Threat Update: Industroyer2

splunk.com/en_us/blog/security/threat-update-industroyer2.html

June 23, 2022

By Splunk Threat Research Team June 23, 2022

The Splunk Threat Research Team (STRT) continues to monitor new relevant payloads to the ongoing conflict in Eastern Europe. One of these new payloads was found by the Ukranian CERT named “Industroyer2.” The name of this new payload references the original "Industroyer" malicious payload used against the country of Ukraine's power grid in 2016 and allegedly was able to affect a fifth of the power capacity of the city of Kyiv.

According to the recent Ukraine CERT and ESET report, Industroyer2 resembles the former Industroyer in functionality and is also targeting the electric grid containing commands targeting high-voltage electrical substations. It was reported that Industroyer2 was also used along with CaddyWiper, another payload recently addressed by the Splunk Threat Research Team. This payload — in combination with previous featured destructive payloads — targets CPEs. These customer premise devices such as modems, cable modems, and internet gateways are devices that provide connectivity to the great majority of commercial and residential customers, and speak to the attacker's intention of overwhelming or degrading the victim's infrastructure.

The following is an analysis of relevant detection opportunities of this payload and observed TTPs during the deployment of this payload.

Parameter Check

The first part of its code is checking parameters that can execute some of its features related to timing and logging. Below is the code screenshot of this checking with its 2 parameters.

```
if ( hMem )
{
  if ( pNumArgs )
  {
    str_minute_timer = (LPCWCH)mw_check_param((int)hMem, (int *)hMem + pNumArgs, (char)L"-t");
    if ( str_minute_timer )
    {
      v1 = mw_get_system_time_and_create_timer(v3, str_minute_timer);
      wrap_WaitForMultipleObjectsEx((int)v1);
      wrap_CloseHandle(v3);
    }
    if ( sub_403B60((int)hMem, (int)hMem + 4 * pNumArgs, (char)L"-o") )
    {
      v11 = mw_check_param((int)hMem, (int *)hMem + pNumArgs, (char)L"-o");
      if ( v11 )
      {
        wrap_InitializeCriticalSection_0();
        mw_generate_logs(v11);
      }
      v2 = wrap_InitializeCriticalSection_0();
      mw_write_console_logs((int)v2, "%d\n", 22);
    }
```

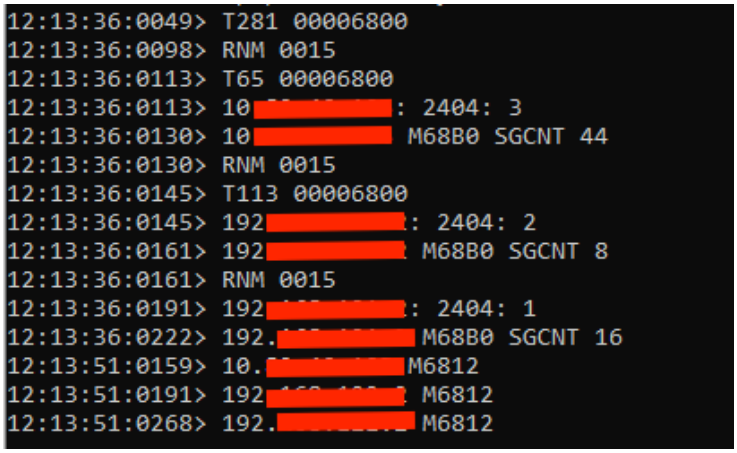
The first parameter is "-t" which will trigger a waiting timer relative to the current minute of the system time. For example, if your system time is 14:19:22 PM and you use this parameter with a value of 25 as the third parameter, it means it will wait 5 mins before it executes its code like the screenshot below.

```
Select Administrator: C:\Windows\system32\cmd.exe

C:\Temp>time
The current time is: 14:19:22.22
Enter the new time:

C:\Temp>industroyer2.exe -t 25
M88B8 - 00:05:30
^C
C:\Temp>
```

While the “-o” parameter is a feature to redirect its console logs to a debug log file you inputted as the 3rd parameter.

Console Logs

Upon executing this malware, it outputs some console logs with a customized code structure that tells something about what features it executes. Some of it will be discussed further in the next subheadings. Below is an example of the console logs during its execution.



Terminate Process and Rename Process File Path

This function enumerates all running processes in the targeted host and looks for the process named “PServiceControl.exe” and also the process name stated in its config data. It will also look for the file path of that process in a specific folder that is in the config file and rename it with “.MZ” file extension.

The code screenshot below shows the process termination and renaming of process file path. We can see in the code snippet the code “RNM” plus the last error code after the call MoveFileA() function that will be displayed in its console logs after executing this part of the code. You can see that in the console log screenshot earlier.

```
for ( dwProcessId = mw_EnumProcess((int)"PServiceControl.exe"); dwProcessId; dwProcessId = mw_EnumProcess(a1 + 0x10047) )
{
  hProcess = OpenProcess(1u, 0, dwProcessId);
  TerminateProcess(hProcess, 0);
}
while ( 1 )
{
  dwProcessIda = mw_EnumProcess(a1 + 0x10047);
  if ( !dwProcessIda )
    break;
  v4 = OpenProcess(1u, 0, dwProcessIda);
  TerminateProcess(v4, 0);
}
if ( *(_BYTE *)(a1 + 0x10154) && a1 != 0xFFFEFEAB )
{
  lpExistingFileName = (const CHAR *)wrap_heapAlloc(0x100u);
  lpNewFileName = (const CHAR *)wrap_heapAlloc(0x100u);
  lpMem = wrap_heapAlloc(0x100u);
  sub_402540(lpExistingFileName, 256);
  sub_402540(lpNewFileName, 256);
  sub_402540(lpMem, 256);
  sub_401A10(lpExistingFileName, a1 + 65877);
  sub_401A10(lpNewFileName, a1 + 65877);
  sub_405360(lpExistingFileName, "\\");
  sub_405360(lpNewFileName, "\\");
  sub_405360(lpExistingFileName, a1 + 65607);
  sub_405360(lpNewFileName, a1 + 65607);
  sub_405360(lpNewFileName, ".MZ");
  MoveFileA(lpExistingFileName, lpNewFileName);
  v3 = GetLastError();
  v1 = wrap_SleepAndInit();
  mw_prepare_console_logs((int)v1, " RNM %04x \n", v3);
  mw_memsetHeap(lpExistingFileName);
  mw_memsetHeap(lpNewFileName);
  mw_memsetHeap(lpMem);
}
return 1;
```

HardCoded Configuration Data

This malware contains hardcoded configuration files that will be parsed with the help of CommandLineToArgvW() function and put in a structure that will be used later in its code. Below is the screenshot of the parsing function.

```
lp_cmd_argvW = CommandLineToArgvW(lpCmdLine, &pNumArgs);
if ( lp_cmd_argvW )
{
  *(_DWORD *)ctr = 0;
  while ( *(_DWORD *)ctr < (unsigned int)pNumArgs )
  {
    if ( mw_parse_ip_addr_in_cfg(*((LPCWCH *)lp_cmd_argvW + *(_DWORD *)ctr)) )
    {
      lpMem = wrap_heap_alloc(0x48u);
      if ( lpMem )
      {
        lpParameter = wrap_heap_alloc(0x10D70u);
        v28 = 1;
        *lpMem = wrap_wide_to_multi_bytes(*((LPCWCH *)lp_cmd_argvW + *(_DWORD *)ctr));// ip_addr
        if ( ++*(_DWORD *)ctr < (unsigned int)pNumArgs && v28 )
        {
          lpMem[1] = wrap_wide_to_multi_bytes(*((LPCWCH *)lp_cmd_argvW + *(_DWORD *)ctr));// port
          ++*(_DWORD *)ctr;
        }
        else
```

The config data contains values and checks that this payload uses through its execution. We saw four main components of its three configuration data settings that are hardcoded to its data section like the screenshot below: The first component is the IP address of devices where it tries to communicate via

IEC-104 protocol, the next one is the port number (2404), third is the process name (PService_PPD.exe) it tries to kill aside from “PServicecontrol.exe” and a file path (D:\OIK\DevCounter) where it locates the process file path it tries to kill to rename it with .MZ file extension.

Detections

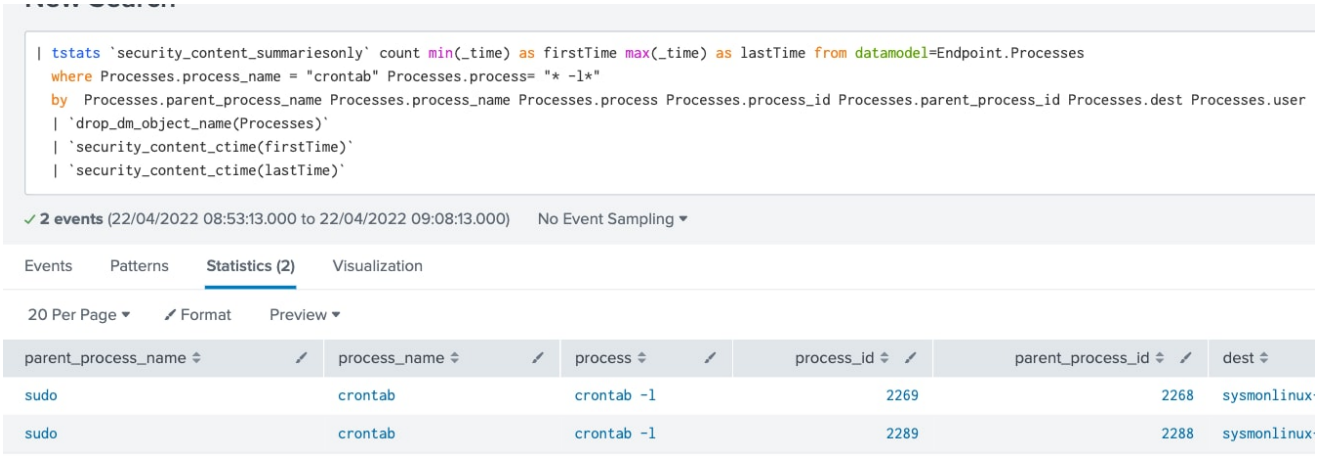
Below are the detections related to the Industroyer2 malware and other components found during the attack that was mentioned in the ESET blog and CERT-UA blog.

Linux Adding Crontab Using List Parameter

This analytic identifies a suspicious cron jobs modification using crontab list parameters. This command line parameter can be abused by malware like Industroyer2, adversaries, and red teamers to add a crontab entry to their malicious code to execute to the schedule they want.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes

  where Processes.process_name = "crontab" Processes.process= "* -l*"

  by  Processes.parent_process_name Processes.process_name Processes.process
Processes.process_id Processes.parent_process_id Processes.dest Processes.user

  | `drop_dm_object_name(Processes)`

  | `security_content_ctime(firstTime)`
```

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



```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
  where Processes.process_name = "crontab" Processes.process= "* -l*"
  by  Processes.parent_process_name Processes.process_name Processes.process Processes.process_id Processes.parent_process_id Processes.dest Processes.user
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ 2 events (22/04/2022 08:53:13.000 to 22/04/2022 09:08:13.000) No Event Sampling

Events Patterns Statistics (2) Visualization

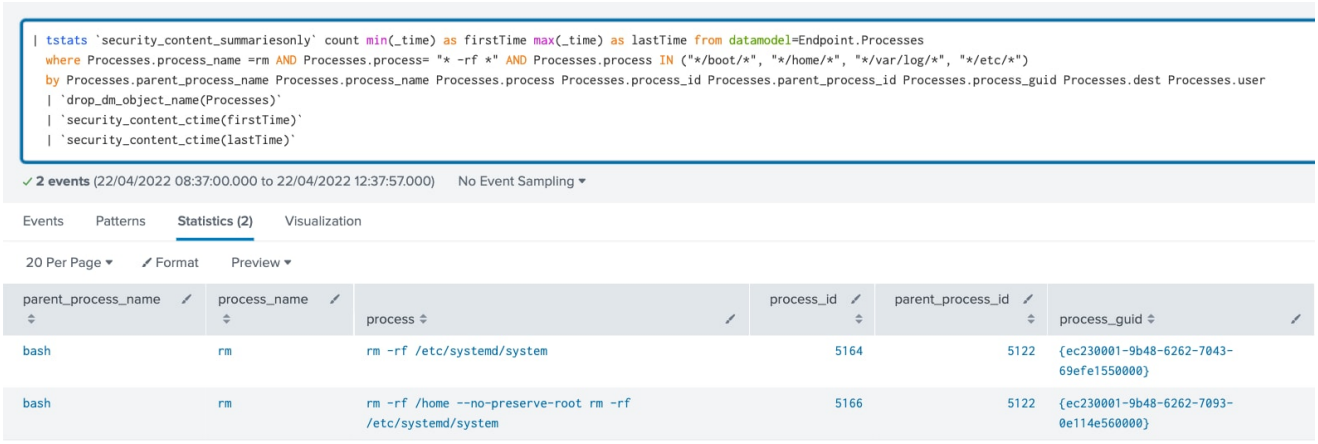
20 Per Page Format Preview

parent_process_name	process_name	process	process_id	parent_process_id	dest
sudo	crontab	crontab -l	2269	2268	sysmonlinux
sudo	crontab	crontab -l	2289	2288	sysmonlinux

Linux Deleting Critical Directory Using RM Command

This analytic identifies a suspicious deletion of a critical folder in Linux machine using rm command. This technique was seen in Industroyer2 campaign to wipe or destroy energy facilities of a targeted sector.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes

  where Processes.process_name =rm AND Processes.process= "* -rf *" AND Processes.process IN
("*/boot/*", "*/var/log/*", "*/etc/*", "*/dev/*")

  by Processes.parent_process_name Processes.process_name Processes.process
Processes.process_id Processes.parent_process_id Processes.process_guid Processes.dest
Processes.user

  | `drop_dm_object_name(Processes)`

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



```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
  where Processes.process_name =rm AND Processes.process= "* -rf *" AND Processes.process IN ("*/boot/*", "*/home/*", "*/var/log/*", "*/etc/*")
  by Processes.parent_process_name Processes.process_name Processes.process Processes.process_id Processes.parent_process_id Processes.process_guid Processes.dest Processes.user
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ 2 events (22/04/2022 08:37:00.000 to 22/04/2022 12:37:57.000) No Event Sampling

Events Patterns Statistics (2) Visualization

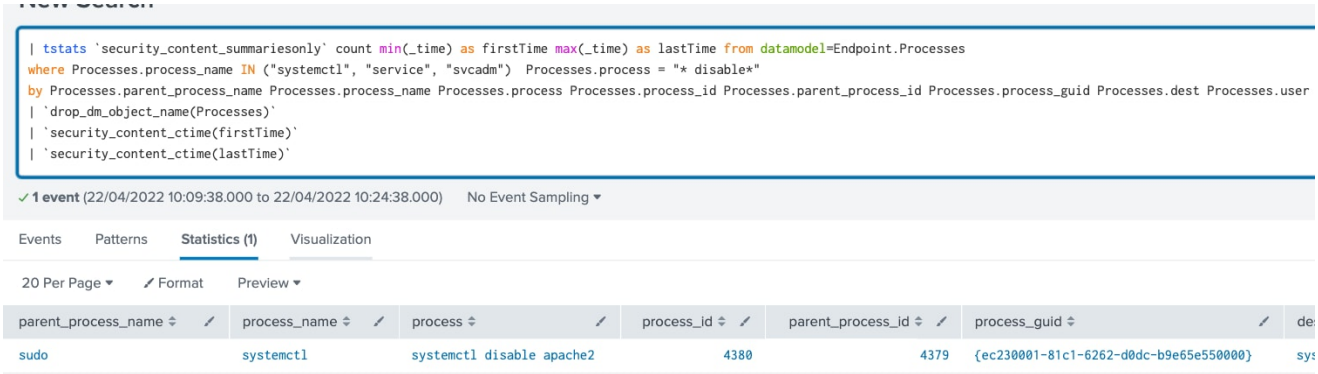
20 Per Page Format Preview

parent_process_name	process_name	process	process_id	parent_process_id	process_guid
bash	rm	rm -rf /etc/systemd/system	5164	5122	{ec230001-9b48-6262-7043-69efe1550000}
bash	rm	rm -rf /home --no-preserve-root rm -rf /etc/systemd/system	5166	5122	{ec230001-9b48-6262-7093-0e114e560000}

Linux Disable Services

This analytic identifies events that attempt to disable a service. This is typically identified in parallel with other instances of service enumeration of attempts to stop a service and then delete it.

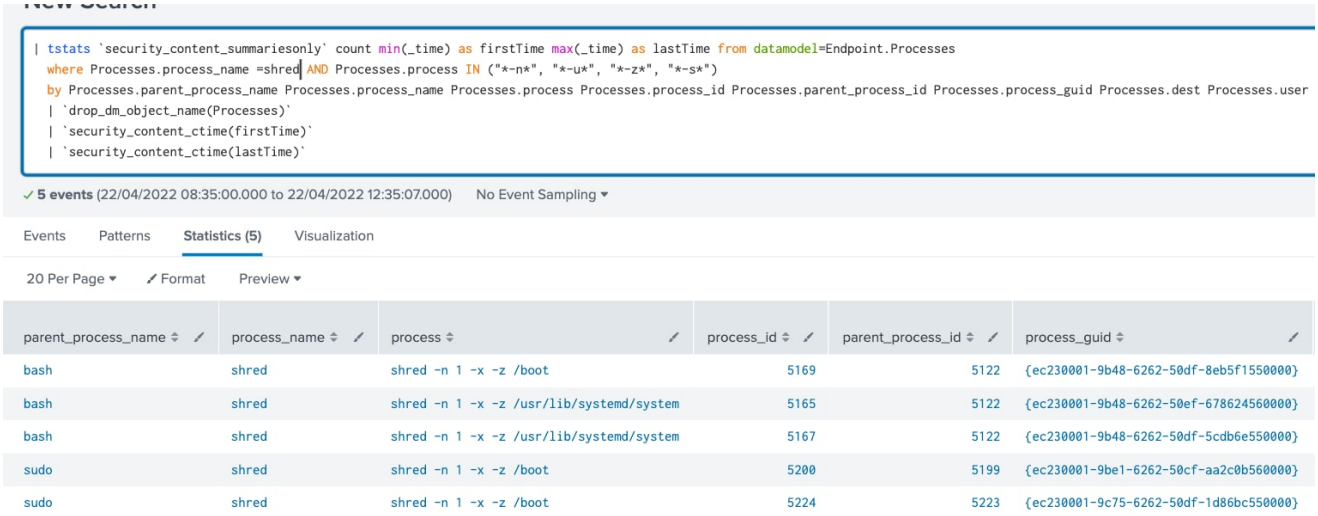
```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes

  where Processes.process_name IN ("systemctl", "service", "svcadm")  Processes.process = "*
disable*"

  by Processes.parent_process_name Processes.process_name Processes.process
Processes.process_id Processes.parent_process_id Processes.process_guid Processes.dest
Processes.user

  | `drop_dm_object_name(Processes)`

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



Linux Shred Overwrite Command

This analytic identifies a shred process to overwrite files in a linux machine. Shred Linux application is designed to overwrite a file to hide its contents or make the deleted file unrecoverable.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes

  where Processes.process_name =shred AND Processes.process IN ("*-n*", "*-u*", "*-z*", "*-s*")

  by Processes.parent_process_name Processes.process_name Processes.process
Processes.process_id Processes.parent_process_id Processes.process_guid Processes.dest
Processes.user

  | `drop_dm_object_name(Processes)`

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



```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
  where Processes.process_name =shred AND Processes.process IN ("*-n*", "*-u*", "*-z*", "*-s*")
  by Processes.parent_process_name Processes.process_name Processes.process Processes.process_id Processes.parent_process_id Processes.process_guid Processes.dest Processes.user
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ **5 events** (22/04/2022 08:35:00.000 to 22/04/2022 12:35:07.000) No Event Sampling

Events Patterns **Statistics (5)** Visualization

20 Per Page Format Preview

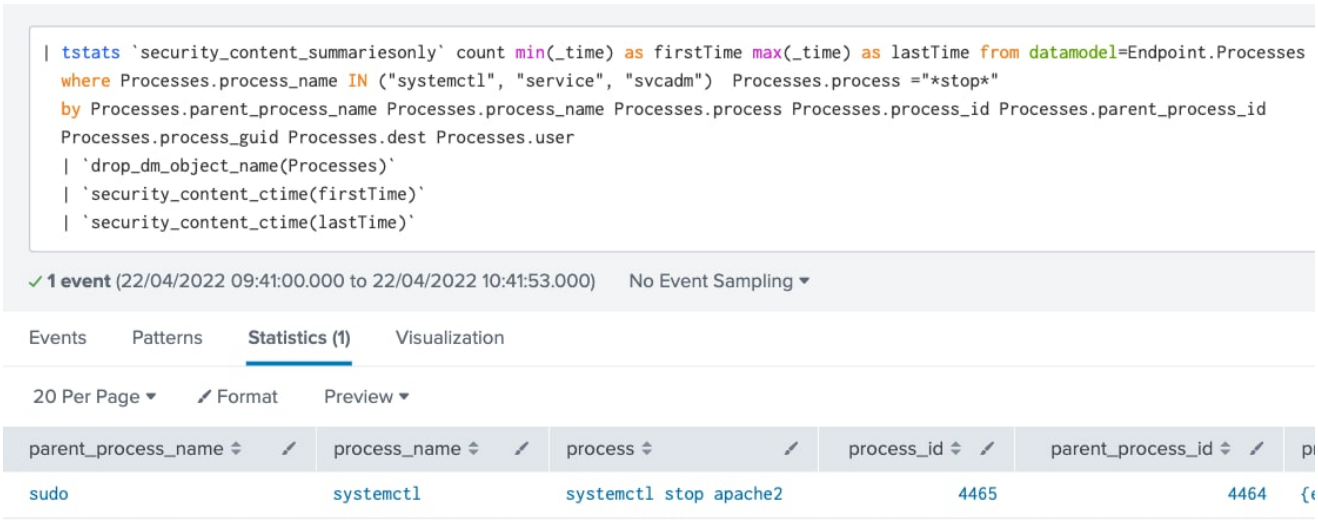
parent_process_name	process_name	process	process_id	parent_process_id	process_guid
bash	shred	shred -n 1 -x -z /boot	5169	5122	{ec230001-9b48-6262-50df-8eb5f1550000}
bash	shred	shred -n 1 -x -z /usr/lib/systemd/system	5165	5122	{ec230001-9b48-6262-50ef-678624560000}
bash	shred	shred -n 1 -x -z /usr/lib/systemd/system	5167	5122	{ec230001-9b48-6262-50df-5cdb6e550000}
sudo	shred	shred -n 1 -x -z /boot	5200	5199	{ec230001-9be1-6262-50cf-aa2c0b560000}
sudo	shred	shred -n 1 -x -z /boot	5224	5223	{ec230001-9c75-6262-50df-1d86bc550000}

Linux Stop Services

This analytic identifies events that attempt to stop or clear a service.

This is typically identified in parallel with other instances of service enumeration of attempts to stop a service and then delete it.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
from datamodel=Endpoint.Processes

  where Processes.process_name IN ("systemctl", "service", "svcadm")  Processes.process
="*stop*"

  by Processes.parent_process_name Processes.process_name Processes.process
Processes.process_id Processes.parent_process_id

  Processes.process_guid Processes.dest Processes.user

  | `drop_dm_object_name(Processes)` | `security_content_ctime(firstTime)`

  | `security_content_ctime(lastTime)`
```



```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes
  where Processes.process_name IN ("systemctl", "service", "svcadm")  Processes.process ="*stop*"
  by Processes.parent_process_name Processes.process_name Processes.process Processes.process_id Processes.parent_process_id
  Processes.process_guid Processes.dest Processes.user
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ **1 event** (22/04/2022 09:41:00.000 to 22/04/2022 10:41:53.000) No Event Sampling

Events Patterns **Statistics (1)** Visualization

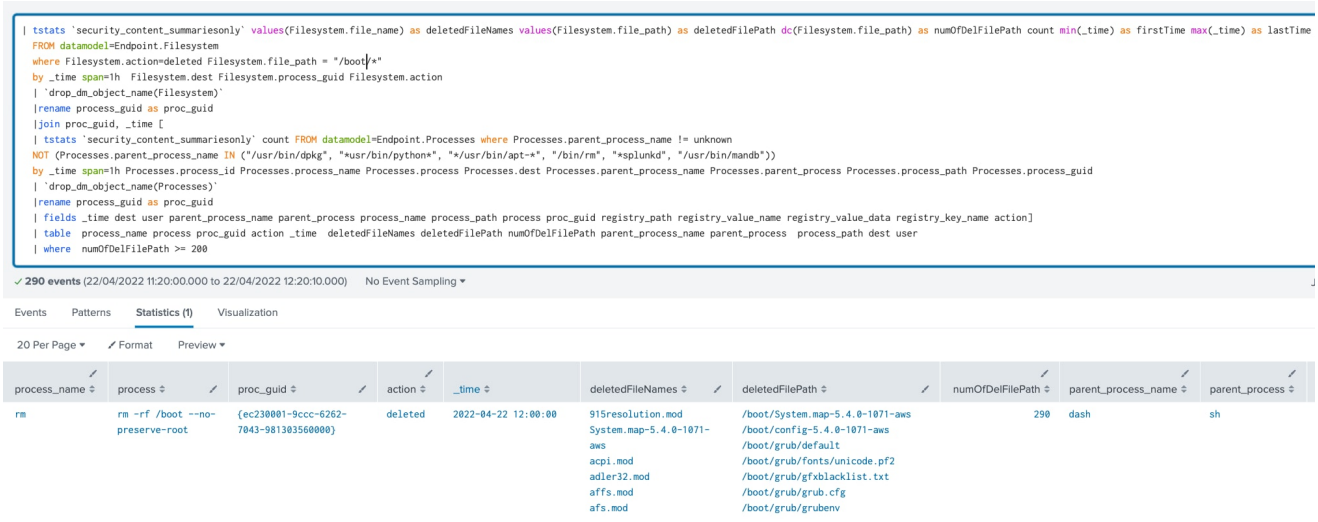
20 Per Page Format Preview

parent_process_name	process_name	process	process_id	parent_process_id	p
sudo	systemctl	systemctl stop apache2	4465	4464	{e

Linux High Frequency Of File Deletion In Boot Folder

This analytic identifies a high frequency of file deletion relative to process name and process id /boot/ folder.

```
| tstats `security_content_summariesonly` values(Filesystem.file_name) as deletedFileNames values(Filesystem.file_path) as deletedFilePath dc(Filesystem.file_path) as numOfDelFilePath count min(_time) as firstTime max(_time) as lastTime

  FROM datamodel=Endpoint.Filesystem

  where Filesystem.action=deleted Filesystem.file_path = "/boot/*"

  by _time span=1h  Filesystem.dest Filesystem.process_guid Filesystem.action

  | `drop_dm_object_name(Filesystem)`

  |rename process_guid as proc_guid

  |join proc_guid, _time [

  | tstats `security_content_summariesonly` count FROM datamodel=Endpoint.Processes where Processes.parent_process_name != unknown

  NOT (Processes.parent_process_name IN ("/usr/bin/dpkg", "*usr/bin/python*", "*/usr/bin/apt-*", "/bin/rm", "*splunkd", "/usr/bin/mandb"))

  by _time span=1h Processes.process_id Processes.process_name Processes.process Processes.dest Processes.parent_process_name Processes.parent_process Processes.process_path Processes.process_guid

  | `drop_dm_object_name(Processes)`

  |rename process_guid as proc_guid

  | fields _time dest user parent_process_name parent_process process_name process_path process proc_guid registry_path registry_value_name registry_value_data registry_key_name action]

  | table  process_name process proc_guid action _time  deletedFileNames deletedFilePath numOfDelFilePath parent_process_name parent_process  process_path dest user

  | where  numOfDelFilePath >= 200
```

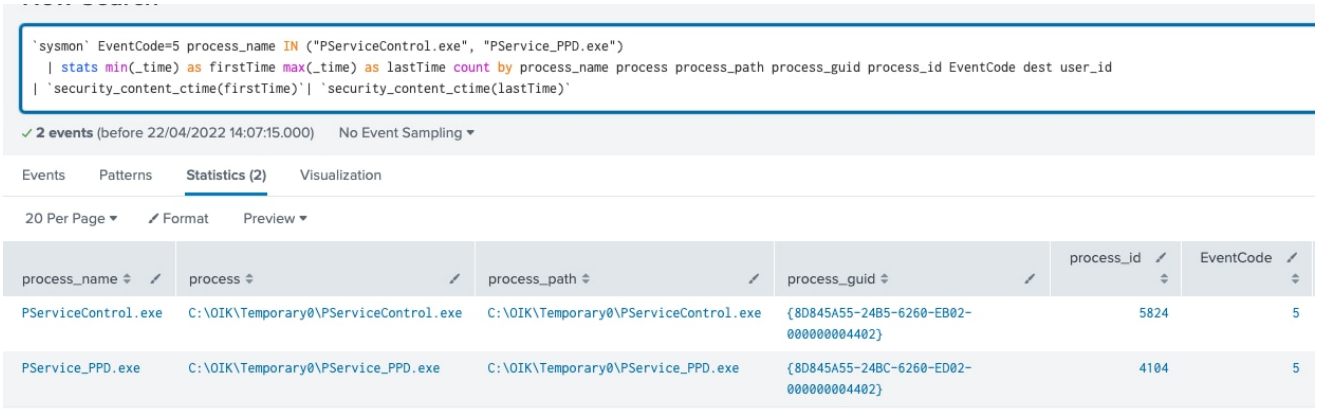


Windows Processes Killed By Industroyer2 Malware

This analytic identifies known processes killed by Industroyer2 malware.

This technique was seen in the Industroyer2 malware attack that tries to kill several processes of windows host machines related to the energy facility network.

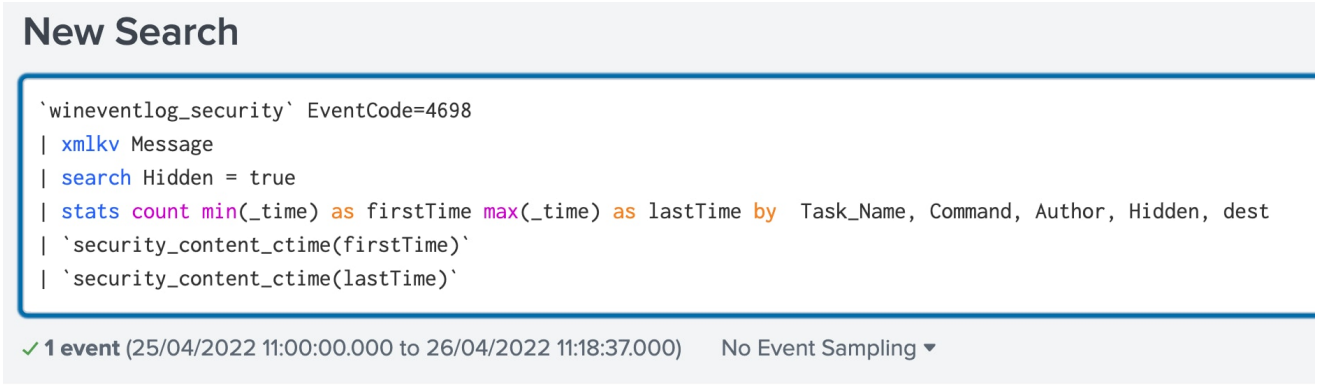
```
`sysmon` EventCode=5 process_name IN ("PServiceControl.exe", "PService_PPD.exe")

  | stats min(_time) as firstTime max(_time) as lastTime count by process_name process
process_path process_guid process_id EventCode dest user_id

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



Windows Hidden Schedule Task Settings

The following query utilizes Windows Security EventCode 4698. A scheduled task was created to identify suspicious tasks registered on Windows either via schtasks.exe OR TaskService with hidden settings that are unique entry of malware like Industroyer2 or attack that uses lolbin to download other files or payload to the infected machine.

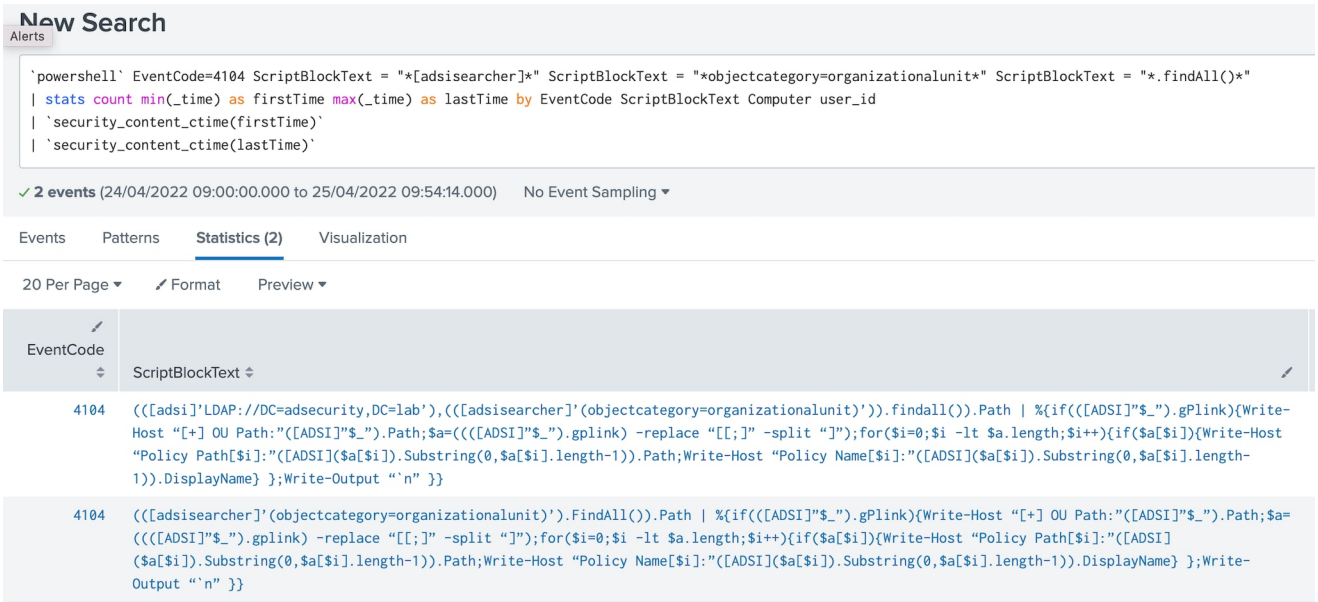
```
`wineventlog_security` EventCode=4698

  | xmlkv Message

  | search Hidden = true

  | stats count min(_time) as firstTime max(_time) as lastTime by  Task_Name, Command, Author,
Hidden, dest

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



Windows Linked Policies In ADSI Discovery

This analytic utilizes PowerShell Script Block Logging (EventCode=4104) to identify the `[Adsisearcher]` type accelerator being used to query Active Directory for domain groups.

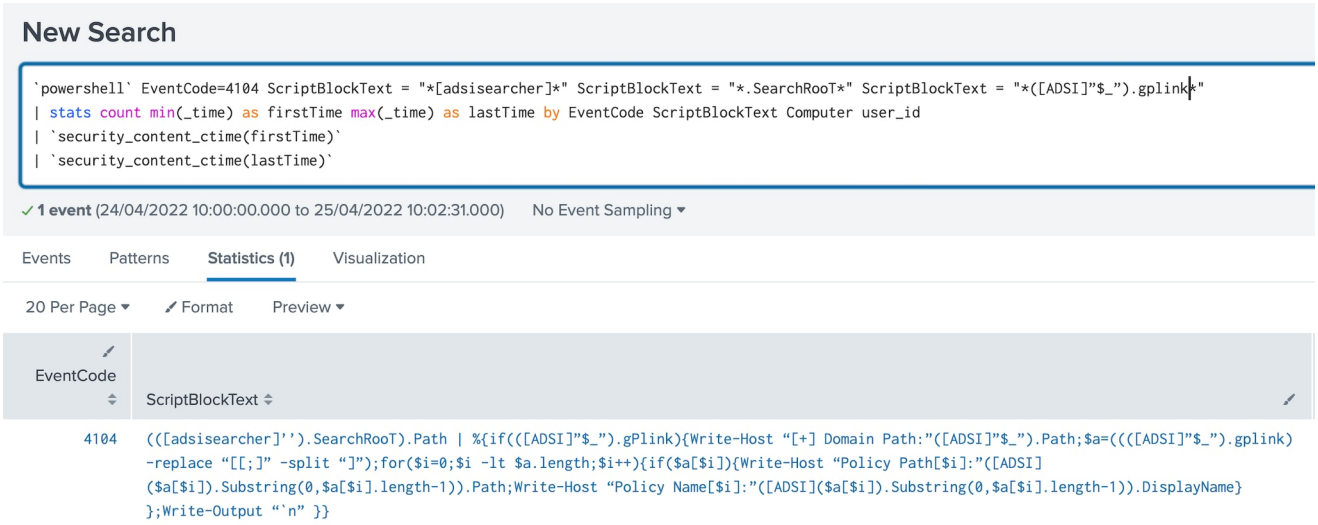
```
`powershell` EventCode=4104 ScriptBlockText = "*[adsisearcher]*" ScriptBlockText =
"*objectcategory=organizationalunit*" ScriptBlockText = "*findAll()*"

  | stats count min(_time) as firstTime max(_time) as lastTime by EventCode ScriptBlockText
Computer user_id

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



Windows Root Domain Linked Policies Discovery

This analytic utilizes PowerShell Script Block Logging (EventCode=4104) to identify the `[Adsisearcher]` type accelerator being used to query Active Directory for domain groups. Red Teams and adversaries may leverage `[Adsisearcher]` to enumerate root domain linked policies for situational awareness and Active Directory Discovery.

```
`powershell` EventCode=4104 ScriptBlockText = "*[adsisearcher]*" ScriptBlockText =
"*.SearchRooT*" ScriptBlockText = "*([ADSI]”$_”).gplink*"

  | stats count min(_time) as firstTime max(_time) as lastTime by EventCode ScriptBlockText
Computer user_id

  | `security_content_ctime(firstTime)`

  | `security_content_ctime(lastTime)`
```



Type	Name	Technique ID	Tactic	Description
TTP	WinEvent Scheduled Task Created Within Public Path (Updated)	T1053.005	Execution, Persistence, Privilege Escalation	The following query utilizes Windows Security EventCode 4698. A scheduled task was created to identify suspicious tasks registered on Windows either via schtasks.exe OR TaskService with a command to be executed from a user-writable file path.
Hunting	WinEvent Windows Task Scheduler Event Action Started	T1053.005	Execution, Persistence, Privilege Escalation	This hunting analytic assists with identifying suspicious tasks that have been registered and run in Windows using EventID 200 (action run) and 201 (action completed).

TTP	Schtasks Run Task On Demand	T1053	Execution, Persistence, Privilege Escalation	This analytic identifies an on-demand run of a Windows Schedule Task through shell or command-line.
TTP	Attempted Credential Dump From Registry via Reg exe	T1003	Credential Access	This analytic identifies the use of reg.exe attempting to export Windows registry keys that contain hashed credentials. Adversaries will utilize this technique to capture and perform offline password cracking.
TTP	Dump LSASS via comsvcs DLL	T1003.001	Credential Access	This analytic identifies the usage of comsvcs.dll for dumping the lsass process.
TTP	Executable File Written in Administrative SMB Share	T1021.002	Lateral Movement	This analytic identifies executable files (.exe or .dll) being written to Windows administrative SMB shares (Admin$, IPC$, C$).
TTP	Suspicious Process File Path	T1543	Persistence, Privilege Escalation	This analytic identifies a suspicious process running in a file path where a process is not commonly seen and is most commonly used by malicious software.
TTP	Executables Or Script Creation In Suspicious Path	T1036	Defense Evasion	This analytic identifies suspicious executables or scripts (known file extensions) in a list of suspicious file paths in Windows.
TTP	Impacket Lateral Movement Commandline Parameters	T1021 T1021.002 T1021.003 T1047 T1543.003	Lateral Movement Execution Persistence, Privilege Escalation	This analytic identifies the presence of suspicious command line parameters typically present when using Impacket tools.
Anomaly	Linux System Network Discovery	T1016	Discovery	This analytic identifies possible enumeration of local network configuration. This technique is commonly used as part of recon of adversaries or threat actors to know some network information for its next or further attack.

TTP	Recon Using WMI Class	T1592	Reconnaissance	This analytic identifies suspicious PowerShell via EventCode 4104, where WMI is performing an event query looking for running processes or running services.
Hunting	Linux Adding Crontab Using List Parameter (New)	T1053.003	Execution, Persistence, Privilege Escalation	This analytic identifies a suspicious cron jobs modification using crontab list parameters.
TTP	Linux Deleting Critical Directory Using RM Command (New)	T1485	Impact	This analytic identifies a suspicious deletion of a critical folder in a Linux machine using rm command.
TTP	Linux Disable Services (New)	T1489	Impact	This analytic identifies events that attempt to disable a service.
TTP	Linux Shred Overwrite Command (New)	T1485	Impact	This analytic identifies a shred process to overwrite files in a Linux machine.
TTP	Linux Stop Services (New)	T1489	Impact	This analytic identifies events that attempt to stop or clear a service.
Anomaly	Windows Processes Killed By Industroyer2 Malware	T1489	Impact	This analytic identifies known processes killed by Industroyer2 malware.
TTP	Windows Hidden Schedule Task Settings (New)	T1053	Execution, Persistence, Privilege Escalation	This query utilizes Windows Security EventCode 4698. A scheduled task was created to identify suspicious tasks registered on Windows either via schtasks.exe OR TaskService with a hidden setting.
Anomaly	Windows Linked Policies In ADSI Discovery	T1087.002	Discovery	This analytic utilizes PowerShell Script Block Logging (EventCode=4104) to identify the `[Adsisearcher]` type accelerator being used to query Active Directory for domain groups.

Anomaly	Windows Root Domain linked policies Discovery	T1087.002	Discovery	This analytic utilizes PowerShell Script Block Logging (EventCode=4104) to identify the `[Adsisearcher]` type to enumerate root domain linked policies for situational awareness and Active Directory Discovery.

** To see a detailed explanation on the different types please refer to this wiki.*

IOC:

Filename	Size	Sha256
industroyer2.exe	37.00 KB (37888 bytes)	d69665f56ddef7ad4e71971f06432e59f1510a7194386e5f0e8926aea7b88e0

Mitigation

Please follow CISA and NSA Joint advisory on securing Operational Technology (OT).

Learn More

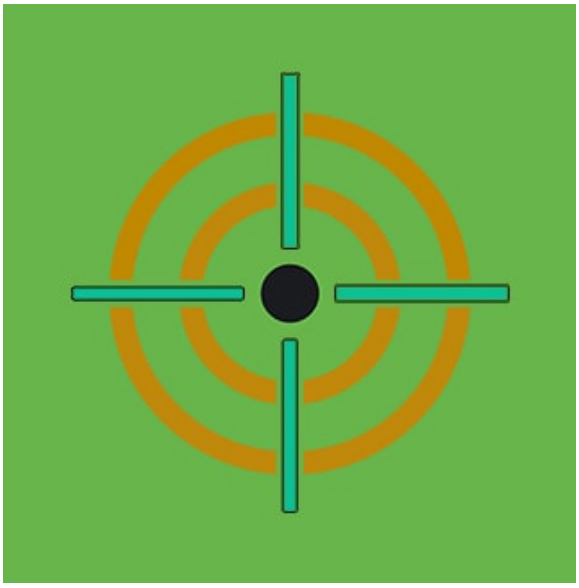
You can find the latest content about security analytic stories on GitHub and in Splunkbase. Splunk Security Essentials also has these detections available via push update. In the upcoming weeks, the Splunk Threat Research Team will be releasing a more detailed blog post on this analytic story. Stay tuned!

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.

We would like to thank the following for their contributions to this post: Teoderick Contreras, Rod Soto, Jose Hernandez, Patrick Barreiss, Lou Stella, Mauricio Velazco, Michael Haag, Bhavin Patel, and Eric McGinnis



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.