections for task at hand
- Conveniently open custom rules
- Discover ideas and inspiration for detection content