MISP restSearch module development

Building a simple export module for the core

CIRCL / Team MISP Project

FIRST workshop

Building a native restSearch export

- Similar in scope to an **export module** of the MISP modules system
- Pros:
  - Can be used for composited data coming from a **filtered query**
  - Fast, **native approach**
  - Can be built to support **several scopes** (events, attributes, sightings)
- Cons...

Building a native restSearch export

- Similar in scope to an **export module** of the MISP modules system
- Pros:
  - Can be used for composited data coming from a **filtered query**
  - Fast, **native approach**
  - Can be built to support **several scopes** (events, attributes, sightings)
- Cons...

So how does restSearch work?

- Standardised way of collecting **parameters**
- Using the parameters, a loop is started to **chunk and gradually build** our export data
- The chunk size depends on memory envelopes
- Each chunk is **converted piece by piece**...
- ... and subsequently are concatenated into a temporary file
- Once no more elements are left, the file is sent in the response

Where does the module system come into play?

- The export modules handle 5 tasks:
  - Pass **meta-information** back to restSearch on the export format itself
  - Add a **start segment** to the exported data
  - Do the actual **conversion** from MISP's internal format to the desired export format
  - Provide a **separator** for data chunks
  - Have a **closing segment** for the returned data, based on the formatś conventions

Our little training module: Nibbler, the ever hungry IDS/IPS

Nibbler

- Simplistic tool with its **own proprietary format**
- Meant to mimic a typical **in-house tool**
- Lightweight scope, for simplicityś sake
- **pipe separated values**
- VALUE | TYPE | DESCRIPTION | REFERENCE | ACTION

Nibbler format - caveats

- Rules can be prepended by comments, each comment line starting with #
- Some characters have to be escaped in some custom, crazy ways
  - linebreaks: ##LINEBREAK##
  - commas: ##COMMA##
  - pipes: ##PIPE##

Nibbler format

- **Value**: The actual indicator value
- **Type**: The format of the indicator
- **Description**: A quick description for analysts investigating the alert, why is this relevant
- **Reference**: A backreference that the analyst can use to find out more about the alert
- **Action**: What should Nibbler do if it trips over the value?

Supported types

- IP
- Domain
- Hostname
- MD5
- SHA1
- SHA256
- Filename

Supported values

- ALERT - default behaviour, create an alert.
- BLOCK - block the action outright. Only set if the tag nibbler:block is present

Mapping the types to MISP

- Though we have types to map from MISP, in some cases several types map to a Nibbler type
- We've created a rough **mapping** (this is probably the most difficult task) in advance
- Some MISP types map to a Nibbler type directly
- **Composite** MISP types map to **2 Nibbler types** each

Mapping the types to MISP

- ip-dst :: IP
- ip-src :: IP
- domain :: Domain
- domain|ip :: Domain, IP
- hostname :: Hostname
- md5 :: MD5
- sha1 :: SHA1
- sha256 :: SHA256
- filename|md5 :: Filename, MD5
- malware-sample :: Filename, MD5
- filename|sha1 :: Filename, SHA1
- filename|sha256 :: Filename, SHA256

Export module skeleton

```
<?php
class NibblerExport
{
    public $additional_params = array();
    public function handler(
        $data, $options = array()
    ) {}
    public function header(
        $options = array()
    ) {}
    public function footer() {}
    public function separator() {}
}
```

Additional parameters

```
public $additional_params = array(
    'flatten' => 1
);
```

Adding our mapping

```
private $__mapping = array(
  'ip-dst' => 'IP',
  'ip-src' => 'IP',
  'domain' => 'Domain',
  'domain|ip' => ['Domain', 'IP'],
  'hostname' => 'Hostname',
  'md5' => 'MD5',
  'sha1' => 'SHA1',
  'sha256' => 'SHA256',
  'filename|md5' => array('Filename', 'MD5'),
  'malware-sample' => array('Filename', 'MD5'),
  'filename|sha1' => array('Filename', 'SHA1'),
  'filename|sha256' => array('Filename', 'SHA256')
);
```

Writing the start of the output

```
public function header($options = array())
{
    return sprintf(
        "# Nibbler rules generated by MISP at %s\n",
        date('Y-m-d H:i:s')
    );
}
```

Footer function - how should the output end?

```
public function footer()
{
    return "\n";
}
```

What separates the chunks?

```
public function separator()
{
    return "\n";
}
```

The actual legwork, the handler

```
public function handler($data, $options = array())
{
  if ($options['scope'] === 'Attribute') {
    $data['Attribute']['AttributeTag'] = $data['AttributeTag'];
    return $this->__convertAttribute($data['Attribute'], $data['Event']);
  }
  if ($options['scope'] === 'Event') {
    $result = array();
    foreach ($data['Attribute'] as $attribute) {
      $temp = $this->__convertAttribute($attribute, $data['Event']);
      if ($temp) $result[] = $temp;
    }
    return implode($this->separator(), $result);
  }
  return '';
}
```

Building an optional internal converter function

```
private function __convertAttribute($attribute, $eve
{
  if (empty($this->__mapping[$attribute['type']])) {
    // mapping not found - invalid type for nibbler
    return '';
  }
  if (is_array($this->__mapping[$attribute['type']]))
    // handle mappings for composites - slide
  } else {
    // handle simple mappings - slide
  }
  // return 1 or 2 lines, separated by separator()
  return implode($this->separator(), $result);
}
```

Handling the simple case

```
$result[] = sprintf(
  '%s|%s|%s|%s|%s',
  $this->__escapeSpecialChars($attribute['value']),
  $this->__mapping[$attribute['type']],
  $event['uuid'],
  $this->__escapeSpecialChars($event['info']),
  'ALERT'
);
```

Handling the case for composites

```
$attribute['value'] = explode(
  '|', $attribute['value']
);
foreach (array(0,1) as $part) {
  $result[] = sprintf(
    '%s|%s|%s|%s|%s',
    $this->__escapeSpecialChars(
      $attribute['value'][$part]
    ),
    $this->__mapping[$attribute['type']][$part],
    $event['uuid'],
    $this->__escapeSpecialChars($event['info']),
    'ALERT'
  );
}
```

Putting it together

```
private function __convertAttribute($attribute, $event) {
  if (empty($this->__mapping[$attribute['type']])) return '';
  $result = array();
  $attributes = array();
  if (is_array($this->__mapping[$attribute['type']])) {
    $attribute['value'] = explode('|', $attribute['value']);
    foreach (array(0,1) as $part) {
      $result[] = sprintf(
        '%s|%s|%s|%s|%s',
        $this->__escapeSpecialChars($attribute['value'][$part]),
        $this->__mapping[$attribute['type']][$part],
        /events/view/ . $event['uuid'],
        $this->__escapeSpecialChars($event['info']),
        $this->__decideOnAction($attribute['AttributeTag'])
      );
    }
  } else {
    $result[] = sprintf(
      '%s|%s|%s|%s|%s',
      $this->__escapeSpecialChars($attribute['value']),
      $this->__mapping[$attribute['type']],
      /events/view/ . $event['uuid'],
      $this->__escapeSpecialChars($event['info']),
      $this->__decideOnAction($attribute['AttributeTag'])
    );
  }
  return implode($this->separator(), $result);
}
```

Adding the function that decides on the action

```
private function __decideOnAction($attributeTags)
{
  foreach($attributeTags as $attributeTag) {
    if (
      $attributeTag['Tag']['name'] ===
        'nibbler:block'
    ) {
      return 'BLOCK';
    }
  }
  return 'ALERT';
}
```

Finalising the export module… The escaping function

```
private function __escapeSpecialChars($value)
{
  $value = preg_replace(
    "/\r|\n/", "##LINEBREAK##", $value
  );
  $value = preg_replace(
    "/,/", "##COMMA##", $value
  );
  $value = preg_replace(
    "/\|/", "##PIPE##", $value
  );
  return $value;
}
```

Modifying the MISP core to know about the export module

- The **models** that we are targeting by scope (Event, Attribute) **need to be updated**
- They are located in **/var/www/MISP/app/Model/**
- The global variable **$validFormats** houses all mappings
- Simply add a new line such as the following:
- 'nibbler' => array('nibbler', 'NibblerExport', 'nibbler')

Let us test the module!

- Use the **rest client** to test it conveniently
- Both the event and attribute level restSearch function should work
- Simply set the **returnFormat** to nibbler, which should also show up as a valid export format

REST CLIENT

HTTP method to use

POST

Relative path to query

/events/restSearch

- [ ] Use full path - disclose my apikey
- [ ] Bookmark query
- [x] Show result
- [ ] Skip SSL validation

HTTP headers

```
Authorization: ArSxnHf20foSapnOSyxfrljMdl9oLDnvmqvHK97q
Accept: application/json
Content-Type: application/json
```

HTTP body

```
{
    "returnFormat": "nibbler",
    "page": 1,
    "limit": 4,
    "type": ["ip-dst", "ip-src", "domain|ip", "hostname", "domain"]
}
```

Run query