Logstash|Dfir|Powershell Cheat Sheet

Scenario: Use powershell to gather forensic data remotely into zipped json and parse it with few simple tricks.

You can extract many forensic data from host, triggering powershell script remotely and copying output into network share.

There, you can parse it with logstash as it appears in filesystem.

```
e.g. Get-Process|ConvertTo-Json|Out-File internal_folder\your.json; Compress-Archive -Path internal_folder - DestinationPath
/path/to/your/zipped/powershell/payload/zipname.zip;  copy-item -Path /path/to/your/zipped/powershell/payload/zipname.zip -Destination \\yourshare\zipname.zip
```

Parsing Powershell zipped json output {

This processes powershell output jsons, zipped and uploaded to elasticsearch server

Really good for any custom-made dfir actions, that involves triggering powershell scripts on endpoints

Gets json from zip file and avoids processing again by removing json from archive

Remove any unexpected random output thanks to /dev/null redirection

```
exec{ command =>
"unzip -p /path/to/your/zipped/powershell/payload/$(ls -Acrt /path/to/your/zipped/powershell/payload/ | tail -n 1) internal_folder\\\\your.json 2>/dev/null;
zip -d /path/to/your/zipped/powershell/payload/$(ls -Acrt /path/to/your/zipped/powershell/payload/ | tail -n 1) internal_folder\\\\your.json 2>/dev/null >/dev/null"
codec => json{charset => "UTF-16"}
interval => 60 }
```

Replacing annoying double backslashes from windows logs with single backslash {

Many guys struggling doing it via gsub, you can't do this directly and need to use ruby

Especially handy if you're working with windows-generated powershell reports

```
ruby{ code => "event.set("your_field", event.get("your_field").gsub!("\\\\", "\u005c".encode(„UTF-8")))"}
```

Checking if field doesn't exist {

handy if you're trying to drop some unmeaningful records, checks if specific field doesn't exists and drops record if so

Works with nested fields as well

works with boolean operators: and or xor nand

```
if ![value]     {    drop { }      }
```

Removing all nested fields in category {

handy if you're working with huge json tables e.g process properties in windows

easily drops all json fields described as [foo][bar]

```
ruby{    code =>  'event.to_hash.keys.each{ |k| if k.start_with?("Foo") then event.remove(k) end}'    }
```

works also in reverse, removing fields with same name, from various categories

```
ruby{    code =>  'event.to_hash.keys.each{ |k| if k.ends_with?("Bar") then event.remove(k) end}'     }
```

Parsing windows timestamps {

when working with powershell output, timestamps generally is exported in common format: name.of.the.timestamp.field(0000000000000)

this snippet converts the timestamp to kibana timestamp

uses regex to find your timestamp

```
mutate{      gsub => ["your_timestamp_field","\D",""]    }

if [your_timestamp_field] =~ /\d+/ {
        date{
            match => ["your_timestamp_field","UNIX_MS"]
            target => "@timestamp"
            timezone => "UTC"
        }
        mutate{
            remove_field => ["[your_timestamp_field]"]
        }
}
```

Insert metadata from filename {

Use the zip filenames to transfer metadata of your investigation into your logs. Insert this snippet before other commands in your exec input

e.g. forensic_12-12-2020_10-53_domain_ads.zip

then remove it from your log in filter

```
Filter{
ruby{
   code => "str = `$(ls -Arct /path/to/file | tail -n 1)`
      event.set('filename', str)" }
    grok{    match => {"filename" => "\S+\_\d+\-\d+-\d+\_\d+\-\d+\_%{DATA:Host}.zip"}   }
    mutate{  remove_field => ["[filename]"]
}
}
```

Lucas Geras | 2020 | lucas.geras@gmail.com