

Weapon System Training - 2

LAB 1: CONFIGURING SECURITYONION AND WINLOGBEATS




Maj Michael Lester and Capt Jon Bynum
32 WPS/DOA | NELLIS AFB, NEVADA

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

CONTENTS

| | |
|--|----|
| Lab 2: Threat Hunting with Event Id 4688 | 2 |
| Overview | 2 |
| Procedures | 3 |
| Step 1. Allow Winlogbeat Traffic to SecurityOnion | 3 |
| Step 2. Download and Configure Winlogbeats Agent | 3 |
| Step 3. Deploy Winlogbeats via GPO | 6 |
| Appendix | 15 |
| Appendix A: Javascript Processor Examples | 15 |
| Javascript Processor Reference Links | 15 |
| References an External Javascript file in the YML Configuration File | 15 |
| Converting Field Types | 15 |
| Creating Generic Fields with Lookups | 16 |
| Extracting Data with Regular Expressions | 17 |
| Combining Processor Functions in a Chain | 18 |

Symbols Table

| Symbol | Name | Meaning |
|---|-----------------------|---|
|  | Note | Detailed information that is required to fully understanding the concept or to be able to execute a procedure but is not necessarily related to a key learning objective. |
|  | Learning Point | Information related to key learning objectives. |
|  | Warning | Important information related to safety and security. |

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

LAB 2: THREAT HUNTING WITH EVENT ID 4688

OVERVIEW

Summary: The purpose of this lab is to acquire the necessary knowledge and skills to effectively configure and deploy SecurityOnion and Winglog Beats to detect and investigate malicious cyber activity.

Outcomes: By the end of the lab, you will be able to perform the following:

- Log into SecurityOnion.
- Configure the SecurityOnion firewall.
- Configure and deploy Winlogbeats via GPO.
 - Configure data augmentation ECMA script.
- Verify that Windows event logs are being ingested.
- Verify the index mapping is configured as desired.

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

PROCEDURES

STEP 1. ALLOW WINLOGBEAT TRAFFICE TO SECURITYONION

1. By default, the SecurityOnion firewall is configured to block most traffic. Exemptions must be made for each service with an explicit set of IP addresses that are allowed to communicate to that service. In order to allow our workstations to push logs into Logstash, we need to allow Winglog Beats traffic to TCP port 5044.
 - 1.1. Login to the SecurityOnion server via SSH using the following credentials:
 - 1.2. **Username:** soadmin
 - 1.3. **Password:** 1qaz!QAZ
2. Run the following commands to allow Winlog Beat traffic from the defended network to SecurityOnion's Logstash instance.

```
[soadmin@securityonion ~]$ sudo so-allow
[sudo] password for soadmin:

Choose the role for the IP or Range you would like to allow

[a] - Analyst - 80/tcp, 443/tcp
[b] - Logstash Beat - 5044/tcp
[e] - Elasticsearch REST API - 9200/tcp
[f] - Strelka frontend - 57314/tcp
[o] - Osquery endpoint - 8090/tcp
[s] - Syslog device - 514/tcp/udp
[w] - Wazuh agent - 1514/tcp/udp
[p] - Wazuh API - 55000/tcp
[r] - Wazuh registration service - 1515/tcp

Please enter your selection: b
Enter a single ip address or range to allow (ex: 10.10.10.10 or 10.10.0.0/16): 192.168.1.0/24
Adding 192.168.1.0/24 to the beats_endpoint role. This can take a few seconds...
[soadmin@securityonion ~]$
```

STEP 2. DOWNLOAD AND CONFIGURE WINLOGBEATS AGENT

The next step is to configure and deploy the Winlogbeats agent. This agent can be deployed and installed as a service executable on Windows systems with a YAML configuration file that specifies which logs you want to be sent to Logstash or Elastisearch.

Open and review the provided configuration file: winlogbeat.yml. We have carefully chosen a small subset of Windows logs and event ids to collect in order to maximize our detection capabilities while minimizing load on the SIEM.

```
##### winlogbeat Configuration Example #####

# This file is an example configuration file highlighting only the most common
```

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

```
# options. The winlogbeat.reference.yml file from the same directory contains
# all the supported options with more comments. You can use it as a reference.
#
# You can find the full configuration reference here:
# https://www.elastic.co/guide/en/beats/winlogbeat/index.html

# ===== winlogbeat specific options =====

# event_logs specifies a list of event logs to monitor as well as any
# accompanying options. The YAML data type of event_logs is a list of
# dictionaries.
#
# The supported keys are name, id, xml_query, tags, fields, fields_under_root,
# forwarded, ignore_older, level, event_id, provider, and include_xml.
# The xml_query key requires an id and must not be used with the name,
# ignore_older, level, event_id, or provider keys. Please visit the
# documentation for the complete details of each option.
# https://go.es.io/winlogbeatConfig

winlogbeat.event_logs:
  # Event Ids collected can be filtered down using the event_id keyword
  - name: Security
    event_id: 4625, 4688, 4697, 4698, 4720, 4624
    processors:
      - script:
          lang: javascript
          file: C:/ProgramData/Elastic/Beats/winlogbeat/winlogbeat-security-custom.js

  - name: Microsoft-Windows-Sysmon/Operational

  - name: windows PowerShell
    event_id: 400, 800


  - name: Microsoft-Windows-PowerShell/Operational
    event_id: 4103, 4104

# ----- Logstash Output -----
output.logstash:
  # The Logstash hosts
  hosts: ["192.168.1.3:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"
```

 **LEARNING POINT:** output.logstash is important for SecurityOnion to be able to execute the pre-processors that are configured in Logstash. This enables the use of data normalization and pre-built dashboards provided by SecurityOnion. Additionally, leveraging Logstash gives analyst the opportunity to

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

create data enrichment pipelines using Logstash in the future, which you lose by going straight to Elasticsearch.

LEARNING POINT: Data augmentation can also be performed on the endpoint. This has the advantage of distributing the processing out as opposed to Logstash pipelines which centralize the processing. In the configuration above, we configure a javascript (ECMA 5.1) script to run on each event collected by Winlogbeats. This script will be loaded into memory once when the Winlogbeat service is started.

The data augmentation script is a modified version of the one provided out-of-the-box by Elastic in order to enable some of the dashboards, queries, and analytics we will configure later. The snippet below shows an example function that creates additional fields for the Event Id 4688 – Process Creation event.

```
var event4688 = new processor.Chain()
  .Add(copySubjectUser)
  .Add(copySubjectUserLogonId)
  .Add(renameNewProcessFields)
  .Add(copyTargetUserToEffective)
  .Add(addEventFields)
  .Add(function(evt) {
    var cmdline = evt.Get("process.child.command_line");
    var pid = evt.Get("process.child.pid");
    evt.Put("winlog.generic_message", "(" + pid + ") " + cmdline);
  })
  .Build();
```

LEARNING POINT: The script above runs several functions that add, drop, or manipulate Event Id 4688 fields before being shipped to Logstash. The last function adds a new field called “winlog.generic_message” with the PID and commandline of the child process. This field will be used for a contextual investigation dashboard later on where a single table will contain the “winlog.generic_message” field for multiple types of events, allowing the operator to see generic information about each event in a single column without having to do any complex aggregations.

The following command can be used to validate the configuration file and any JavaScript processors referenced in the YML files are syntactically correct. It does **NOT** validate that the configuration file will do what you want it to though. The only way to validate the full functionality is to run winlogbeat with the provided YML file and verify the data you want is ingested.

```
winlogbeat.exe test config -c 'C:\Users\administrator\Desktop\winlogbeat.yml' -e
```

You should get the following response if your YAML file was syntactically correct.

```
Config OK
```

In order to properly deploy Winlogbeats, we need to understand the default paths.

| Path | Description |
|--|---|
| C:\Program Files\Elastic\Beats\7.14.4\ | The default Winlogbeats installation directory. This is where all of the executables are stored. This is not an ideal location for storing configuration files because the version number keeps changing. A |

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

| | |
|--|--|
| | static path that does not change would be preferable for pushing configuration files so that they configuration files can be updated without having to account for the version of winlogbeats installed on the system. |
| C:\ProgramData\Elastic\Beats\winlogbeat\ | The default location for configuration files. |
| C:\ProgramData\Elastic\Beats\winlogbeat\winlogbeat.yml | The default configuration file. |

The key location for our use is the default configuration file location. That is the exact location we need to distribute our configuration file via GPO.

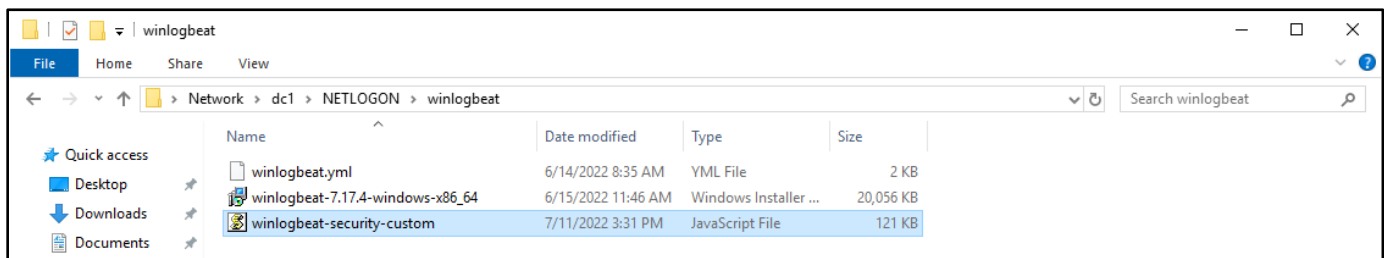
STEP 3. DEPLOY WINLOGBEATS VIA GPO

To deploy Winlogbeats via GPO, we are going to:

1. Share the installer via the netlogon share.
2. Create a GPO to run the Winlogbeats installer.
3. Distribute the configuration YML and JS.
4. Configure and run the winlogbeat service.

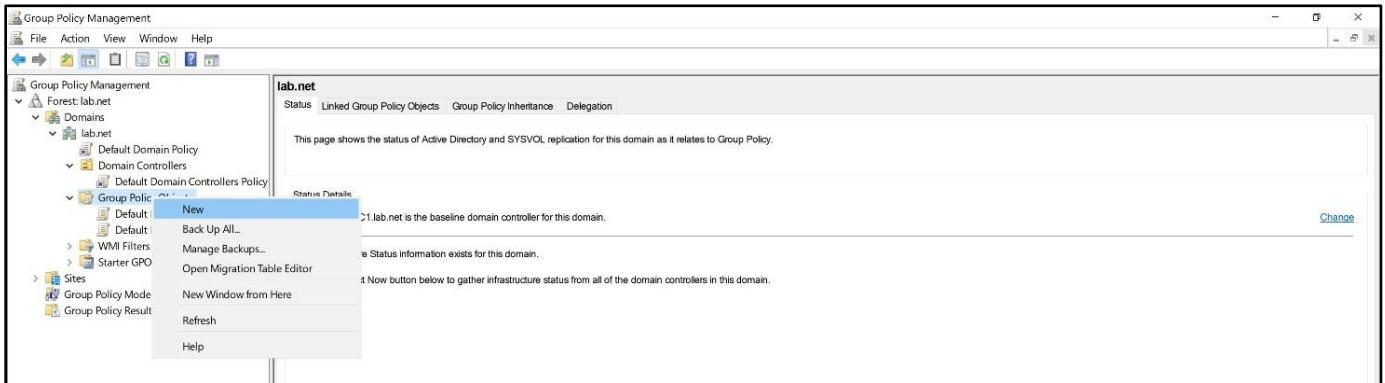
For this deployment, we are going to be using the Winlogbeats installer provided by the current distribution of SecurityOnion. We use this installer to ensure compatibility with SecurityOnion. The files can all be found on the desktop of the primary domain controller.

1. The first step is to place both the winlogbeat installer, winlogbeat-security-custom.js, and the winlogbeat.yml configuration file in a shared location that is accessible by all systems on the domain. For this lab, we are going to use the NETLOGON share which should be accessed using the domain UNC path (\\lab.net\NETLOGON\). By specifying the domain in the UNC path, we ensure that workstations in the environment use their respective domain controllers to pull down a copy of the file, thus load balancing the deployment of the software across all domain controllers in the environment.

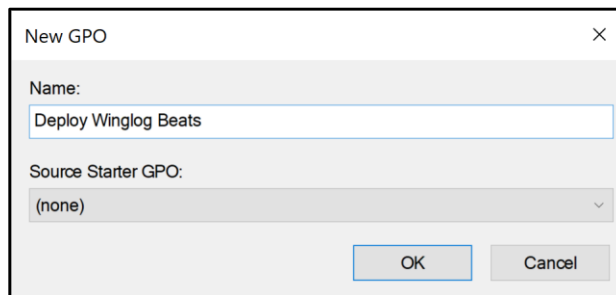


2. Next, open up the Group Policy Management console by using the Windows search feature. Right-click on the Group Policy Objects folder underneath the lab.net domain and then click the “New” menu item on the context menu that pops up.

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats



- This is going to be a very granular GPO that deploys Winlogbeats and nothing else. This GPO can then be assigned to whichever computers that need Winlogbeats. We will give it an easy to remember and understand name. Do not clone from a starter GPO to ensure we start from a clean GPO. Next, click the “OK” button.



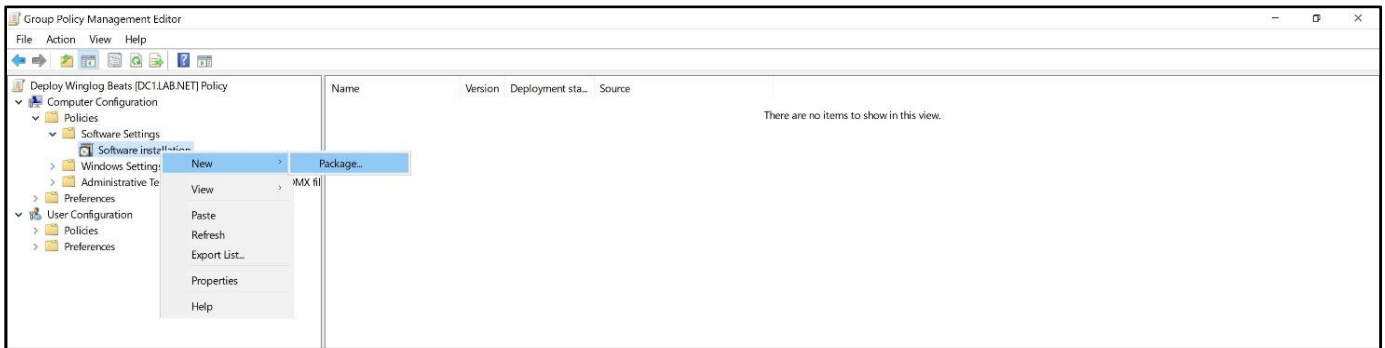
- Now that the GPO has been created, we will open up the GPO Editor and configure the GPO to deploy Winlogbeats. Right-click the GPO and then click the “Edit” button on the context menu that pops up.



- In the GPO Editor, expand the tree to show the Software Installation settings.
5.1. Computer Configuration -> Policies -> Software Settings -> Software Installation

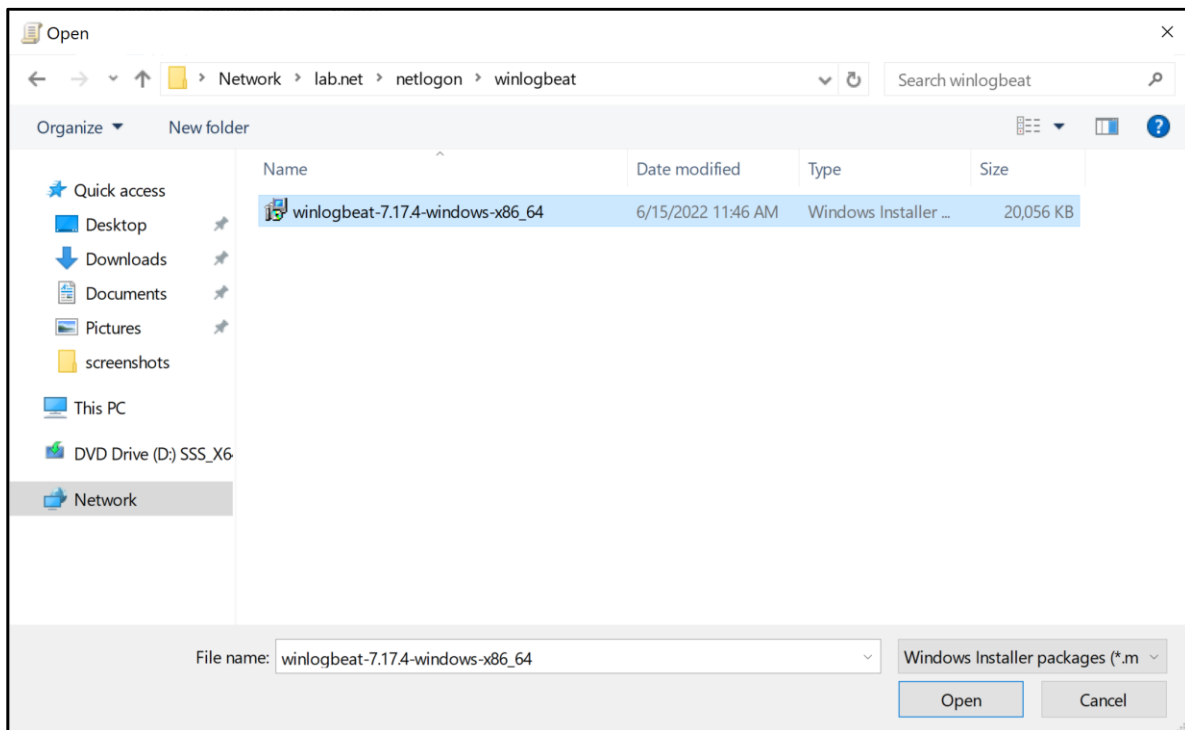
Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

6. Right-click on “Software Installation” then select New -> Package from the context menu.



7. Navigate to the Winlogbeats installer located in the NETLOGON folder and then click “OPEN”.

⚠ WARNING! The GPO is smart enough to select the appropriate default installation options such as /quiet; however, any advanced configuration options would need to use ORCA to configure the MSI or use something more advanced such as System Center Configuration Manager (SCCM).

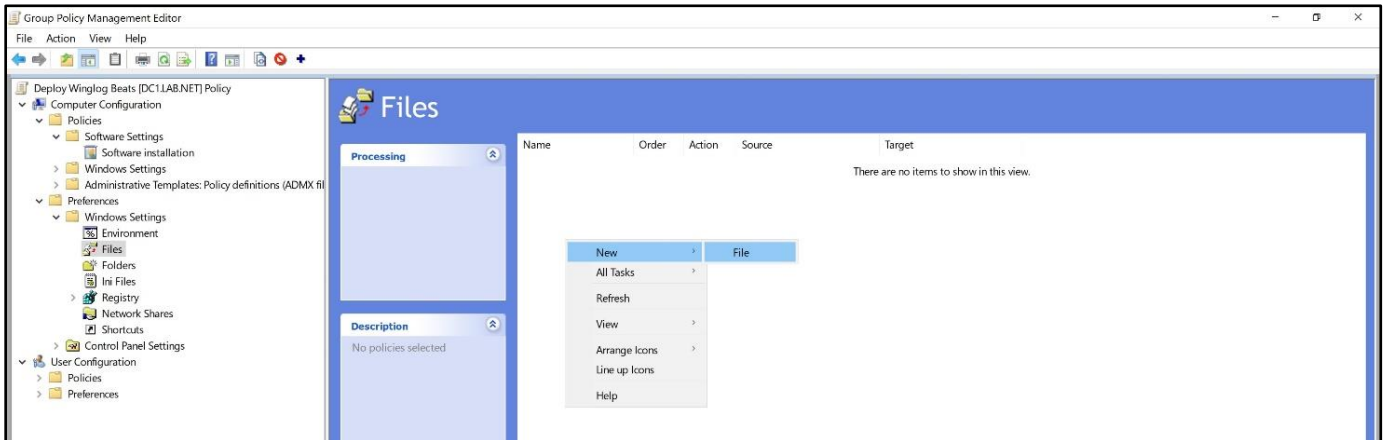


8. Next, we need to configure the GPO to pull down the winlogbeats.yml configuration file and place it in the C:\ProgramData\Elastic\Beats\winlogbeat\ folder. This can be configured by expanding the tree to find the File configuration option.

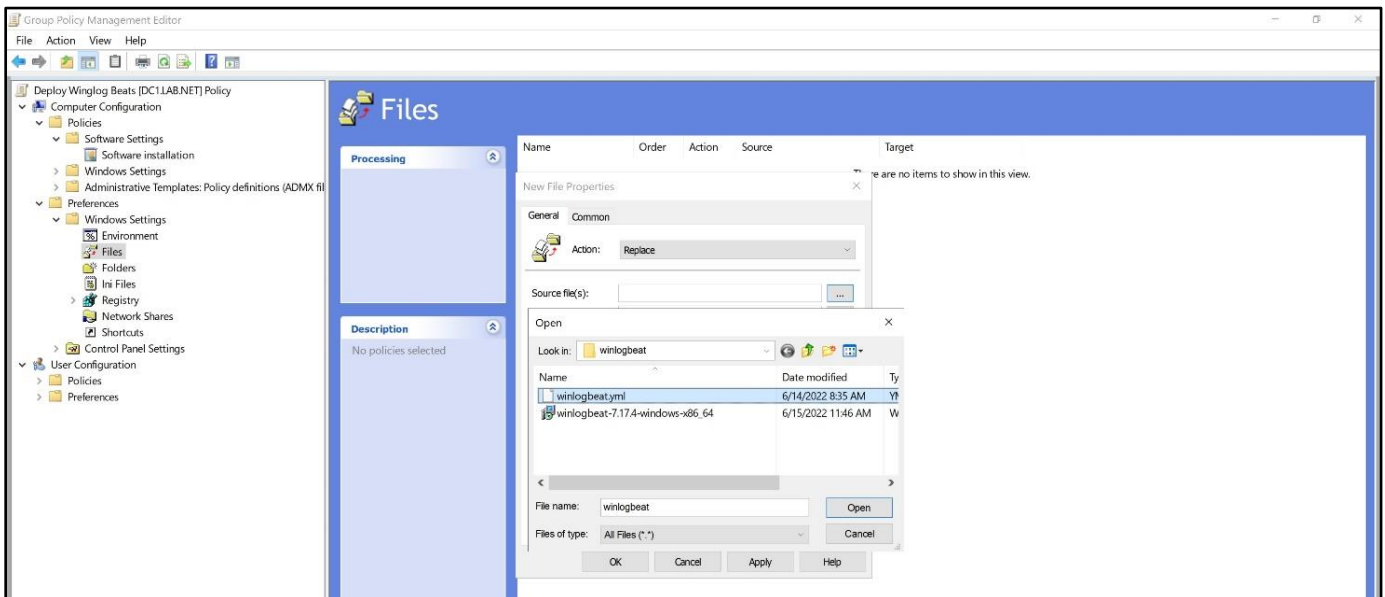
8.1. Computer Configuration -> Preferences -> Windows Settings -> Files

9. Right-click in the empty space and click New -> File from the context menu that pops up.

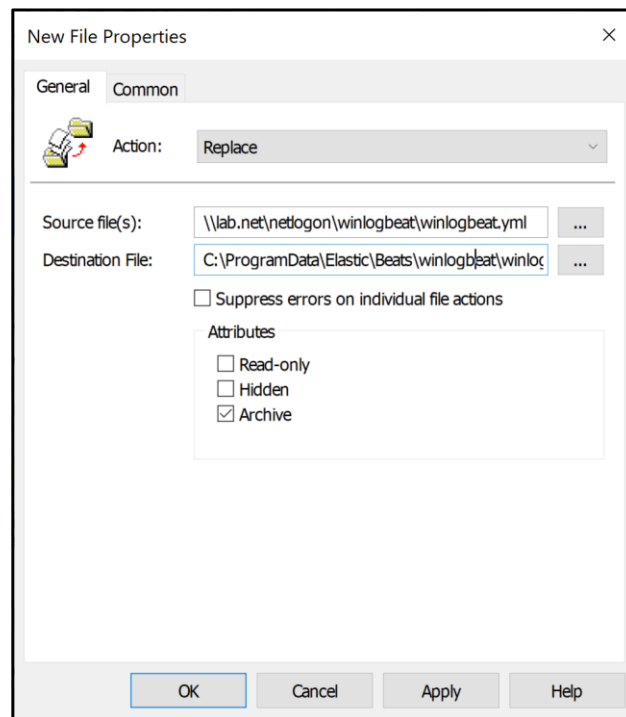
Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats



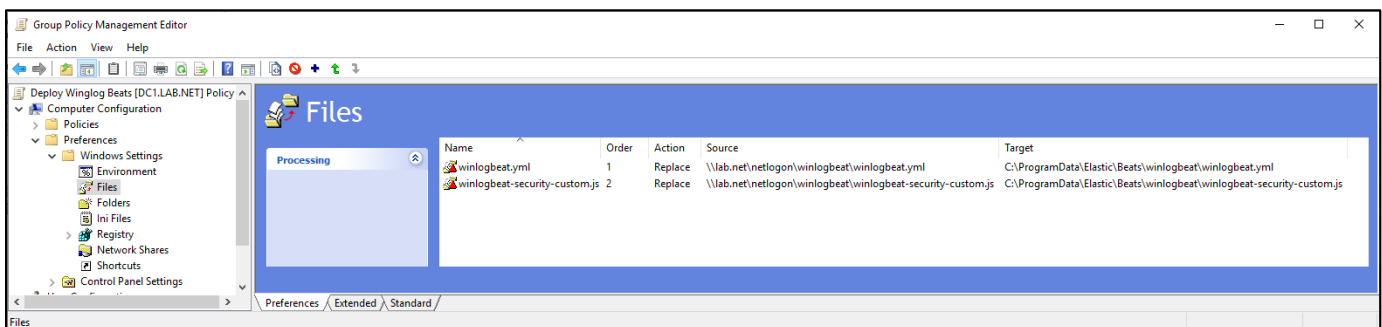
10. For “Source”, select the path to the winlogbeat.yml file in the NETLOGON directory. Ensure that the path is the full UNC path and not a local path. Set the “Destination” path to C:\ProgramData\Elastic\Beats\winlogbeat\winlogbeat.yml. Set the action to “Replace”, then click the “Apply” button.



Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats



11. Repeat steps 9 and 10 for the winlogbeat-security-custom.js script. The file deployment component of the GPO should look like the screen below when finished.

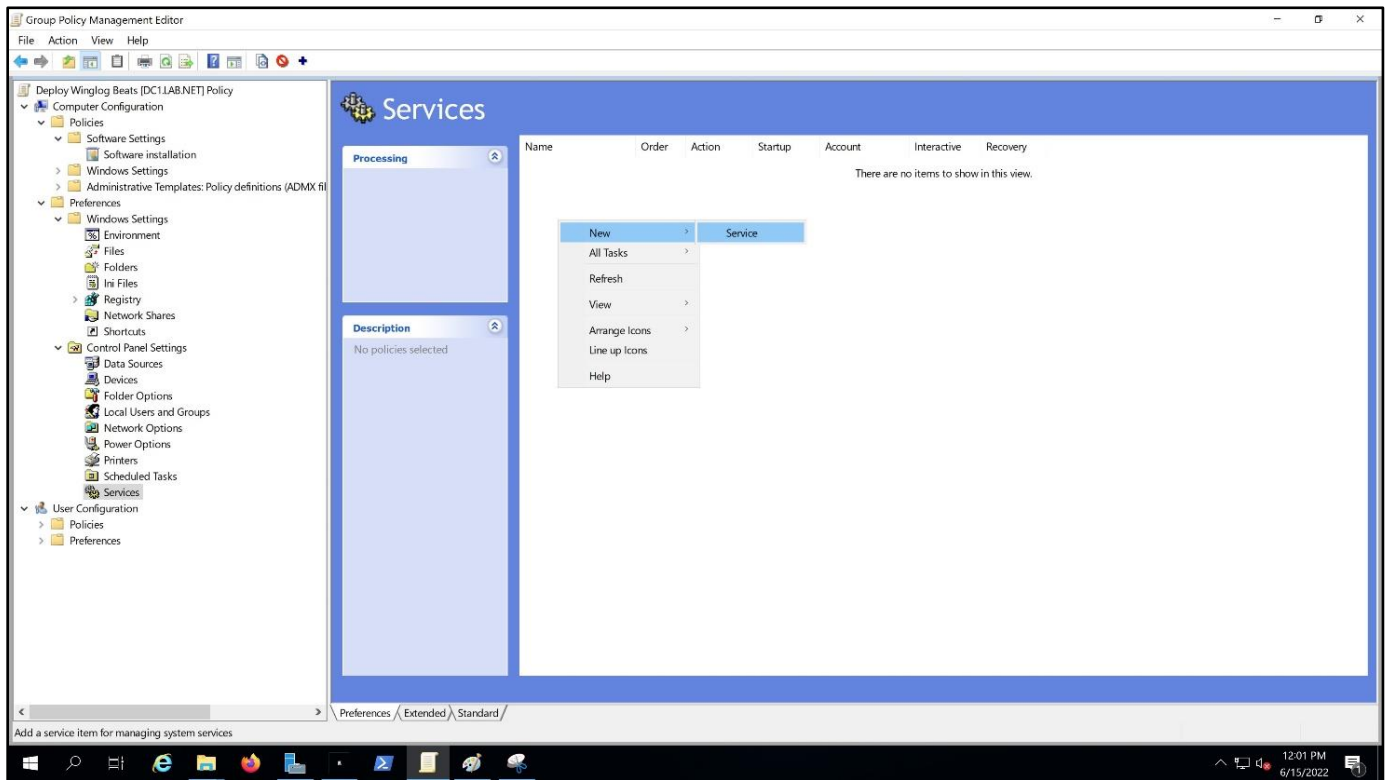


12. Next, we need to configure a policy to automatically start the winlogbeat service and to run it for the first time. By default, the installer will create the winlogbeat service, but it will not configure it to run automatically, nor will it start the service. To do that, we will use the Service settings found by navigating to the following location in the tree view:

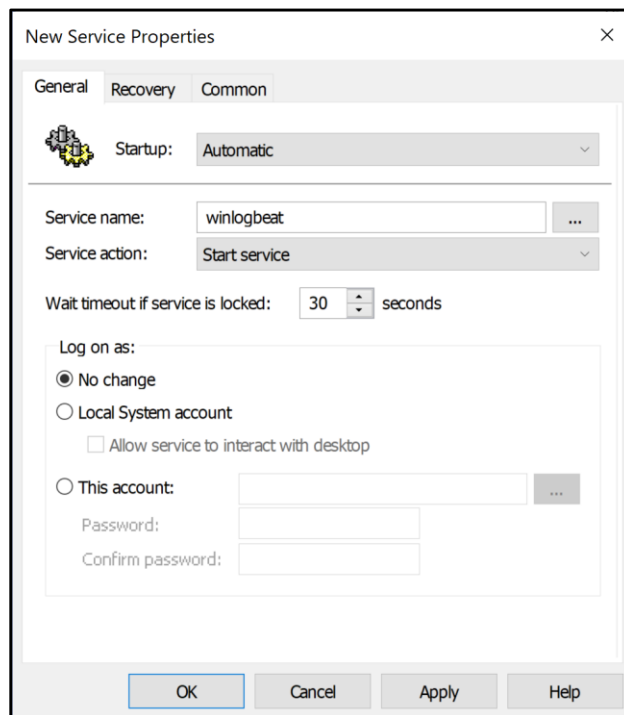
12.1. Computer Configuration -> Preferences -> Control Panel Settings -> Services

13. Right-click in the empty space and select New -> Service from the context menu.

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

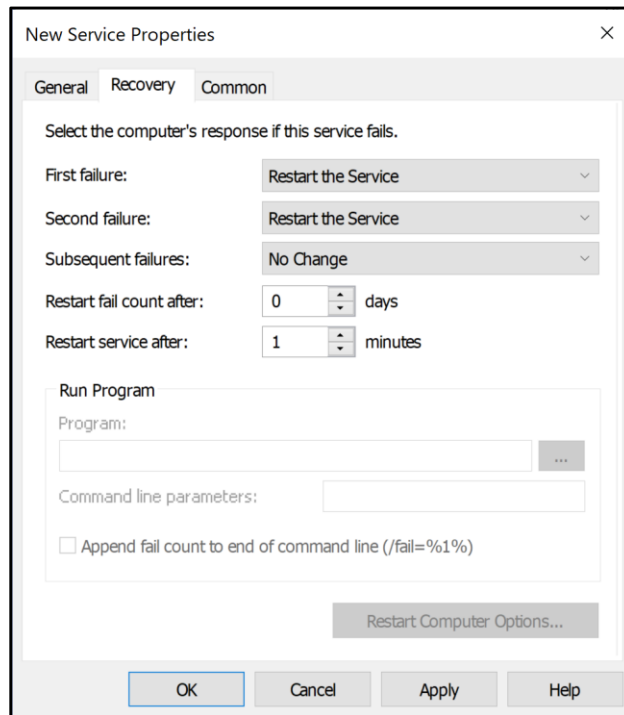


14. Change the “Startup” setting to “Automatic”. Type the service name “winlogbeat” in the “Service Name” text box. Set the “Service Action” to “Start service”.



Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

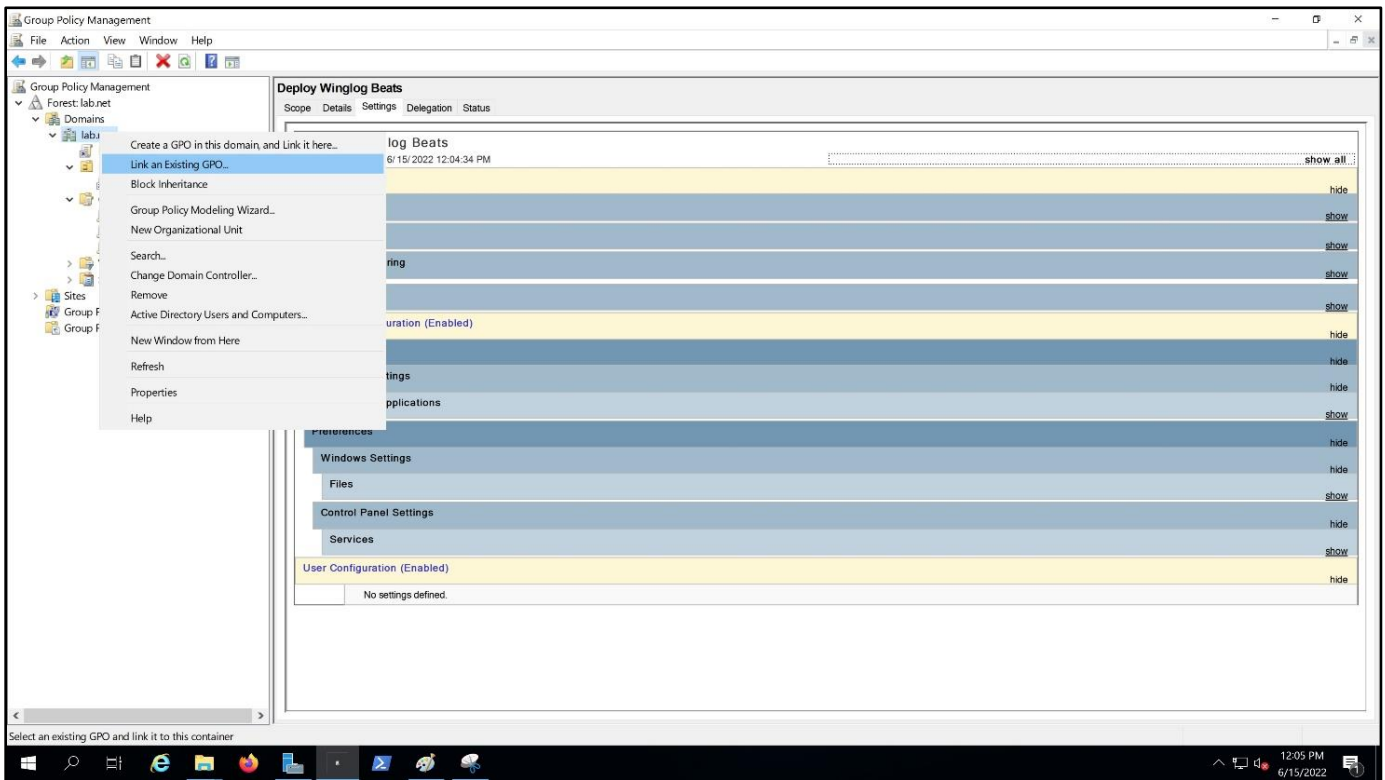
15. Navigate to the “Recovery” tab and configure the first and second failures to “Restart the Service”. Then click the “Apply” button.



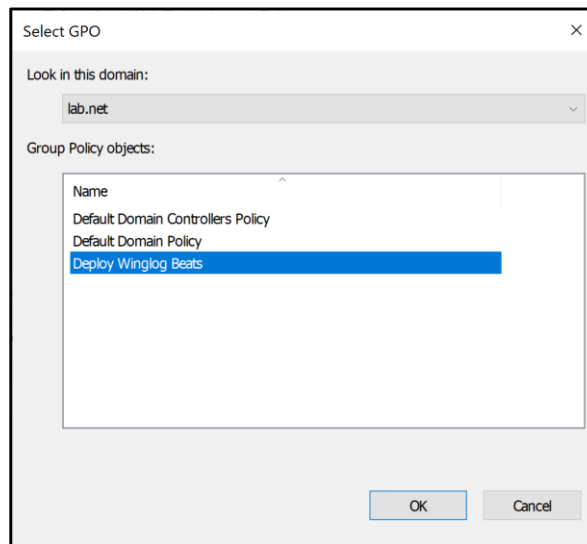
The screenshot shows the 'New Service Properties' dialog box with the 'Recovery' tab selected. The 'General' tab is also visible. The 'Common' tab is selected, and the 'Recovery' section is expanded. The 'First failure:' dropdown is set to 'Restart the Service'. The 'Second failure:' dropdown is also set to 'Restart the Service'. The 'Subsequent failures:' dropdown is set to 'No Change'. The 'Restart fail count after:' is set to '0' days. The 'Restart service after:' is set to '1' minutes. The 'Run Program' section is expanded, showing a 'Program:' field and a 'Command line parameters:' field. There is a checkbox for 'Append fail count to end of command line (/fail=%1%)'. At the bottom, there are buttons for 'OK', 'Cancel', 'Apply', and 'Help'. A 'Restart Computer Options...' button is also visible.

16. The GPO has now been configured! Now we need to link the GPO to an OU in order to deploy winlogbeats. Close out of the GPO Editor and head back to the GPO Management Console. Right-click on the “lab.net” domain folder in the tree view then click “Link and Existing GPO” from the context menu.

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

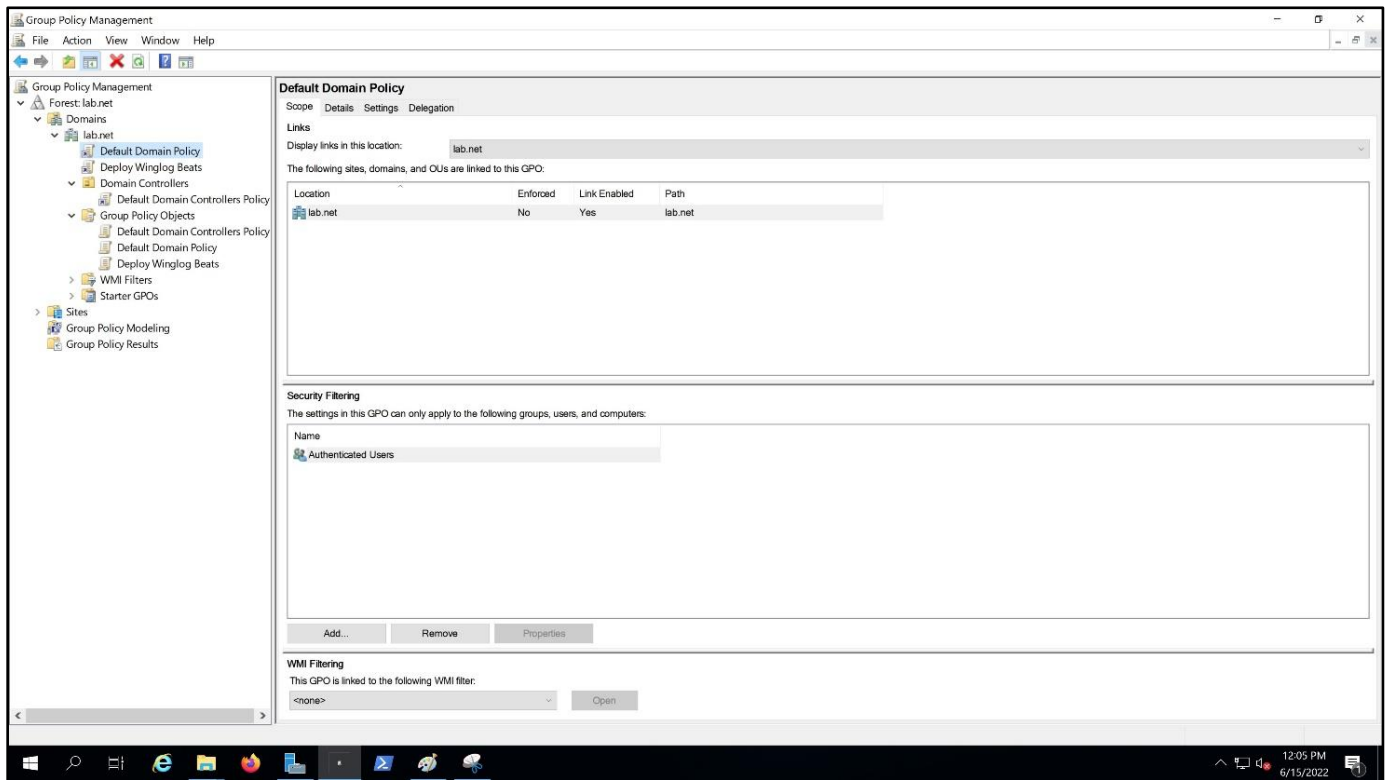


17. Select the “Deploy Winlogbeats” GPO from the menu and then click “OK”.



18. Now that this GPO has been deployed, all of the systems in the environment will install the Winlogbeats package on the next restart.

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats



STEP 4. CONFIGURE AUDIT POLICY GPO

<insert instructions to configure an audit policy>

STEP 5. CONFIGURE ADVANCED KIBANA SETTINGS

<insert instructions to configure the following settings>

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

APPENDIX

APPENDIX A: JAVASCRIPT PROCESSOR EXAMPLES

JAVASCRIPT PROCESSOR REFERENCE LINKS

The Winlogbeat processor reference can be found at the link below.


<https://www.elastic.co/guide/en/beats/winlogbeat/master/processor-script.html>

The script processor executes Javascript code to process an event. The processor uses a pure Go implementation of ECMAScript 5.1 and has no external dependencies. Essentially, the script is loaded when the Winlogbeat service executable, winlogbeat.exe, starts up. As the service processes events, it passes them to the provided processor script, which is already loaded into memory. This makes the process more efficient than loading the script from disk for each event.

REFERENCES AN EXTERNAL JAVASCRIPT FILE IN THE YML CONFIGURATION FILE

Use the following syntax to reference an external JavaScript file as a script processor.

```
winlogbeat.event_logs:
- name: security
  event_id: 4625, 4688, 4697, 4698, 4720, 4624
  processors:
  - script:
    lang: javascript
    file: C:/ProgramData/Elastic/Beats/winlogbeat/winlogbeat-security-custom.js
```

 **NOTE:** The winlogbeat-security-custom.js script will still need to be distributed to each computer since it is referenced locally.

CONVERTING FIELD TYPES

Description: This code snippet below defines a function, stored as a variable called processIdConverter, that converts one field type to another.

Convert Function Arguments:

| Name | Required | Description |
|----------------|----------|---|
| fields | Yes | This is the list of fields to convert. At least one item must be contained in the list. Each item in the list must have a from key that specifies the source field. The to key is optional and specifies where to assign the converted value. If to is omitted then the from field is updated in-place. The type key specifies the data type to convert the value to. If type is omitted then the processor copies or renames the field without any type conversion. The supported types include: integer, long, float, double, string, boolean, and ip. |
| ignore_missing | No | If true the processor continues to the next field when the from key is not found in the event. If false then the processor returns an error and does not process the remaining fields. Default is false. |
| fail_on_error | No | If false type conversion failures are ignored and the processor continues to the next field. Default is true. |

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

| | | | |
|--------------|---|----|---|
| | mode | No | When both from and to are defined for a field then mode controls whether to copy or rename the field when the type conversion is successful. Default is copy. |
| | <p>💡 LEARNING POINT: The “fail_on_error” property is useful for developing, testing, and debugging your javascript file before pushing to production. In production, it may be more desirable to ignore errors and continue with whatever fields are available, versus dropping the event entirely.</p> <p>✅ NOTE: Processor Chains are objects that can contain multiple processor functions. These objects need to be “built”, so the last step of the chain must be the Build() function.</p> | | |
| Code: | <pre>//Define the dependencies var path = require("path"); var processor = require("processor"); var windows = require("windows"); var processIdConverter = new processor.Chain() .Convert({ fields: [{from: "winlog.event_data.ProcessId", to: "process.pid", type: "long"}, {from: "winlog.event_data.NewProcessId", to: "process.child.pid", type: "long"}, {from: "winlog.event_data.NewProcessName", to: "process.child.executable" }, {from: "winlog.event_data.ParentProcessName", to: "process.executable"}], mode: "copy", ignore_missing: true, fail_on_error: false, }) .Build();</pre> | | |

CREATING GENERIC FIELDS WITH LOOKUPS

| | |
|---------------------|--|
| Description: | <p>This code snippet below defines a function, stored as a variable called addEventFields, that creates new fields called event.category, event.type, and event.action that stores a human readable string describing the type of even that was created. These fields are generic and can be added to any event. Once in Elasticsearch, these fields can be used or searches, filters, aggregations, and to provide context to the operator.</p> <p>💡 LEARNING POINT: Break down your processors so that they can be used across multiple event types. This reduces rework and makes your processor script more robust.</p> |
| Code: | <pre>//Define the dependencies var path = require("path"); var processor = require("processor"); var windows = require("windows"); //Define a dictionary with key value pairs, where the key is a string representing the //winlog.event_id field and the value is an array of strings defined as //[event.category, event.type, event.action] var eventActionTypes = { "4624": ["authentication", ["start"], "logged-in"], "4625": ["authentication", ["start"], "logon-failed"], "4688": ["process", ["start"], "created-process"], "4689": ["process", ["end"], "exited-process"],</pre> |

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

```
"4697": [["iam", "configuration"], ["admin", "change"], "service-installed"],
"4698": [["iam", "configuration"], ["creation", "admin"], "scheduled-task-created"],
"4699": [["iam", "configuration"], ["deletion", "admin"], "scheduled-task-deleted"],
"4720": [["iam"], ["user", "creation"], "added-user-account"],
//Additional dictionary members omitted for brevity
};

var addEventFields = function(evt) {
  var code = evt.Get("event.code");
  if (!code) {
    return;
  }

  var eventActionDescription = eventActionTypes[code][2];
  if (eventActionDescription) {
    evt.Put("event.category", eventActionTypes[code][0]);
    evt.Put("event.type", eventActionTypes[code][1]);
    evt.Put("event.action", eventActionTypes[code][2]);
  }
};
```


EXTRACTING DATA WITH REGULAR EXPRESSIONS

Description: This code snippet below defines a function, stored as a variable called `extractSchtasksCommandAndArguments`, that runs a regular expression to extract specific data from an internal XML blob.

In ECMAScript, regular expressions can be defined using the `RegExp` class which takes in two arguments: (1) the regex string and (2) configuration options. For this particular script, we pass in the “i” option which tells the regex engine that our expression is case insensitive.

We are trying to extract the command and arguments fields from the XML stored in the `winlog.event_data.TaskContent` field, which is not extracted by default. To do that, we use a grouping construct, highlighted in yellow below, within the regular expression. Inside of the grouping construct, we match any contents between those two tags.

```
"<Command>(.*?)</Command>"
```

 **LEARNING POINT:** It’s hard to search, filter, query, and aggregate fields with nested structured data without processing it. Regular Expressions are a great way to extract information into new fields that are easier to process once loaded into ElasticSearch.

Code:

```
//Define the dependencies
var path = require("path");
var processor = require("processor");
var windows = require("windows");

var extractSchtasksCommandAndArguments = function(evt) {
  //This line retrieves an event field generated by the windows Event Log
  var taskxml = evt.Get("winlog.event_data.TaskContent");
```

Weapon System Training – 2: Configuring SecurityOnion and Winlogbeats

```
//Verify that the property exists and has a value
if(!taskxml) {
    return;
}

//The task name will be added to the generic_message field, but is not used in the regex
var name = evt.Get("winlog.event_data.TaskName");

//Define the regular expressions
var commandRegExp = new RegExp("<Command>(.*?)</Command>", "i");
var argsRegExp = new RegExp("<Arguments>(.*?)</Arguments>", "i");

//Run the regular expressions on the XML in order to extract the desired fields
var command = commandRegExp.exec(taskxml);
var args = argsRegExp.exec(taskxml);

//Successful matches will return an array of string objects. The first element will be
//the fully matched regex containing the XML tags (e.g. <Command>), which we don't want.
//We are only interested in the contents between those tags, so we select the second
//element which contains the contents of the first group match.
evt.Put("winlog.event_data.command", command[1]);
evt.Put("winlog.event_data.args", args[1]);

//Create a new field called winlog.generic_message that contains the relevant information
//for this particular type of event.
evt.Put("winlog.generic_message", "(" + name + ") " + command[1] + " " + args[1]);
});
```

COMBINING PROCESSOR FUNCTIONS IN A CHAIN

| | |
|---------------------|--|
| Description: | The code snippet below combines multiple processor functions into a single processor chain that is executed for a particular event. |
| Code: | <pre>//Define the dependencies var path = require("path"); var processor = require("processor"); var windows = require("windows"); var event4688 = new processor.Chain() .Add(copySubjectUser) .Add(copySubjectUserLogonId) .Add(renameNewProcessFields) .Add(copyTargetUserToEffective) .Add(addEventFields) .Add(function(evt) { var commandline = evt.Get("process.child.command_line"); var pid = evt.Get("process.child.pid"); evt.Put("winlog.generic_message", "(" + pid + ") " + commandline); }) .Build();</pre> |