I Am the Snake Now: Analysis of Snake Malware

Threat detection engineers must often write detections against novel pieces of malware, exploits, and techniques. At times, proof of concept tools or attack simulation frameworks exist for demonstrating novel malware and generating side effects (such as log entries) on a targeted system. Other times, particularly in the case of advanced malware or malware which leverages Living Off the Land techniques, those simulations don't exist. Instead, we are often given diagrams, flowcharts, or English language descriptions of malware and a number of Indicators of Compromise (IOCs) suitable for manual inspection (either of log data or of a compromised host). This is usually enough information to author novel Atomic Simulations, which mimic the behavior of said malware and generate log events useful for detecting that malware.

Unfortunately, this process can be time-consuming and error prone. For example, not all threat detection engineers are also experienced PowerShell developers, Python developers, or Windows Registry Experts. Similarly, the amount of time to build Atomics for a large number of Emerging Threats can easily surpass the staffing resources for a typical SOC.

The Splunk Threat Research Team (STRT) used the assistance of ChatGPT, a cutting edge and conversational Large Language Model (LLM), to help build Atomic Simulations based on the Plain English descriptions in the Cybersecurity and Infrastructure Security (CISA) Writeup on the novel Snake Malware. As you will see, using the tool to augment, not replace, an experienced threat detection engineer can save countless hours of time and produce high-quality Atomic Simulations.

A Note on ChatGPT and other Large Language Models: The accuracy, validity, quality, and safety of the outputs from these models is currently the subject of much debate. They may produce code or descriptions that are invalid (syntactically incorrect), are simply wrong (functionally incorrect), have poor accuracy, or even introduce vulnerabilities into your environment. As such, any output from any LLM should be treated as untrusted and reviewed before running it. While these risks exist today, acknowledging them is the first step towards using them safely in your workflow. Overall, we believe that the quality of these tools is already good enough to augment, but not replace, an experienced threat detection engineer.

This document walks through the process the Splunk Threat Research Team used to develop Atomic Simulations with ChatGPT based on the indicators provided in the report and simulate the activity. It also provides an overview of new security content developed to identify adversaries.

Snake Malware

The Snake implant is a highly advanced cyber espionage tool, developed and employed by Russia's Federal Security Service's (FSB) Center 16 for persistent intelligence gathering on important targets. They established a clandestine global peer-to-peer network of Snake-infected computers to carry out operations. This network includes relay nodes that facilitate encrypted and fragmented traffic between Snake implants and FSB's targets, designed to evade detection and data collection.

Snake infrastructure has been identified in over 50 countries across all continents, including the US and Russia. While its infrastructure spans all sectors, Snake's targeting is tactical and intentional. FSB has leveraged Snake globally to extract sensitive intelligence from key targets like government networks, research establishments, and journalists. For instance, they used Snake to access and leak confidential diplomatic documents from a victim in a NATO country. Within the US, FSB has targeted sectors such as education, small businesses, media, government facilities, financial services, critical manufacturing, and communications.

For more information on FSB and Russian state-sponsored cyber activity, please see the joint advisory Russian State-Sponsored and Criminal Cyber Threats to Critical Infrastructure and CISA's Russia Cyber Threat Overview and Advisories webpage.

Leveraging IOCs to Build Atomic Simulations and Content

Within the report there are 4 main Host-Based Detections described in plain English, however no malicious samples were provided. The Splunk Threat Research Team looked at each as if they were software requirements and prompted ChatGPT to build software which met these requirements. Below we break down each detection one by one and provide the prompt used to generate the content. Note that most of our interactions with ChatGPT are conversational - we do not typically accept ChatGPT's first response, but work with it to refine the result until we have something that more closely aligns with the requirement.

Finally, we tested these Atomic Simulations using Splunk Attack Range, an easy-to-use and open source tool that can automatically configure and deploy and configure a full Splunk Environment, including endpoints, in minutes. Attack Range is the perfect tool for simulating and understanding attacks since it forwards log data to a Splunk instance out-of-the-box.

Registry Blog

Indicators of Compromise:

The registry blob is generally found at the location listed below. If it is not present at its typical location, the registry blob can be found by searching the full registry for a value of at least 0x1000 bytes in size and entropy of at least 7.9.

- **Typical Name:** Unknown (RegBlob)
- **Typical Path:** HKLM\SOFTWARE\Classes\.wav\OpenWithProgIds
- **Characteristics:** High Entropy

First, the STRT wanted a way to identify the contents of the requirements in the registry.

Prompt
A recent malware campaign was identified. it says "The registry blob is generally found at the location listed below. In case it is not present at its typical location, the registry blob can be found by searching the full registry for a value of at least 0x1000 bytes in size and entropy of at least 7.9." How can I do that in powershell?
Initial response

You can search the Windows registry for a value meeting the specified criteria using PowerShell. To do this, you can use the following script:
 	
``powershell
# Set search parameters 	
$minimumSize = 0x1000
$minimumEntropy = 7.9
 	
# Function to calculate entropy of a byte array 	
function Get-Entropy($bytes) {
   $byteFrequency = @{}
   $byteCount = $bytes.Count

```
    foreach ($byte in $bytes) {
        if ($byteFrequency.ContainsKey($byte)) {
            $byteFrequency[$byte]++
        } else {
            $byteFrequency[$byte] = 1
        }
    }

    $entropy = 0
    foreach ($freq in $byteFrequency.Values) {
        $probability = $freq / $byteCount
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}

# Function to search registry for the specified criteria
function Search-Registry {
    param(
        [Parameter(Mandatory=$true)] [ValidateRange(0, [int]::MaxValue)] $minSize,
        [Parameter(Mandatory=$true)] [double] $minEntropy
    )

    $hives = "HKLM:", "HKCU:", "HKCR:", "HKU:", "HKCC:"

    foreach ($hive in $hives) {
        Get-ChildItem -Path $hive -Recurse -ErrorAction SilentlyContinue |
        Get-ItemProperty -ErrorAction SilentlyContinue |
        ForEach-Object {
            $_.PSObject.Properties |
            Where-Object { $_.Value -is [byte[]] -and $_.Value.Count -ge $minSize -and (Get-Entropy $_.Value) -
ge $minEntropy } |
            Select-Object @{Name="Path"; Expression={$_.PSPath}}, @{Name="Name";
Expression={$_.Name}}, @{Name="Value"; Expression={$_.Value}}
        }
    }
}

# Run the search
```

Search-Registry -minSize $minimumSize -minEntropy $minimumEntropy
```

Copy and paste the above script into a PowerShell script file (e.g., "SearchRegistry.ps1") and run it in PowerShell. The script will search the registry for a value with a size of at least 0x1000 bytes and an entropy of at least 7.9, and then display the results.

Please note that searching the entire registry may take a while, and running this script might require administrator privileges. Make sure you understand the implications and potential risks of searching and modifying the registry before proceeding.

We then provided more context and asked another question:

**Prompt**

this was the full datapoint:

Registry Blob

The registry blob is generally found at the location listed below. In case it is not present at its typical

location, the registry blob can be found by searching the full registry for a value of at least 0x1000

bytes in size and entropy of at least 7.9.

Typical Name: Unknown (RegBlob)

Typical Path: HKLM\SOFTWARE\Classes\.wav\OpenWithProgIds

Characteristics: High Entropy

Can you modify the script to check for the typical path, if not there have it scan the full registry. In addition, add some sort of write-host or status update stdout as it is progressing through the registry.

**Updated script response**

I have modified the script to first check the typical path, and if not found, it will then scan the full registry. I have also added status updates as it progresses through the registry.

```powershell
# Set search parameters
$minimumSize = 0x1000
$minimumEntropy = 7.9
$typicalName = "Unknown (RegBlob)"
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"

# Function to calculate entropy of a byte array
function Get-Entropy($bytes) {
```
```

```
    $byteFrequency = @{}
    $byteCount = $bytes.Count

    foreach ($byte in $bytes) {
        if ($byteFrequency.ContainsKey($byte)) {
            $byteFrequency[$byte]++
        } else {
            $byteFrequency[$byte] = 1
        }
    }

    $entropy = 0
    foreach ($freq in $byteFrequency.Values) {
        $probability = $freq / $byteCount
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}

# Function to search registry for the specified criteria
function Search-Registry {
    param(
        [Parameter(Mandatory=$true)] [ValidateRange(0, [int]::MaxValue)] $minSize,
        [Parameter(Mandatory=$true)] [double] $minEntropy
    )

    $hives = "HKLM:", "HKCU:", "HKCR:", "HKU:", "HKCC:"

    foreach ($hive in $hives) {
        Write-Host "Searching in hive: $hive"
        Get-ChildItem -Path $hive -Recurse -ErrorAction SilentlyContinue |
        Get-ItemProperty -ErrorAction SilentlyContinue |
        ForEach-Object {
            $_.PSObject.Properties |
            Where-Object { $_.Value -is [byte[]] -and $_.Value.Count -ge $minSize -and (Get-Entropy $_.Value) -
ge $minEntropy } |
            Select-Object @{Name="Path"; Expression={$_.PSPath}}, @{Name="Name";
Expression={$_.Name}}, @{Name="Value"; Expression={$_.Value}}
        }
    }
```

}

Check the typical path first

Write-Host "Checking typical path: $typicalPath"

$typicalResult = Get-ItemProperty -Path $typicalPath -ErrorAction SilentlyContinue |

Where-Object { $_.PSObject.Properties.Name -eq $typicalName }

if ($typicalResult) {

Write-Host "Found at typical path!"

$typicalResult

} else {

Write-Host "Not found at typical path, scanning the full registry..."

Search-Registry -minSize $minimumSize -minEntropy $minimumEntropy

}

```

Copy and paste the above script into a PowerShell script file (e.g., "SearchRegistry.ps1") and run it in PowerShell. The script will first check the typical path, and if not found, it will search the full registry. Status updates will be displayed as it progresses through the registry.

As before, running this script might require administrator privileges. Make sure you understand the implications and potential risks of searching and modifying the registry before proceeding.

At this point, we can now scan the registry looking for the exact requirement of " 0x1000 bytes in size and entropy of at least 7.9." and the typical path.

However, in our environment nothing matched. The STRT then requested ChatGPT to produce a one-liner that matches the requirements.

**Response**

Here's a one-liner that creates an example registry key with a value that meets the criteria (0x1000 bytes in size and entropy of at least 7.9). This example key can be used for testing purposes.

```powershell
New-Item -Path "HKCU:\Software\TestRegBlob" -Force; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); Set-ItemProperty -Path "HKCU:\Software\TestRegBlob" -Name "Example (RegBlob)" -Value $randomBytes
```

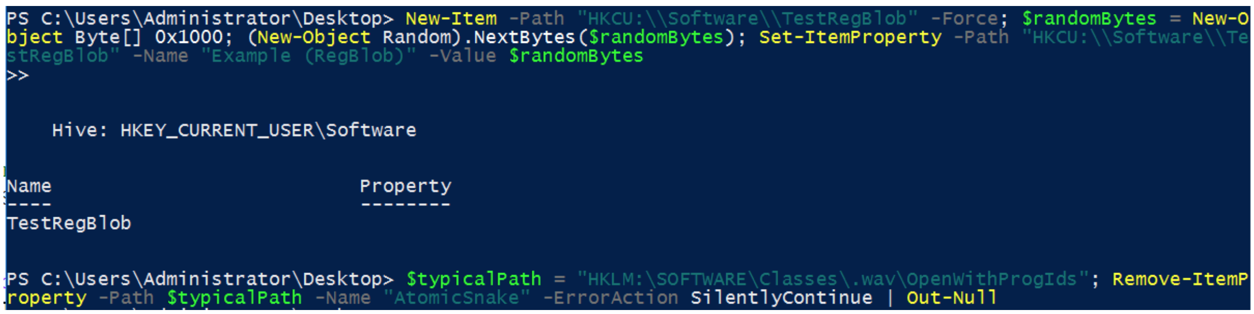
This one-liner creates a new registry key at "HKEY_CURRENT_USER\Software\TestRegBlob" and sets a value named "Example (RegBlob)" with 0x1000 random bytes. You can use this to test the modified script provided in the previous answer.
```

Please note that creating registry keys can have unintended side effects. Always be cautious when modifying the registry and make sure you understand the implications. When you are done testing, you can remove the test key using the following one-liner:

```powershell
Remove-Item -Path "HKCU:\Software\TestRegBlob" -Recurse -Force
```

This one-liner will delete the "HKEY_CURRENT_USER\Software\TestRegBlob" registry key along with its values.

The report mentions that if the blob is not in the typical path to then scan the entire registry. This allows for creating a registry blob that will match the requirements and allow for our scanning script to identify it in a non-typical path.



After a few small changes with ChatGPT, the final response and script looks like this:

Final Response

```
# Set search parameters
$minimumSize = 0x1000
$minimumEntropy = 7.9
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"

# Function to calculate entropy of a byte array
function Get-Entropy($bytes) {
    $byteFrequency = @{}
    $byteCount = $bytes.Count

    foreach ($byte in $bytes) {
        if ($byteFrequency.ContainsKey($byte)) {
            $byteFrequency[$byte]++
```

```
        } else {
            $byteFrequency[$byte] = 1
        }
    }

    $entropy = 0
    foreach ($freq in $byteFrequency.Values) {
        $probability = $freq / $byteCount
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}

# Function to search registry for the specified criteria
function Search-Registry {
    param(
        [Parameter(Mandatory=$true)] [ValidateRange(0, [int]::MaxValue)] $minSize,
        [Parameter(Mandatory=$true)] [double] $minEntropy
    )

    $hives = "HKLM:", "HKCU:", "HKCR:", "HKU:", "HKCC:"

    foreach ($hive in $hives) {
        Write-Host "Searching in hive: $hive"
        Get-ChildItem -Path $hive -Recurse -ErrorAction SilentlyContinue |
        Get-ItemProperty -ErrorAction SilentlyContinue |
        ForEach-Object {
            $properties = $_.PSObject.Properties |
                        Where-Object { $_.Value -is [byte[]] -and $_.Value.Count -ge $minSize -and (Get-Entropy
$_.Value) -ge $minEntropy }
            if ($properties) {
                Write-Host "Found matching registry value:"
                $properties | Select-Object @{Name="Path"; Expression={$_.PSPath}}, @{Name="Name";
Expression={$_.Name}}, @{Name="Value"; Expression={$_.Value}}
            }
        }
    }
}

# Measure the time taken for the scan
```

```
$sw = [System.Diagnostics.Stopwatch]::StartNew()


# Check the typical path first
Write-Host "Checking typical path: $typicalPath"
$typicalResult = Get-ItemProperty -Path $typicalPath -ErrorAction SilentlyContinue |
            Where-Object { $_.PSObject.Properties.Value -is [byte[]] -and
$_.PSObject.Properties.Value.Count -ge $minimumSize -and (Get-Entropy $_.PSObject.Properties.Value) -
ge $minimumEntropy }


if ($typicalResult) {
  Write-Host "Found at typical path!"
  $typicalResult
} else {
  Write-Host "Not found at typical path, scanning the full registry..."
  Search-Registry -minSize $minimumSize -minEntropy $minimumEntropy
}


$sw.Stop()
$elapsedTime = $sw.Elapsed
Write-Host "Time taken for the scan: $($elapsedTime.ToString())"
```

In our lab environment, this is what the scan looks like if it is in the typical path:

```
PS C:\Users\Administrator\Desktop> .\snake_registryblob.ps1
Checking typical path: HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds
Found at typical path!
Time taken for the scan: 00:00:00.0193715
```

When the registry blob is not in the typical path, it will then scan the registry looking for a match:

```
PS C:\Users\Administrator\Desktop> .\snake_registryblob.ps1
Checking typical path: HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds
Not found at typical path, scanning the full registry...
Searching in hive: HKLM:
Get-ItemProperty : Specified cast is not valid.
At C:\Users\Administrator\Desktop\snake_registryblob.ps1:40 char:9
+         Get-ItemProperty -ErrorAction SilentlyContinue |
+         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Get-ItemProperty], InvalidCastException
    + FullyQualifiedErrorId : System.InvalidCastException,Microsoft.PowerShell.Commands.GetItemPropertyComm
   and

Searching in hive: HKCU:
Found matching registry value:

Searching in hive: HKCR:
Searching in hive: HKU:
Searching in hive: HKCC:
Time taken for the scan: 00:04:43.8215230
Path                                                                      Name             Value
----                                                                      ----             -----
Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\SOFTWARE\TestRegBlob Example (RegBlob) {49, 11, 220...
```

The scan typically takes about 4-5 minutes to complete. Once it finds the blob, it outputs as shown above.

For Atomic Red Team, T1112 test number 56 is available to simulate this behavior. ChatGPT assisted with crafting the one-liner.

Prompt
Can you give me a one liner like this Here's a one-liner that creates an example registry key with a value that meets the criteria (0x1000 bytes in size and entropy of at least 7.9). This example key can be used for testing purposes. that places the registry blob in this path: $minimumSize = 0x1000 $minimumEntropy = 7.9 $typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"
Response
Here's a one-liner that creates an example registry key with a value that meets the criteria (0x1000 bytes in size and entropy of at least 7.9) in the specified typical path for testing purposes: ``` $typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); New-ItemProperty -Path $typicalPath -Name "ExampleValue" -Value $randomBytes -PropertyType Binary -Force	Out-Null ``` This script creates a new binary value named "ExampleValue" under the "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds" registry key, and sets its data to a randomly generated byte array of 0x1000 bytes in size. Note that this might not have a perfect entropy of 7.9 or higher, but it should be close for testing purposes.

For an Atomic test, we also want this to have a cleanup:

Prompt
And a delete
Response
Here's a one-liner that deletes the "ExampleValue" we created earlier in the specified typical path: ``` $typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; Remove-ItemProperty -Path

```
$typicalPath -Name "ExampleValue" -ErrorAction SilentlyContinue | Out-Null
```

This script removes the binary value named "ExampleValue" under the "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds" registry key. The `-ErrorAction SilentlyContinue` flag ensures that no error messages are displayed if the value is not found.

The final result in the Atomic Red Team page [T1112](T1112):

Atomic Test #56 - Snake Malware Registry Blob

The following Atomic Test creates a registry blob in HKLM:\SOFTWARE\Classes.wav\OpenWithProgIds, which is related to Snake Malware. Per the report, upon execution, Snake's WerFault.exe will attempt to decrypt an encrypted blob within the Windows registry that is typically found at HKLM:\SOFTWARE\Classes.wav\OpenWithProgIds. The encrypted data includes the AES key, IV, and path that is used to find and decrypt the file containing Snake's kernel driver and kernel driver loader.

Supported Platforms: Windows

auto_generated_guid: 8318ad20-0488-4a64-98f4-72525a012f6b

Attack Commands: Run with `powershell`! Elevation Required (e.g. root or admin)

```
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; $randomBytes = New-Object Byte[] 0x1000; (New-Object R
```

Cleanup Commands:

```
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; Remove-ItemProperty -Path $typicalPath -Name "AtomicSn
```

With Invoke-AtomicRedTeam, we ran this simulation:

Invoke-AtomicTest T1112 -TestNumbers 56 -showdetailsbrief

Invoke-AtomicTest T1112 -TestNumbers 56

```
PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1112 -ShowDetails -TestNumbers 56
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: Modify Registry T1112
Atomic Test Name: Snake Malware Registry Blob
Atomic Test Number: 56
Atomic Test GUID: 8318ad20-0488-4a64-98f4-72525a012f6b
Description: The following Atomic Test creates a registry blob in HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds
, which is related to Snake Malware. Per the report, upon execution, Snake's WerFault.exe will attempt to dec
rypt an encrypted blob within the Windows registry that is typically found at HKLM:\SOFTWARE\Classes\.wav\Ope
nWithProgIds. The encrypted data includes the AES key, IV, and path that is used to find and decrypt the file
 containing Snake's kernel driver and kernel driver loader.

Attack Commands:
Executor: powershell
ElevationRequired: True
Command:
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; $randomBytes = New-Object Byte[] 0x1000; (New-O
bject Random).NextBytes($randomBytes); New-ItemProperty -Path $typicalPath -Name "AtomicSnake" -Value $random
Bytes -PropertyType Binary -Force | Out-Null

Cleanup Commands:
Command:
$typicalPath = "HKLM:\SOFTWARE\Classes\.wav\OpenWithProgIds"; Remove-ItemProperty -Path $typicalPath -Name "A
tomicSnake" -ErrorAction SilentlyContinue | Out-Null
[!!!!!!!!END TEST!!!!!!!]


PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1112 -TestNumbers 56
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1112-56 Snake Malware Registry Blob
Done executing test: T1112-56 Snake Malware Registry Blob
```

Queue File

The next host based indicator provided in the report was the queue file, which is described as “The last host-based artifact to discuss is the Queue File. Typically, this file has been found within the %windows%\Registration directory with the format of .crmlog, and is decrypted by Snake’s kernel driver. Due to the complexity and importance of the Queue File, its details are discussed at length in the following subsection.”

With that, we took the indicators and asked ChatGPT to assist.

Indicator of compromise:

- **Typical Name:** < RANDOM_GUID >..crmlog
- **Typical Path:** %windows\registration\
- **Unique Characteristics:** High Entropy, file attributes of hidden, system, and archive
- **Role:** Snake Queue File

First, we crafted a script to look for the indicators on disk.

Prompt

For our next script, the report talks about a Queue File.

Typical Name: < RANDOM_GUID >.<RANDOM_GUID>.crmlog

Typical Path: %windows%\registration\

Unique Characteristics: High Entropy, file attributes of hidden, system, and archive

Role: Snake Queue File

The Snake Queue File generally has a predictable path and filename structure, in addition to being high entropy. The Snake Queue File can be located by scanning all files in the typical queue path with filenames matching a regular expression that captures the typical naming convention. Files meeting these criteria should be scanned for high entropy, which is performed by the Yara rule below:

```
rule HighEntropy
{
 meta:
 description = "entropy rule"
 condition:
 math.entropy(0, filesize) >= 7.0
}
```

The following UNIX find command will scan files with names matching the GUID-based convention (note that the HighEntropy yara rule is assumed to be contained in a file named “1.yar”):

```
find /PATH/TO/WINDOWS_DIR -type f -regextype posix-egrep -iregex \
 '.*\/registration/(\{[0-9A-F]{8}\-([0-9A-F]{4}\-){3}[0-9A-F]{12}\}\.){2}crmlog' \
 -exec yara 1.yar {} \;
```

The following PowerShell command does the same:

```
Get-ChildItem -Recurse -File -Path %WINDOWS% | Where-Object {
```

$_.FullName -match
'(?i)/registration/(\{[0-9A-F]{8}\-([0-9A-F]{4}\-){3}[0-9A-F]{12}\}\.){2}crmlog$'
} | ForEach-Object {
yara 1.yar $_.FullName
}

With this info, expand on the provided powershell without using the yara rule.

Response

Here's a PowerShell script that searches for files in the specified typical path with names matching the GUID-based convention, checks if they have the attributes Hidden, System, and Archive, and calculates the entropy of the files:

```powershell
# Set search parameters
$typicalPath = Join-Path $env:windir "registration"
$filenamePattern = '^.*\\registration\\(\{[0-9A-F]{8}\-([0-9A-F]{4}\-){3}[0-9A-F]{12}\}\.){2}crmlog$'
$minimumEntropy = 7.0

# Function to calculate entropy of a byte array
function Get-Entropy($bytes) {
    $byteFrequency = @{}
    $byteCount = $bytes.Count

    foreach ($byte in $bytes) {
        if ($byteFrequency.ContainsKey($byte)) {
            $byteFrequency[$byte]++
        } else {
            $byteFrequency[$byte] = 1
        }
    }

    $entropy = 0
    foreach ($freq in $byteFrequency.Values) {
        $probability = $freq / $byteCount
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
```

```
}

# Function to check if a file has Hidden, System, and Archive attributes
function Test-FileAttributes($file) {
  $attributes = $file.Attributes
  return ($attributes -band [System.IO.FileAttributes]::Hidden) -and
      ($attributes -band [System.IO.FileAttributes]::System) -and
      ($attributes -band [System.IO.FileAttributes]::Archive)
}

# Find and process files in the typical path
Get-ChildItem -Recurse -File -Path $typicalPath | Where-Object {
  $_.FullName -match $filenamePattern -and Test-FileAttributes $_
} | ForEach-Object {
  $bytes = [System.IO.File]::ReadAllBytes($_.FullName)
  $entropy = Get-Entropy $bytes

  if ($entropy -ge $minimumEntropy) {
    Write-Host "Found matching file:"
    $_.FullName
    Write-Host "Entropy: $entropy"
  }
}
```

This script first defines the typical path, filename pattern, and minimum entropy for the Snake Queue Files. It then defines two functions: one to calculate the entropy of a byte array, and another to check if a file has the required attributes (Hidden, System, and Archive). The script then searches for files in the typical path that match the given criteria (name pattern, attributes, and entropy).

Please note that this script does not use the Yara rule mentioned in the report. The entropy calculation is done using the provided `Get-Entropy` function.

The initial script provided did not work, so the STRT iterated over it a few times, and ChatGPT even responded with:

“I apologize for the confusion. It appears that the issue is related to the regular expression pattern used to match the file paths. The regular expression pattern contains double backslashes (\\), which should be replaced with single backslashes (\). Here's the updated script with the corrected regular expression:” This appeared to be a regex issue in the script, and that is one of the few pitfalls of ChatGPT. Regex can be complicated and the regex is not correct 100% of the time.

The final working script provided by ChatGPT was:

```
# Define the typical path of the Snake Queue File
$filePath = "$env:windir\registration\"

# Create the folder if it doesn't exist
$null = New-Item -Path $filePath -ItemType Directory -Force

# Generate a random GUID
$guid = [guid]::NewGuid().ToString()

# Define the file name using the generated GUID and the regex pattern
$fileName = "$guid.$guid.crmlog"

# Create the file
$file = New-Item -Path $filePath -Name $fileName -ItemType "file"

# Set the file attributes to hidden, system, and archive
$file.Attributes = "Hidden", "System", "Archive"

function CalculateEntropy($bytes) {
    $byteCounts = @{}

    foreach ($byte in $bytes) {
        if ($byteCounts.ContainsKey($byte)) {
            $byteCounts[$byte]++
        } else {
            $byteCounts[$byte] = 1
        }
    }

    $entropy = 0
    $totalBytes = $bytes.Count

    foreach ($byteCount in $byteCounts.Values) {
        $probability = $byteCount / $totalBytes
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}
```

```
$minimumEntropy = 7

# Generate bytes with entropy above the minimum required
do {
   $randomBytes = New-Object Byte[] 0x1000
   (New-Object Random).NextBytes($randomBytes)
   $entropy = CalculateEntropy($randomBytes)
} while ($entropy -le $minimumEntropy)

# Write the random bytes with high entropy to the file
[System.IO.File]::WriteAllBytes($file.FullName, $randomBytes)

# Output the file path
Write-Host "File created with entropy above 7: $($file.FullName)"
```

The STRT then asked for a way to trigger the script.

Prompt

can you give me a one liner that matches what the script is looking for so that if my system doesn't have anything, I can at least trigger on a example.

Response

Here's a one-liner that creates an example file that meets the criteria (name pattern, attributes, and high entropy). This example file can be used for testing purposes.

```
$examplePath = Join-Path $env:windir "registration"; $guid1 = [guid]::NewGuid().ToString(); $guid2 = [guid]::NewGuid().ToString(); $exampleName = "$guid1.$guid2.crmlog"; $exampleFullPath = Join-Path $examplePath $exampleName; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); [System.IO.File]::WriteAllBytes($exampleFullPath, $randomBytes); $fileInfo = Get-Item $exampleFullPath; $fileInfo.Attributes = $fileInfo.Attributes -bor [System.IO.FileAttributes]::Hidden -bor [System.IO.FileAttributes]::System -bor [System.IO.FileAttributes]::Archive
```

This one-liner creates a new file at "%windows%\registration\{GUID1}.{GUID2}.crmlog" with 0x1000 random bytes and sets the file attributes to Hidden, System, and Archive. You can use this to test the script provided in the previous answer.

Please note that creating such files can have unintended side effects. Always be cautious when creating files on your system and make sure you understand the implications. When you are done testing, you can remove the test file using the following one-liner:

```
Remove-Item -Path $exampleFullPath -Force
```

This one-liner will delete the example file created earlier.

Creating the file on disk:

```
PS C:\Users\Administrator\Desktop> $examplePath = Join-Path $env:windir "registration"; $guid1 = [guid]::NewG
uid().ToString(); $guid2 = [guid]::NewGuid().ToString(); $exampleName = "$guid1.$guid2.crmlog"; $exampleFullP
ath = Join-Path $examplePath $exampleName; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextB
ytes($randomBytes); [System.IO.File]::WriteAllBytes($exampleFullPath, $randomBytes); $fileInfo = Get-Item $ex
ampleFullPath; $fileInfo.Attributes = $fileInfo.Attributes -bor [System.IO.FileAttributes]::Hidden -bor [Syst
em.IO.FileAttributes]::System -bor [System.IO.FileAttributes]::Archive
PS C:\Users\Administrator\Desktop>
```

Next, we scan for it:

```
PS C:\Users\Administrator\Desktop> .\queue_file_hunt.ps1
File created with entropy above 7: C:\Windows\registration\5eec2f43-e0d2-45e7-8c31-9017bc3e33fb.5eec2f43-e0d2
-45e7-8c31-9017bc3e33fb.crmlog
PS C:\Users\Administrator\Desktop>
```

Atomic Red Team test, T1027 test number 9, simulates the activity.

Atomic Test #9 - Snake Malware Encrypted crmlog file

The following Atomic Test will create a file with a specific name and sets its attributes to Hidden, System, and Archive. This was related to the Snake Malware campaign and is later decrypted by Snake's kernel driver. Snake Malware - CISA

Supported Platforms: Windows

auto_generated_guid: 7e47ee60-9dd1-4269-9c4f-97953b183268

Attack Commands: Run with `powershell`! Elevation Required (e.g. root or admin)

```
$file = New-Item $env:windir\registration\04e53197-72be-4dd8-88b1-533fe6eed577.04e53197-72be-4dd8-88b1-533fe6eed577.
```

Cleanup Commands:

```
$fileNameToDelete = '04e53197-72be-4dd8-88b1-533fe6eed577.04e53197-72be-4dd8-88b1-533fe6eed577.crmlog'; $filePathToD
```

Running the following with Invoke-AtomicTest:

Invoke-AtomicTest T1027 -testnumbers 9 -ShowDetails

Invoke-AtomicTest T1027 -testnumbers 9

```
PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1027 -testnumbers 9 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: Obfuscated Files or Information T1027
Atomic Test Name: Snake Malware Encrypted crmlog file
Atomic Test Number: 9
Atomic Test GUID: 7e47ee60-9dd1-4269-9c4f-97953b183268
Description: The following Atomic Test will create a file with a specific name and sets its attributes to Hid
den, System, and Archive. This was related to the Snake Malware campaign and is later decrypted by Snake's ke
rnel driver. [Snake Malware - CISA](https://media.defense.gov/2023/May/09/2003218554/-1/-1/0/JOINT_CSA_HUNTIN
G_RU_INTEL_SNAKE_MALWARE_20230509.PDF)

Attack Commands:
Executor: powershell
ElevationRequired: True
Command:
$file = New-Item $env:windir\registration\04e53197-72be-4dd8-88b1-533fe6eed577.04e53197-72be-4dd8-88b1-533fe6
eed577.crmlog; $file.Attributes = 'Hidden', 'System', 'Archive'; Write-Host "File created: $($file.FullName)"

Cleanup Commands:
Command:
$fileNameToDelete = '04e53197-72be-4dd8-88b1-533fe6eed577.04e53197-72be-4dd8-88b1-533fe6eed577.crmlog'; $file
PathToDelete = "$env:windir\registration\"; $fullPathToDelete = Join-Path $filePathToDelete $fileNameToDelete
; if (Test-Path $fullPathToDelete) { Remove-Item -Path $fullPathToDelete -Force; Write-Host "File deleted: $f
ullPathToDelete" } else { Write-Host "File not found: $fullPathToDelete" }
[!!!!!!!!END TEST!!!!!!!]


PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1027 -testnumbers 9
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1027-9 Snake Malware Encrypted crmlog file
File created: C:\Windows\registration\04e53197-72be-4dd8-88b1-533fe6eed577.04e53197-72be-4dd8-88b1-533fe6eed5
77.crmlog
Done executing test: T1027-9 Snake Malware Encrypted crmlog file
```

Coadmin

Per the CISA report, Snake's installer drops the kernel driver and a custom DLL which is used to load the driver into a single AES encrypted file on disk. Typically, this file is named "comadmin.dat" and is stored in the %windows%\system32\Com directory. The STRT took this as part of the requirement that we shared with ChatGPT to generate this behavior.

- **Typical Name:** comadmin.dat
- **Typical Path:** %windows%\system32\Com
- **Unique Characteristics:** High Entropy
- **Role:** Houses Snake's kernel driver and the driver's loader

As before, our first prompt is to identify if Snake malware was present on the endpoint.

Prompt
On to the next script. the report says Comadmin attributes are: Typical Name: comadmin.dat Typical Path: %windows%\system32\Com Unique Characteristics: High Entropy Role: Houses Snake's kernel driver and the driver's loader The Snake Comadmin file can be found using analogous techniques to that presented above for locating the Snake Queue File. The following UNIX find command will do so: find /PATH/TO/WINDOWS -type f -regextype posix-egrep -iregex \\ '.*\/system32/Com/comadmin\\.dat' \\ -exec yara 1.yar {} \\; The following PowerShell command does the same:

Get-ChildItem -Recurse -File -Path %WINDOWS% | Where-Object {
 $_.FullName -match '(?i)/system32/Com/comadmin\.dat$'
} | ForEach-Object {
 yara 1.yar $_.FullName

Use the powershell script provided
Get-ChildItem -Recurse -File -Path %WINDOWS% | Where-Object {
 $_.FullName -match '(?i)/system32/Com/comadmin\.dat$'

to write our script to find the .dat file. In addition, add write-hosts

Response

Here's a PowerShell script to find the `comadmin.dat` file, check for high entropy, and provide status updates using `Write-Host`:

```powershell
function CalculateEntropy($bytes) {
    $byteCounts = @{}

    foreach ($byte in $bytes) {
        if ($byteCounts.ContainsKey($byte)) {
            $byteCounts[$byte]++
        } else {
            $byteCounts[$byte] = 1
        }
    }

    $entropy = 0
    $totalBytes = $bytes.Count

    foreach ($byteCount in $byteCounts.Values) {
        $probability = $byteCount / $totalBytes
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}
```

```
$windowsPath = $env:windir
$filenamePattern = '(?i)/system32/Com/comadmin\.dat$'

Write-Host "Scanning for comadmin.dat files..."

$comadminFiles = Get-ChildItem -Recurse -File -Path $windowsPath | Where-Object {
    $_.FullName -match $filenamePattern
}

if ($comadminFiles) {
    Write-Host "Found comadmin.dat files. Checking entropy..."

    foreach ($file in $comadminFiles) {
        $bytes = [System.IO.File]::ReadAllBytes($file.FullName)
        $entropy = CalculateEntropy($bytes)

        if ($entropy -ge 7.0) {
            Write-Host "High entropy comadmin.dat found at: $($file.FullName)"
        } else {
            Write-Host "comadmin.dat with low entropy found at: $($file.FullName)"
        }
    }
} else {
    Write-Host "No comadmin.dat files found."
}
```

This script scans the `%WINDOWS%` directory for `comadmin.dat` files, calculates the entropy for each file, and outputs the status with `Write-Host`.

The STRT went back to ChatGPT two more times to add some additional logic and the final script was:

```
function CalculateEntropy($bytes) {
    $byteCounts = @{}

    foreach ($byte in $bytes) {
        if ($byteCounts.ContainsKey($byte)) {
            $byteCounts[$byte]++
        } else {
```

```
            $byteCounts[$byte] = 1
        }
    }

    $entropy = 0
    $totalBytes = $bytes.Count

    foreach ($byteCount in $byteCounts.Values) {
        $probability = $byteCount / $totalBytes
        $entropy -= $probability * [Math]::Log($probability, 2)
    }

    return $entropy
}

$comPath = Join-Path $env:windir "system32\Com"
$filenamePattern = 'comadmin\.dat$'

Write-Host "Scanning for comadmin.dat files in $comPath..."

$comadminFiles = Get-ChildItem -File -Path $comPath | Where-Object {
    $_.Name -match $filenamePattern
}

if ($comadminFiles) {
    Write-Host "Found comadmin.dat files. Checking entropy..."

    foreach ($file in $comadminFiles) {
        $bytes = [System.IO.File]::ReadAllBytes($file.FullName)
        $entropy = CalculateEntropy($bytes)

        if ($entropy -ge 7.0) {
            Write-Host "High entropy comadmin.dat found at: $($file.FullName)"
        } else {
            Write-Host "comadmin.dat with low entropy found at: $($file.FullName)"
        }
    }
} else {
    Write-Host "No comadmin.dat files found in $comPath."
}
```

To test scanning:

```
PS C:\Users\Administrator\Desktop> .\comadmin.ps1
Scanning for comadmin.dat files in C:\Windows\system32\Com...
No comadmin.dat files found in C:\Windows\system32\Com.
PS C:\Users\Administrator\Desktop>
```

We found no file on disk.

For the next prompt, we want to be able to lay a file on disk that matches the scanning script.

Prompt
can you give me a one liner that matches what the script is looking for so that if my system doesn't have anything, i can at least trigger on a example
Response
Here's a one-liner to create a high entropy \`comadmin.dat\` file in the correct location: \`\`\` $examplePath = Join-Path $env:windir "system32\Com"; if (-not (Test-Path $examplePath)) { New-Item -ItemType Directory -Path $examplePath | Out-Null }; $exampleName = "comadmin.dat"; $exampleFullPath = Join-Path $examplePath $exampleName; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); [System.IO.File]::WriteAllBytes($exampleFullPath, $randomBytes) \`\`\`  This one-liner will create a \`comadmin.dat\` file with high entropy in the \`%windows%\system32\Com\` directory. If the directory doesn't exist, it will create it.

Creating the file and running the scan again:

```
PS C:\Users\Administrator\Desktop> $examplePath = Join-Path $env:windir "system32\Com"; if (-not (Test-Path $examplePath)) { New-Item -ItemType Directory -Path $examplePath | Out-Null }; $exampleName = "comadmin.dat";
$exampleFullPath = Join-Path $examplePath $exampleName; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); [System.IO.File]::WriteAllBytes($exampleFullPath, $randomBytes)
PS C:\Users\Administrator\Desktop> .\comadmin.ps1
Scanning for comadmin.dat files in C:\Windows\system32\Com...
Found comadmin.dat files. Checking entropy...
High entropy comadmin.dat found at: C:\Windows\system32\Com\comadmin.dat
```

This was the final part we needed to put a file matching the report on disk and our scanner to look for and detect it.

The Atomic Test is under T1547.006 and may be simulated with the following command:
Invoke-AtomicTest T1547.006 -TestNumbers 4

Atomic Test #4 - Snake Malware Kernel Driver Comadmin

The following Atomic Test will write an file, comadmin.dat, to disk. From the report, Snake's installer drops the kernel driver and a custom DLL which is used to load the driver into a single AES encrypted file on disk. Typically, this file is named "comadmin.dat" and is stored in the %windows%\system32\Com directory. This Atomic Test will write a hardcoded named file to disk in the com directory named comadmin.dat. Snake Malware - CISA

Supported Platforms: Windows

auto_generated_guid: e5cb5564-cc7b-4050-86e8-f2d9eec1941f

Attack Commands: Run with `powershell`! Elevation Required (e.g. root or admin)

```
$examplePath = Join-Path $env:windir "system32\Com"; if (-not (Test-Path $examplePath)) { New-Item -ItemType Directo
```

Cleanup Commands:

```
$examplePath = Join-Path $env:windir "system32\Com"; $exampleName = "comadmin.dat"; $exampleFullPath = Join-Path $ex
```

Invoke-AtomicTest T1547.006 -TestNumbers 4 -ShowDetails

Invoke-AtomicTest T1547.006 -TestNumbers 4

```
PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1547.006 -TestNumbers 4 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: Boot or Logon Autostart Execution: Kernel Modules and Extensions T1547.006
Atomic Test Name: Snake Malware Kernel Driver Comadmin
Atomic Test Number: 4
Atomic Test GUID: e5cb5564-cc7b-4050-86e8-f2d9eec1941f
Description: The following Atomic Test will write an file, comadmin.dat, to disk. From the report, Snake's in
staller drops the kernel driver and a custom DLL which is used to load the driver into a single AES encrypted
 file on disk. Typically, this file is named â€œcomadmin.datâ€ and is stored in the %windows%\system32\Com d
irectory.  This Atomic Test will write a hardcoded named file to disk in the com directory named comadmin.dat
. [Snake Malware - CISA](https://media.defense.gov/2023/May/09/2003218554/-1/-1/0/JOINT_CSA_HUNTING_RU_INTEL_
SNAKE_MALWARE_20230509.PDF)

Attack Commands:
Executor: powershell
ElevationRequired: True
Command:
$examplePath = Join-Path $env:windir "system32\Com"; if (-not (Test-Path $examplePath)) { New-Item -ItemType
Directory -Path $examplePath | Out-Null }; $exampleName = "comadmin.dat"; $exampleFullPath = Join-Path $examp
lePath $exampleName; $randomBytes = New-Object Byte[] 0x1000; (New-Object Random).NextBytes($randomBytes); [S
ystem.IO.File]::WriteAllBytes($exampleFullPath, $randomBytes)

Cleanup Commands:
Command:
$examplePath = Join-Path $env:windir "system32\Com"; $exampleName = "comadmin.dat"; $exampleFullPath = Join-P
ath $examplePath $exampleName; if (Test-Path $exampleFullPath) { Remove-Item $exampleFullPath -Force }
[!!!!!!!!END TEST!!!!!!!]


PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1547.006 -TestNumbers 4
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1547.006-4 Snake Malware Kernel Driver Comadmin
Done executing test: T1547.006-4 Snake Malware Kernel Driver Comadmin
```

Werfault

Indicator of compromise:

- **Typical Name:** Werfault.exe
- **Typical Path:** %windows%\WinSxS\x86_microsoft-windowserrorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a0502\
- **Unique Characteristics:** Icon is different than that of a valid Windows Werfault.exe file
- **Role:** Persistence mechanism

Like before, we start with the indicators of compromise prompt.

Prompt

The next script we want to emulate of Snake is the following:

The Snake version primarily discussed in this advisory registers a service to maintain persistence on a system. Typically, this service is named “WerFaultSvc,” which we assess was used to blend in with the legitimate Windows service WerSvc. On boot, this service will execute Snake’s WerFault.exe, which Snake developers chose to hide among the numerous valid Windows “WerFault.exe” files in the %windows%\WinSxS\ directory. Executing WerFault.exe will start the process of decrypting Snake’s components and loading them into memory.

Response

To emulate the Snake's persistence mechanism using a service named "WerFaultSvc" and executing "WerFault.exe" on boot, you can follow these steps:

1. Create a fake "WerFault.exe" file and place it in the "%windows%\WinSxS\" directory.
2. Create a new Windows service named "WerFaultSvc" that executes the fake "WerFault.exe" on system startup.

Step 1: Create a fake "WerFault.exe" file

You can create a simple PowerShell script or a batch file that does nothing and save it as "WerFault.exe". Here's an example of a PowerShell script that does nothing:

```powershell
# Do nothing
```

Save this script as "WerFault.exe" and place it in the "%windows%\WinSxS\" directory.

Step 2: Create a new Windows service

You can use the `New-Service` cmdlet in PowerShell to create a new service that starts the fake "WerFault.exe" on system startup. Make sure to run PowerShell as an administrator to create the service. Replace `<path_to_WerFault.exe>` with the actual path of the "WerFault.exe" file you created in step 1.

```powershell
New-Service -Name "WerFaultSvc" -BinaryPathName "<path_to_WerFault.exe>" -StartupType Automatic -Description "Fake WerFaultSvc for Snake emulation"
```

Now, the "WerFaultSvc" service will execute the fake "WerFault.exe" on system startup, emulating Snake's persistence mechanism. To remove the service later, use the `Remove-Service` cmdlet:

```powershell
Remove-Service -Name "WerFaultSvc"
```

Make sure to run PowerShell as an administrator when removing the service.

Follow-up Prompt

create a single all in one script to do all this

Response

Here's a single script that combines all the steps to create a fake "WerFault.exe" file, place it in the "%windows%\WinSxS\" directory, and create a new Windows service named "WerFaultSvc" that executes the fake "WerFault.exe" on system startup. Make sure to run PowerShell as an administrator to create the service.

```powershell
# Check if running as administrator
if (-NOT ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
    Write-Host "Please run PowerShell as an administrator to create the service." -ForegroundColor Red
    exit
}

# Create a fake WerFault.exe PowerShell script that does nothing
$werFaultContent = @"
# Do nothing
"@

# Save the WerFault.exe script to the WinSxS directory
$winSxSPath = Join-Path $env:windir "WinSxS"
$werFaultPath = Join-Path $winSxSPath "WerFault.exe"

Set-Content -Path $werFaultPath -Value $werFaultContent

# Create a new Windows service that starts the fake WerFault.exe on system startup
```

~~~~
New-Service -Name "WerFaultSvc" -BinaryPathName "powershell.exe -ExecutionPolicy Bypass -File $werFaultPath" -StartupType Automatic -Description "Fake WerFaultSvc for Snake emulation"

Write-Host "WerFaultSvc service created and configured to start on system startup." -ForegroundColor Green
```

To remove the service and the fake "WerFault.exe" file later, use the following script. Make sure to run PowerShell as an administrator when removing the service.

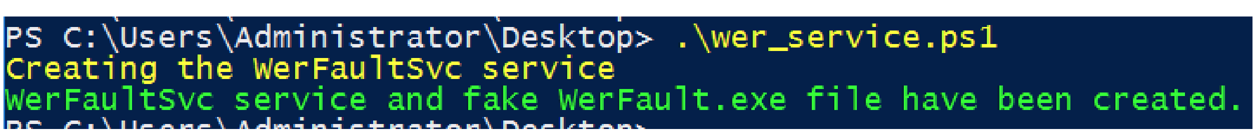
```powershell
# Check if running as administrator
if (-NOT ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
    Write-Host "Please run PowerShell as an administrator to remove the service." -ForegroundColor Red
    exit
}

# Remove the WerFaultSvc service
Remove-Service -Name "WerFaultSvc"

# Remove the fake WerFault.exe file
$winSxSPath = Join-Path $env:windir "WinSxS"
$werFaultPath = Join-Path $winSxSPath "WerFault.exe"

Remove-Item -Path $werFaultPath

Write-Host "WerFaultSvc service and the fake WerFault.exe file have been removed." -ForegroundColor Green
```
~~~~

The Service creation took a bit of back and forth to get right. Many errors and certain PowerShell commandlets were not on Server 2019 (Splunk Attack Range’s Windows Endpoint Operating System). The STRT ended up with two script types. First, ChatGPT provided this:

```
# Check if running as administrator
if (-NOT ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
    Write-Host "Please run PowerShell as an administrator to create the service and fake WerFault.exe file." -ForegroundColor Red
    exit
}
```

```
# Original path in WinSxS folder
#$winSxSPath = Join-Path $env:windir "WinSxS"
#$typicalPath = Join-Path $winSxSPath "x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050"

# Create the custom WerFault.exe path
$customPath = Join-Path $env:ProgramData "CustomWerFault"
$werFaultPath = Join-Path $customPath "WerFault.exe"

if (-not (Test-Path $customPath)) {
    Write-Host "Creating the custom path at: $customPath" -ForegroundColor Yellow
    New-Item -Path $customPath -ItemType Directory | Out-Null
}

# Create the fake WerFault.exe file
#Write-Host "Creating the fake WerFault.exe file at: $werFaultPath" -ForegroundColor Yellow
#New-Item -Path $werFaultPath -ItemType File -Force | Out-Null

# Create the WerFaultSvc service
$serviceName = "WerFaultSvc"
$serviceDisplayName = "WerFault Service"
$serviceDescription = "Example Snake-like service"

Write-Host "Creating the WerFaultSvc service" -ForegroundColor Yellow
#New-Service -Name $serviceName -BinaryPathName $werFaultPath -DisplayName $serviceDisplayName -Description $serviceDescription -StartupType Automatic | Out-Null
New-Service -Name $serviceName -BinaryPathName $typicalPath -DisplayName $serviceDisplayName -Description $serviceDescription -StartupType Automatic | Out-Null
Write-Host "WerFaultSvc service and fake WerFault.exe file have been created." -ForegroundColor Green
```

This script will do everything we requested. Note that we added a few comments to use the WinSXS path or program data, this was due to permissions and path issues in WinSXS.



The second script ChatGPT provided had two PowerShell functions:

```
function CreateService {
    param (
        [Parameter(Mandatory = $true)]
        [ValidateSet("typical", "custom")]
        [string]$pathType
    )

    # Check if running as administrator
    if (-NOT
([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Pri
ncipal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Please run PowerShell as an administrator to create the service and fake WerFault.exe file." -
ForegroundColor Red
        exit
    }

    # Original path in WinSxS folder
    $winSxSPath = Join-Path $env:windir "WinSxS"
    $typicalPath = Join-Path $winSxSPath "x86_microsoft-windows-
errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe"

    if ($pathType -eq "custom") {
        # Create the custom WerFault.exe path
        $customPath = Join-Path $env:ProgramData "CustomWerFault"
        $werFaultPath = Join-Path $customPath "WerFault.exe"

        if (-not (Test-Path $customPath)) {
            Write-Host "Creating the custom path at: $customPath" -ForegroundColor Yellow
            New-Item -Path $customPath -ItemType Directory | Out-Null
        }

        # Create the fake WerFault.exe file
        Write-Host "Creating the fake WerFault.exe file at: $werFaultPath" -ForegroundColor Yellow
        New-Item -Path $werFaultPath -ItemType File -Force | Out-Null
    } else {
        $werFaultPath = $typicalPath
    }

    # Create the WerFaultSvc service
    $serviceName = "WerFaultSvc"
```

```
    $serviceDisplayName = "WerFault Service"
    $serviceDescription = "Example Snake-like service"

    Write-Host "Creating the WerFaultSvc service" -ForegroundColor Yellow
    New-Service -Name $serviceName -BinaryPathName $werFaultPath -DisplayName $serviceDisplayName -
Description $serviceDescription -StartupType Automatic | Out-Null

    Write-Host "WerFaultSvc service and fake WerFault.exe file have been created." -ForegroundColor Green
}



# Function to delete the service
function DeleteService {
    # Check if running as administrator
    if (-NOT
([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Pri
ncipal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Please run PowerShell as an administrator to remove the service and fake WerFault.exe file."
-ForegroundColor Red
        exit
    }

    # Stop and remove the WerFaultSvc service
    $serviceName = "WerFaultSvc"

    if (Get-Service $serviceName -ErrorAction SilentlyContinue) {
        Write-Host "Stopping and removing the WerFaultSvc service" -ForegroundColor Yellow
        Stop-Service $serviceName -Force -ErrorAction SilentlyContinue
        sc.exe delete $serviceName | Out-Null
    }

    # Delete the fake WerFault.exe file
    $werFaultPath = Join-Path $env:ProgramData "CustomWerFault\WerFault.exe"

    if (Test-Path $werFaultPath) {
        Write-Host "Deleting the fake WerFault.exe file at: $werFaultPath" -ForegroundColor Yellow
        Remove-Item $werFaultPath -Force
    }

    # Delete the custom path if it's empty
    $customPath = Join-Path $env:ProgramData "CustomWerFault"
```

```
    if ((Test-Path $customPath) -and (!(Get-ChildItem $customPath))) {
        Write-Host "Deleting the empty custom path at: $customPath" -ForegroundColor Yellow
        Remove-Item $customPath -Force
    }

    Write-Host "WerFaultSvc service and fake WerFault.exe file have been removed." -ForegroundColor Green
}
```

The first function has a pathType option to specify custom or typical (typical, being found in the report).

```
PS C:\Users\Administrator\Desktop> CreateService -pathType typical
Creating the WerFaultSvc service
WerFaultSvc service and fake WerFault.exe file have been created.
```

And the DeleteService

```
PS C:\Users\Administrator\Desktop> DeleteService
Stopping and removing the WerFaultSvc service
WerFaultSvc service and fake WerFault.exe file have been removed.
PS C:\Users\Administrator\Desktop>
```

While developing the Atomic Test, we were able to iterate with ChatGPT on two methods. One with PowerShell and the other with sc.exe

PowerShell
New-Service -Name "WerFaultSvc" -BinaryPathName "$env:windir\WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe" -DisplayName "WerFault Service" -Description "Example Snake-like service" -StartupType Automatic
Sc.exe
sc.exe create "WerFaultSvc" binPath= "$env:windir\WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe" DisplayName= "WerFault Service" start= auto

This provides two ways to simulate the same behavior.

Atomic Red Team [T1569.002](T1569.002) test number 6 was produced to allow for testing with Invoke-AtomicRedTeam.

Atomic Test #6 - Snake Malware Service Create

The following Atomic Test will create a new service named WerFaultSvc with a binary path of WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe. This was recently seen in the Snake Malware report. Upon execution, sc.exe will create a new service named WerFaultSvc with a bin path \WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe and a display name of WerFault Service.
Snake Malware - CISA

Supported Platforms: Windows

auto_generated_guid: b8db787e-dbea-493c-96cb-9272296ddc49

Attack Commands: Run with `command_prompt`! Elevation Required (e.g. root or admin)

```
sc.exe create "WerFaultSvc" binPath= "$env:windir\WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_
```

Cleanup Commands:

```
sc.exe delete "WerFaultSvc"
```

The following will run the simulation:

Invoke-AtomicTest T1569.002 -TestNumbers 6 -ShowDetails

Invoke-AtomicTest T1569.002 -TestNumbers 6

```
PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1569.002 -TestNumbers 6 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[********BEGIN TEST*******]
Technique: System Services: Service Execution T1569.002
Atomic Test Name: Snake Malware Service Create
Atomic Test Number: 6
Atomic Test GUID: b8db787e-dbea-493c-96cb-9272296ddc49
Description: The following Atomic Test will create a new service named WerFaultSvc with a binary path of WinS
xS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.
exe. This was recently seen in the Snake Malware report.  Upon execution, sc.exe will create a new service na
med WerFaultSvc with a bin path \WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.
16384_none_a13f7e283339a050\WerFault.exe and a display name of WerFault Service. [Snake Malware - CISA](https
://media.defense.gov/2023/May/09/2003218554/-1/-1/0/JOINT_CSA_HUNTING_RU_INTEL_SNAKE_MALWARE_20230509.PDF)

Attack Commands:
Executor: command_prompt
ElevationRequired: True
Command:
sc.exe create "WerFaultSvc" binPath= "$env:windir\WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856a
d364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe" DisplayName= "WerFault Service" start= auto

Cleanup Commands:
Command:
sc.exe delete "WerFaultSvc"
[!!!!!!!!END TEST!!!!!!!]


PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1569.002 -TestNumbers 6
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1569.002-6 Snake Malware Service Create
[SC] CreateService SUCCESS
Done executing test: T1569.002-6 Snake Malware Service Create
PS C:\Users\Administrator\Desktop> Invoke-AtomicTest T1569.002 -TestNumbers 6 -Cleanup
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing cleanup for test: T1569.002-6 Snake Malware Service Create
[SC] DeleteService SUCCESS
Done executing cleanup for test: T1569.002-6 Snake Malware Service Create
```

Security Content

The Splunk Threat Research Team has curated relevant detections and tagged them to the [Snake Malware Analytic Story] to help security practitioners detect adversaries leveraging the Snake malware.

For this release, we used and considered the relevant data endpoint telemetry sources such as:

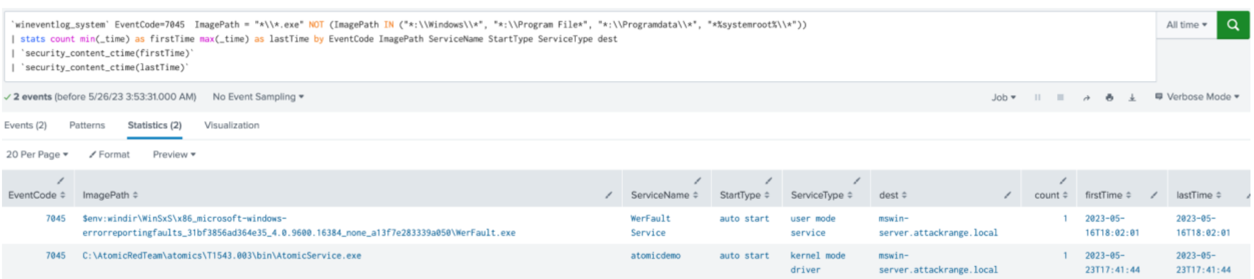
- Process Execution & Command Line Logging
- Sysmon, or any Common Information Model compliant EDR technology
- Windows Security Event Log

We used the EventID 7045 to develop the analytics that monitor service creations. The Snake malware was found to register a service located in the WinSxS path with a specific name. Holistically, it's best to monitor for any new service creations across the fleet and monitor for non-standard paths or binary names.

Windows Service Created with Suspicious Service Path

```
`wineventlog_system` EventCode=7045  ImagePath = "*\\*.exe" NOT (ImagePath IN ("*:\\Windows\\*", "*:\\Program File*", "*:\\Programdata\\*", "*%systemroot%\\*"))

| stats count min(_time) as firstTime max(_time) as lastTime by EventCode ImagePath ServiceName StartType ServiceType dest

| `security_content_ctime(firstTime)`

| `security_content_ctime(lastTime)`
```

This analytic identifies the creation of a new service that does not occur within these directories (":\Windows\", ":\Program File", ":\Programdata\", "%systemroot%\"). For Snake malware, it would create its service in the WinSxS path, which is uncommon.



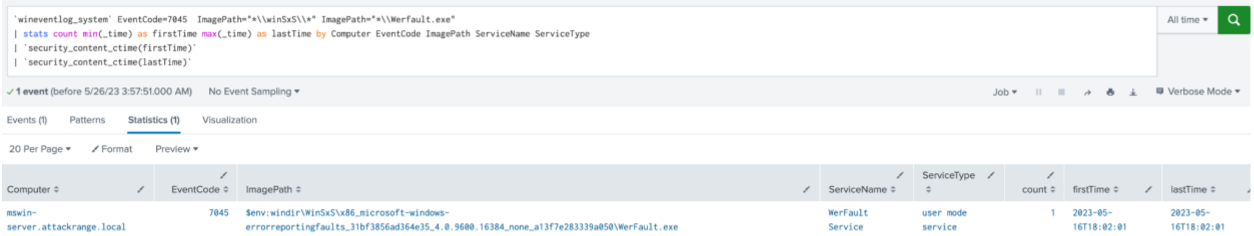
EventCode	ImagePath	ServiceName	StartType	ServiceType	dest	count	firstTime	lastTime
7045	$env:windir\WinSxS\x86_microsoft-windows-errorreportingfaults_31bf3856ad364e35_4.0.9600.16384_none_a13f7e283339a050\WerFault.exe	WerFault Service	auto start	user mode service	mswin-server.attackrange.local	1	2023-05-16T18:02:01	2023-05-16T18:02:01
7045	C:\AtomicRedTeam\atomics\T1543.003\bin\AtomicService.exe	atomicdemo	auto start	kernel mode driver	mswin-server.attackrange.local	1	2023-05-23T17:41:44	2023-05-23T17:41:44

Windows Snake Malware Service Create

For Snake malware, we created a more precise analytic based on the Snake behaviors outlined in the report and what we created during our simulation.

```
`wineventlog_system` EventCode=7045  ImagePath="*\\windows\\winSxS\\*" ImagePath="*\Werfault.exe"

| stats count min(_time) as firstTime max(_time) as lastTime by Computer EventCode ImagePath ServiceName ServiceType

| `security_content_ctime(firstTime)`
```

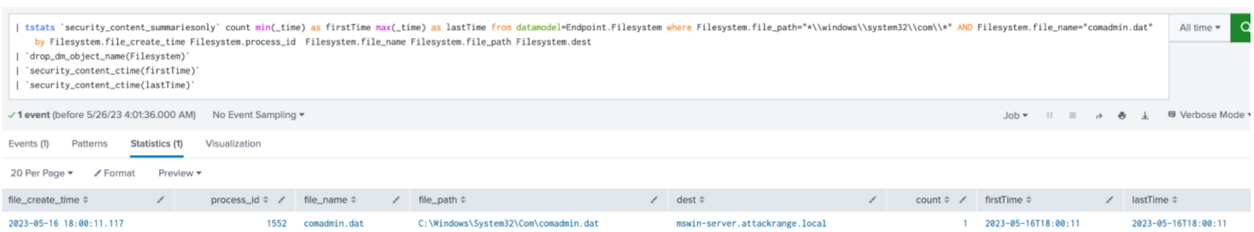
```
| `security_content_ctime(lastTime)`
```



Windows Snake Malware Kernel Driver Comadmin

This analytic identifies the comadmin.dat file written to disk, which is related to Snake Malware. From the report, Snake's installer drops the kernel driver and a custom DLL which is used to load the driver into a single AES encrypted file on disk. Typically, this file is named comadmin.dat and is stored in the %windows%\system32\Com directory.

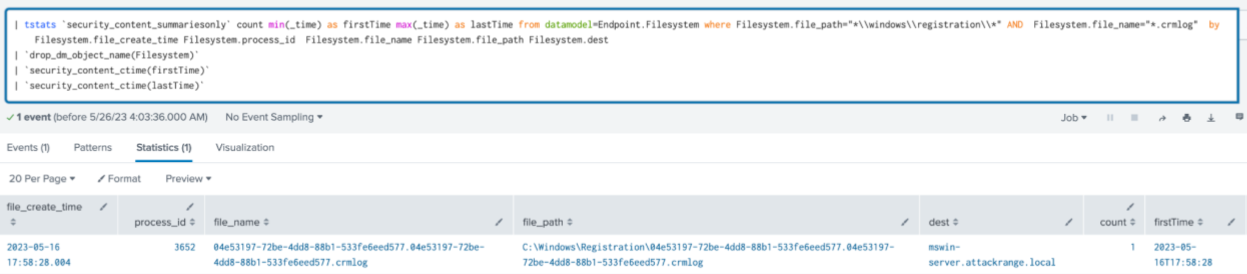
```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Filesystem where Filesystem.file_path="*\\windows\\system32\\com\\*" AND Filesystem.file_name="comadmin.dat"  by Filesystem.file_create_time Filesystem.process_id Filesystem.file_name Filesystem.file_path Filesystem.dest

| `drop_dm_object_name(Filesystem)`

| `security_content_ctime(firstTime)`

| `security_content_ctime(lastTime)`
```



Windows Snake Malware File Modification Crmlog

This analytic identifies a .crmlog written to windows\registration. Per the report, typically, this file has been found within the %windows%\Registration directory with the format of <RANDOM_GUID>.<RANDOM_GUID>.crmlog and is decrypted by Snake's kernel driver.

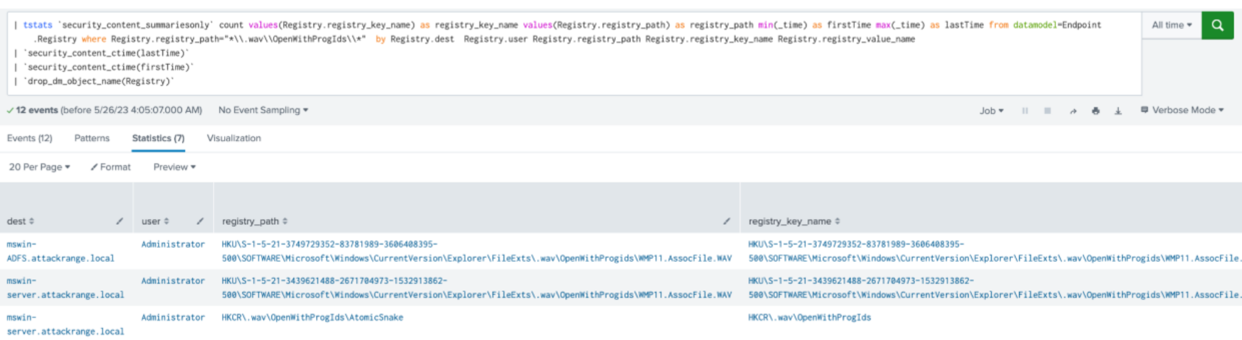
```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Filesystem where Filesystem.file_path="*\\windows\\registration\\*" AND Filesystem.file_name="*.crmlog"  by Filesystem.file_create_time Filesystem.process_id  Filesystem.file_name Filesystem.file_path Filesystem.dest

| `drop_dm_object_name(Filesystem)`

| `security_content_ctime(firstTime)`

| `security_content_ctime(lastTime)`
```



Windows Snake Malware Registry Modification wav OpenWithProgIds

This analytic identifies the registry being modified at .wav\OpenWithProgIds\, which is related to the Snake Malware campaign. Upon execution, Snake's WerFault.exe will attempt to decrypt an encrypted blob within the Windows registry that is typically found at HKLM:\SOFTWARE\Classes.wav\OpenWithProgIds. The encrypted data includes the AES key, IV, and path that is used to find and decrypt the file containing Snake's kernel driver and kernel driver loader.

```
| tstats `security_content_summariesonly` count values(Registry.registry_key_name) as registry_key_name values(Registry.registry_path) as registry_path min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Registry where Registry.registry_path="*\\.wav\\OpenWithProgIds\\*"  by Registry.dest Registry.user Registry.registry_path Registry.registry_key_name Registry.registry_value_name

| `security_content_ctime(lastTime)`

| `security_content_ctime(firstTime)`

| `drop_dm_object_name(Registry)`
```



Why Should You Care?

This security content enables security analysts, blue teamers and Splunk customers to identify **Snake Malware** by helping the community understand how tools like LLMs or ChatGPT can recreate endpoint artifacts without malicious software samples. By understanding the behaviors, we were able to generate telemetry and datasets to develop and test Splunk detections designed to defend and respond against this threat.

What Did We Learn and What Can We Improve?

The STRT is excited that what began as an experiment with a new tool quickly helped us build timely, relevant, verifiable content. Since our team firmly believes in publishing real datasets along with our content, being able to understand and test against the events generated by Snake Malware's behaviors is critical to ensuring the quality of that content. Collaborating with ChatGPT to build PowerShell Scripts that mimic Snake Malware behavior saved us hours of development, research, and testing effort. This approach can be a force multiplier for SOCs and Detection Engineers.

We also learned that both ChatGPT and our use of the tool have room for improvement. As always - trust but verify. The first response from ChatGPT rarely met all the objectives of the prompt. Iterative scrutiny and testing of the outputs, conversational feedback to ChatGPT, and even manual updates to PowerShell Scripts produced the best results. Additionally, understanding malware behavior and producing requirements is still a complicated task. ChatGPT output is entirely dependent on the quality of input, so standardization of prompts and methodologies will likely yield more consistent and better results.

Learn More

You can find the latest content about security analytic stories on GitHub and in Splunkbase. Splunk Security Essentials also has all these detections available via push update. For a full list of security content, check out the release notes on Splunk Docs.

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 Michael Haag, Eric McGinnis and Teoderick Contreras for authoring this document and the entire Splunk Threat Research Team for their contributions: Mauricio Velazco, Lou Stella, Bhavin Patel, Rod Soto, and Patrick Bareiss.