SPLICE Splunk Indicator of Compromise Engine

Introduction

SPLICE provides a way of consuming IOCs in Splunk to leverage the embedded atomic indicators and provide greater context than common threat feeds. Once ingested, those IOCs becomes searchable across all your data - any kind of data - using a set of commands. SPLICE allows you to leverage Filenames, Hashes, Domain Names, URLs and IP Addresses (more atomic indicators coming soon).

SPLICE relies on libtaxii, python-cybox and python-stix to support STIX v1.1 (including v1.1.1), CybOX v2.1 and OpenIOC v1.0

SPLICE can monitor local directories, or mount points, for incoming IOCs as well as TAXII feeds to periodically poll IOCs.

SPLICE has been successfully tested with: Avalanche v1.3, v1.4, and MISP v1.0. Support for Soltra (aka Avalanche 2.0) coming soon.

OpenIOC

A typical OpenIOC can look like the following example found on www.iocbucket.com

```
<?xml version="1.0" encoding="us-ascii"?>
<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ↪xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="9
    ↪bd56357-844e-4cb9-97c8-0a0f954f262a" last-modified
    ↪="2013-08-08T18:02:12" xmlns="http://schemas.mandiant.
    ↪com/2010/ioc">
  <short_description>Reveton</short_description>
  <description>http://bharath-m-narayan.blogspot.com
      ↪/2013/08/live-security-professional.html</description
      ↪>
  <authored_by>Megan Carney</authored_by>
  <authored_date>2013-08-08T17:55:36</authored_date>
  <links />
  <definition>
    <Indicator operator="OR" id="0ed08e74-ec02-495b-baf8-3
        ↪ab5fc61be05">
      <IndicatorItem id="d93e769b-dc89-413f-93bf-
          ↪ecd538d13290" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum
            ↪" type="mir" />
        <Content type="md5">2eeeaa69b70944cac8a30545b3f49b77
            ↪</Content>
      </IndicatorItem>
      <IndicatorItem id="61c6dfdc-6b59-473e-9643-
          ↪a712e8c44e92" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum
            ↪" type="mir" />
        <Content type="md5">e3675273325b7f7df3b13fe93cd30fac
            ↪</Content>
      </IndicatorItem>
      <IndicatorItem id="76672f94-0014-425e-a58a-53
          ↪f5a372b37b" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum
            ↪" type="mir" />
        <Content type="md5">8a6e45d16c82c4c79cbd7730207183ca
            ↪</Content>
      </IndicatorItem>
      <IndicatorItem id="b0ac27a7-14a5-46b7-91df-
          ↪b2ffe97f55ac" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum
            ↪" type="mir" />
        <Content type="md5">e5a2409ad36943053135ba9bd3e08ba6
            ↪</Content>
      </IndicatorItem>
      <IndicatorItem id="0dd5b32d-4002-45a8-abba-
          ↪abebd6c2316f" condition="contains">
        <Context document="UrlHistoryItem" search="
            ↪UrlHistoryItem/URL" type="mir" />
        <Content type="string">beg.rocklandgrad.com/forum/wm
            ↪/keys/board.php?connect=17</Content>
```

```
      </IndicatorItem>
      <IndicatorItem id="4e4e5842-2bc5-46c7-a5fc-230
          ↪f899c139a" condition="contains">
        <Context document="UrlHistoryItem" search="
            ↪UrlHistoryItem/URL" type="mir" />
        <Content type="string">beg.rocklandgrad.com/forum/wm
            ↪/keys/WFolw</Content>
      </IndicatorItem>
      <IndicatorItem id="2d388b3f-5928-4583-99b9-3
          ↪eabc8d167b5" condition="contains">
        <Context document="UrlHistoryItem" search="
            ↪UrlHistoryItem/URL" type="mir" />
        <Content type="string">beg.rocklandgrad.com/forum/wm
            ↪/keys/7T8INre2</Content>
      </IndicatorItem>
      <IndicatorItem id="08dd1e7c-e3de-4cc2-ba58-85
          ↪a1f690fdb5" condition="contains">
        <Context document="UrlHistoryItem" search="
            ↪UrlHistoryItem/URL" type="mir" />
        <Content type="string">down.jjconway.com/backend.php
            ↪?nomic=638&amp;main=7&amp;watch=112&amp;energy
            ↪=1121&amp;beta=400&amp;bugs=134&amp;linux=168&
            ↪amp;rates=371&amp;apply=677&amp;outdoors
            ↪=1569755419</Content>
      </IndicatorItem>
    </Indicator>
  </definition>
</ioc>
```

Several `blocks` exist in this IOC for different types of information. To make it simple, we can assume that an IOC is a collection of `IndicatorItem` blocks like the following:

```
      <IndicatorItem id="2d388b3f-5928-4583-99b9-3
          ↪eabc8d167b5" condition="contains">
        <Context document="UrlHistoryItem" search="
            ↪UrlHistoryItem/URL" type="mir" />
        <Content type="string">beg.rocklandgrad.com/forum/wm
            ↪/keys/7T8INre2</Content>
      </IndicatorItem>
```

Or,

```
      <IndicatorItem id="76672f94-0014-425e-a58a-53
          ↪f5a372b37b" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum
            ↪" type="mir" />
        <Content type="md5">8a6e45d16c82c4c79cbd7730207183ca
            ↪</Content>
      </IndicatorItem>
```

Those `IndicatorItem` blocks actually handle the real content that's interesting from a Splunk perspective: the data you want to search for in your particular context. In the previous example, the actual data is :

- a URL (`UrlHistoryItem/URL`) : `beg.rocklandgrad.com/forum/wm/keys/7`
  `↪T8INre2`
- a MD5 Hash of a file (`FileItem/Md5sum`) : `8a6e45d16c82c4c79cbd7730207183ca`
  `↪`

STIX, CybOX

STIX is a standard developed by MITRE and seeks to describe threat information in a comprehensive and detailed manner. With STIX, one should be able to describe attack campaigns, threat actors, courses of action, TTPs, exploit targets, indicators, observables and more. The coverage of STIX is significantly larger than OpenIOC.

STIX uses CybOX to describe indicators and while the logic is essentially the same as presented for OpenIOC, the language is different. Here is an example of a CybOX object:

```
<cybox:Properties xsi:type="FileObj:FileObjectType">
    <FileObj:File_Name>Iran's Oil and Nuclear Situation.doc
        ↪</FileObj:File_Name>
    <FileObj:Size_In_Bytes>106604</FileObj:Size_In_Bytes>
    <FileObj:Hashes>
       <cyboxCommon:Hash>
          <cyboxCommon:Type>MD5</cyboxCommon:Type>
          <cyboxCommon:Simple_Hash_Value condition="Equals">
              ↪E92A4FC283EB2802AD6D0E24C7FCC857</cyboxCommon
              ↪:Simple_Hash_Value>
       </cyboxCommon:Hash>
    </FileObj:Hashes>
</cybox:Properties>
```

Or,

```
<cybox:Properties xsi:type="URIObj:URIObjectType" type="URL
    ↪">
    <URIObj:Value condition="Equals">www.documents.myPicture
        ↪.info</URIObj:Value>
</cybox:Properties>
```

In CybOX language, indicators are `cybox:Properties`. Just like in the example with OpenIOC, the two previous samples describe :

- a URL (`URIObj:URIObjectType`) : `www.documents.myPicture.info`

- a MD5 Hash (`FileObj:FileObjectType`) : `E92A4FC283EB2802AD6D0E24C7FCC857` ↪

IOC Logic

The atomic indicators are linked to each other in an IOC file with Boolean logic. More complex IOCs can be described as in the following example: (Indicator A) AND ((Indicator B) OR (Indicator C)) AND NOT (Indicator D).

This logic is not supported in the current release of SPLICE (1.1) because:

- In most cases, IOCs are created to search for the existence of something. In other words, only few IOCs exist that test the non-existence of something (e.g., the key Y is not present in the registry).
- IOCs are created by humans and humans make mistakes. Alerting on a small subset of an IOC is still relevant from a security perspective, especially in SIEM or SOC contexts.
- Threats evolve. An IOC may describe version X of a malware but the next iteration of the malware may change slightly causing the IOC to become inaccurate.

However, note that understanding the IOC Boolean logic is on the roadmap of SPLICE.

SPLICE

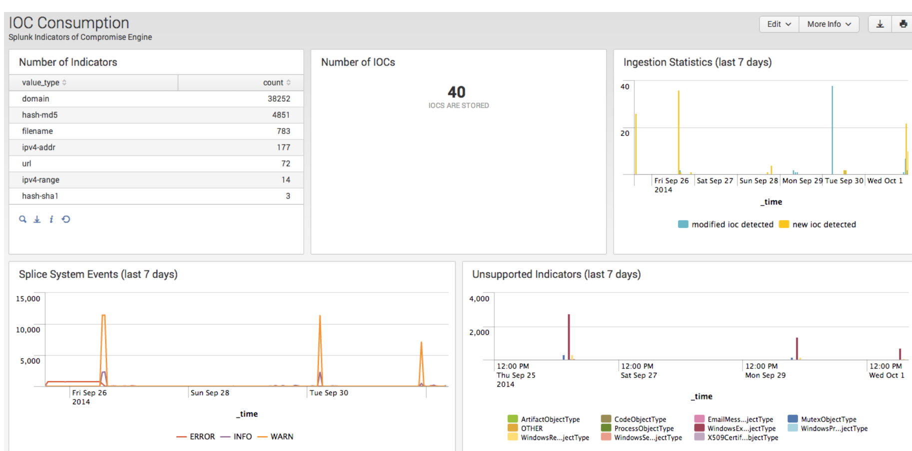


Figure 1: SPLICE

Installing SPLICE

SPLICE can be found on Splunk Base. Once downloaded, use the Manage Apps menu as you would do with any other Splunk add-on : *Apps > Manage Apps > Install app from file > select the previously downloaded file.*

Please note that SPLICE has only been tested on Linux systems at this time.

SPLICE relies on MongoDB to store the ingested IOCs and the extracted atomic indicators. You will need to setup a MongoDB instance somewhere in your environment in order for SPLICE to work (use the regular MongoDB installation process). Ideally, you would install this MongoDB instance on the Splice Search Head but you could alternatively install it elsewhere.

The first time you access SPLICE, you will be asked to configure the minimally required options illustrated by the following screens:

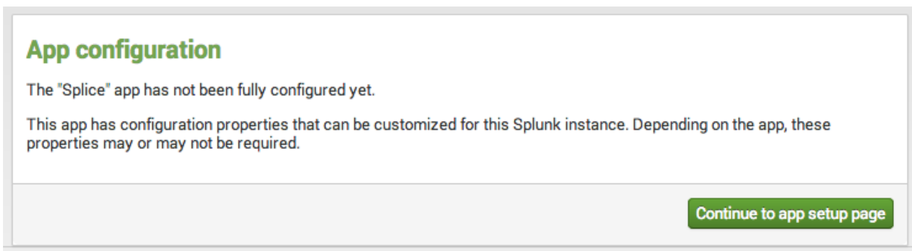


Figure 2: App Configuration

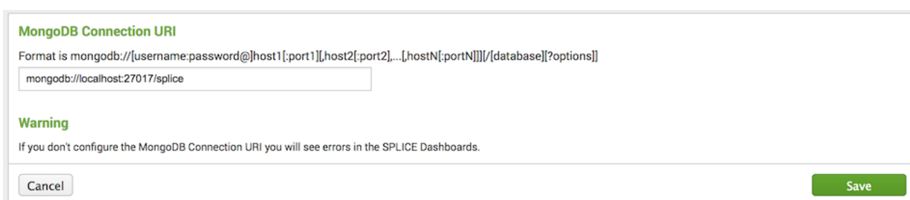


Figure 3: MongoDB Connection URI

Since SPLICE stores the IOCs and the extracted atomic indicators in a MongoDB server, this step is absolutely mandatory. This configuration will be shared across all modular inputs and the custom search commands of SPLICE.

Upgrading to v1.2

No particular steps are required to upgrade to v1.2 from v1.0 or v1.1. Simply install the new app through Splunk App Manager and check the "`Upgrade app...`" checkbox.

Note: The scheduled searches have changed in v1.2. The CIM mapping for the IP searches changed from "src" to "src_ip" and "dest_ip" in order to make the search more specific. Also, due to user feedback, the previous individual scheduled hash searches were replaced with a scheduled universal hash search called `ioc_default_search_hash`

Splunk license impact

SPLICE does **not** impact your Splunk license:

- IOCs are stored directly in MongoDB (out of the scope of Splunk licensing)
- Matching events are stored using the `collect` command which works only on already indexed data (see Automating IOC searches for details)
- SPLICE is a free app.

Data Inputs

SPLICE provides two Modular Input options that are accessible via two new menus under the **Settings > Data Inputs** menu. Note that those Modular Inputs rely on MongoDB and so the installation steps need to have been successfully completed (installation steps).

Mount point monitor The Modular Input "IOC - Mount point monitor" allows monitoring of directories for incoming IOCs. Those directories can be local directories or mount points with at least read-only permissions. This Modular Input will monitor *.ioc* and *.xml* files (case insensitive).

Once a new file is detected, or an existing file has been modified, the file is read and stored in the MongoDB in its original form (collection "raw"). The stored IOC is marked as "to be parsed" in order to extract from it the atomic indicators.

Please note that the name you use for the data input will be the name that is used for the IOC Sources dashboard to compare your different sources of IOCs.

The configuration should be quite straightforward like in the following example:

TAXII Feeds TAXII defines a set of services and message exchanges that, when implemented, enable sharing of actionable cyber threat information across organization and product/service boundaries. TAXII, through its member specifications, defines concepts, protocols, and message exchanges to exchange cyber threat information for the detection, prevention, and mitigation of cyber threats.

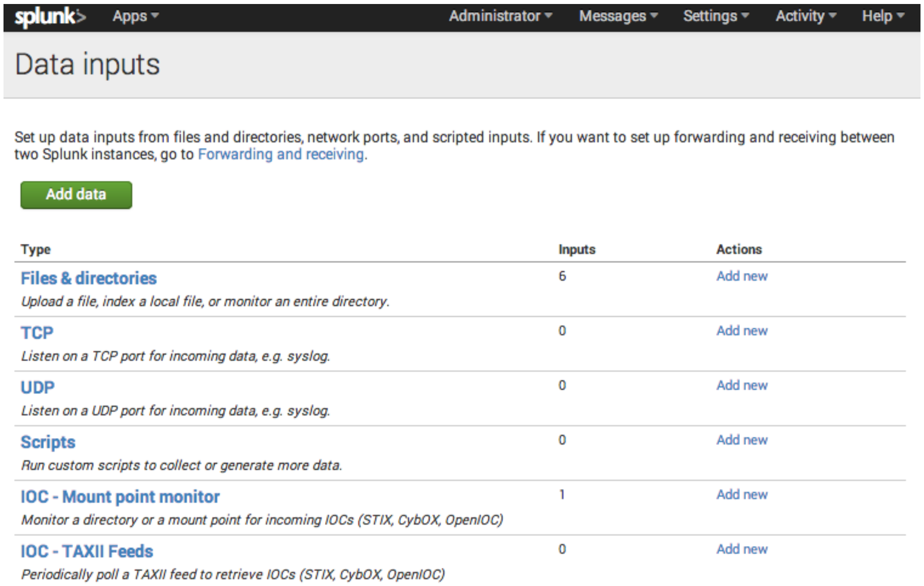


Figure 4: Data Inputs

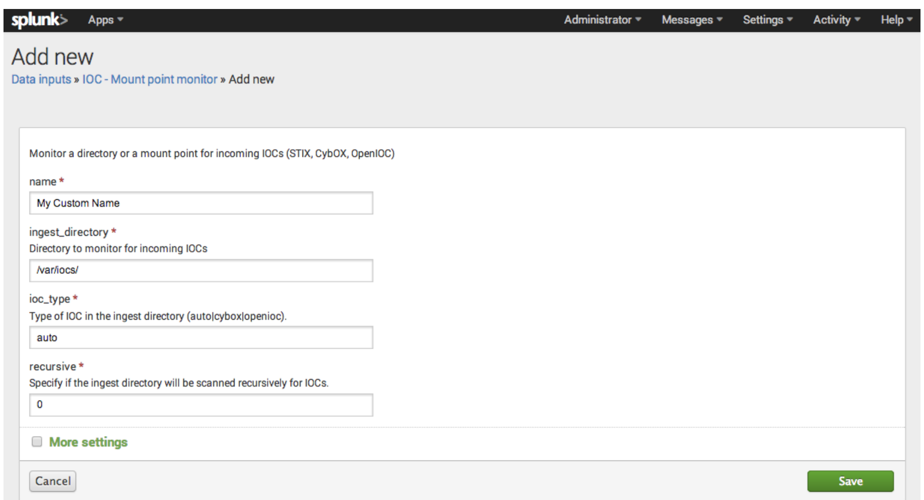


Figure 5: Data Inputs Mount Point

TAXII is the preferred method of exchanging information represented using the Structured Threat Information Expression (STIX) language, enabling organizations to share structured cyber threat information in a secure and automated manner.

Please note that the time synchronization between your TAXII server and your Splunk server is **very important** as the retrieved IOCs are filtered by time. If a clock drift exists, you may miss some IOCs or even consume the very same IOC multiple times.

IOCs and their Indicators

An IOC, regardless of its description language (OpenIOC, STIX, ...), is composed of `atomic indicators` (`IndicatorItem`, `cybox:Properties`, ...). This term is a generic way of describing the smallest amount of measurable data in the IOC for one to utilize for correlation.

Moreover, an IOC is generally built with a unique identifier (id) like in the following examples:

```
# STIX header

<stix:STIX_Package  xmlns:cyboxCommon="http://cybox.mitre.
    ↪org/common-2"  xmlns:cybox="http://cybox.mitre.org/
    ↪cybox-2"  ....
     id="opensource:Package-c6afaad1-92e7-4c01-b18f-
         ↪eb3fdca0247d" version="1.1.1"  timestamp
         ↪="2014-09-17T17:01:41.076336+00:00">
```

Or,

```
# OpenIOC header

<?xml version="1.0" encoding="us-ascii"?>
<ioc xmlns="http://schemas.mandiant.com/2010/ioc" ....
     id="7ae7eec2-c722-4995-8dfd-2fd904d38cdf" last-modified
         ↪="2013-12-18T23:04:46">
```

SPLICE extracts this information when the IOC is ingested and references it as `ioc_id`. As this field is optional, every IOC ingested by SPLICE will also have an internal id referenced as `ioc_raw_id`.

This logic also applies to the atomic indicators. Splice uses `indicator_id` to reference the indicator id when it's present, or if its missing, `indicator_raw_id` for the SPLICE's internal indicator id.

Every IOC and atomic indicator stored by SPLICE is accessible by the user (see `iocdisplay` command). However, please note that not setting an id for either the IOC or its indicators is generally considered a bad practice. Future versions of STIX may make the ids mandatory.

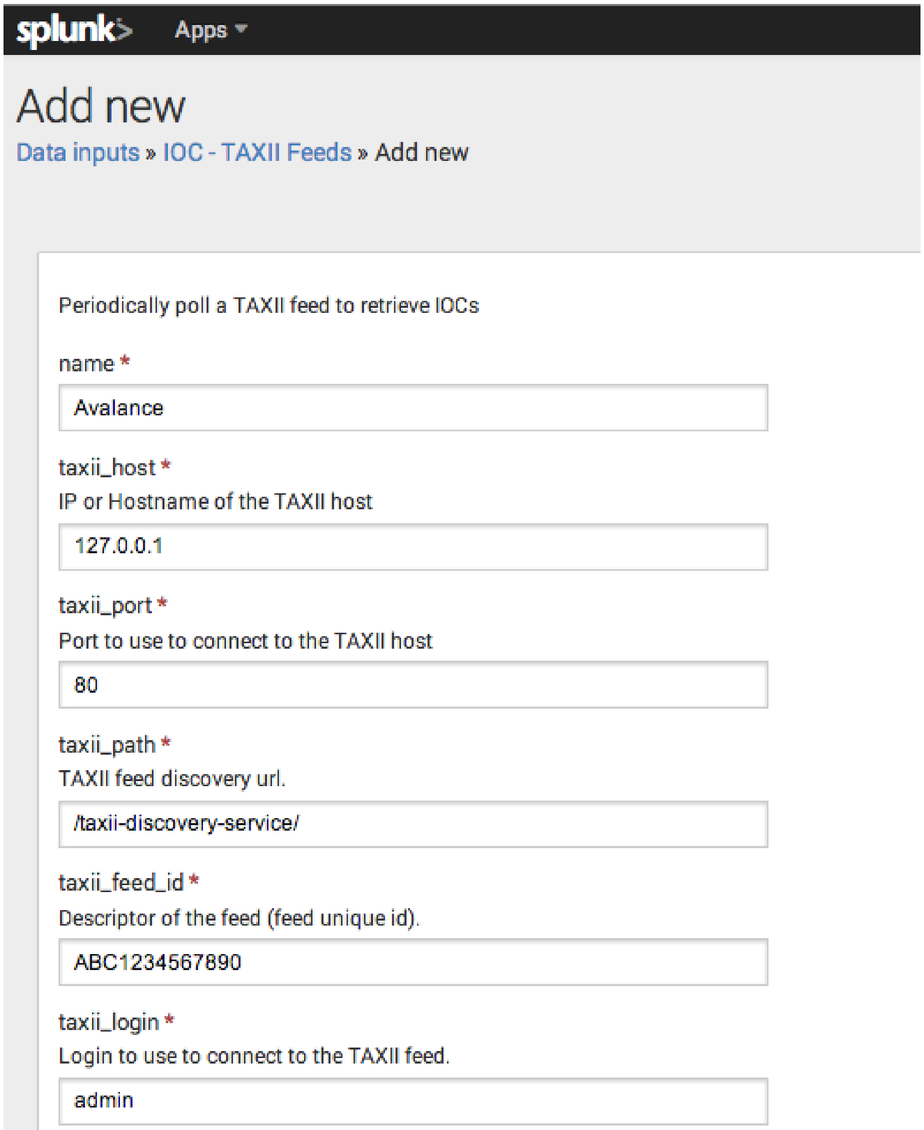


Figure 6: Data Inputs TAXII Feed

Finally, an IOC stored in the Mongo database looks like the following:

```
# Example of an IOC stored in the Mongo database.
{
        "_id"    : ObjectId("542b716feaa76eb49c7be3d4"),
        "ioc_id" : "opensource:Package-c6afaad1-92e7-4c01-
            ↪b18f-eb3fdca0247d",
        "path"   : "/var/iocs/stix/sample_zeustracker.xml",
        content" : "<xml ...> ... content of the IOC file
            ↪... </xml>",
        // .. other fields ...
}
```

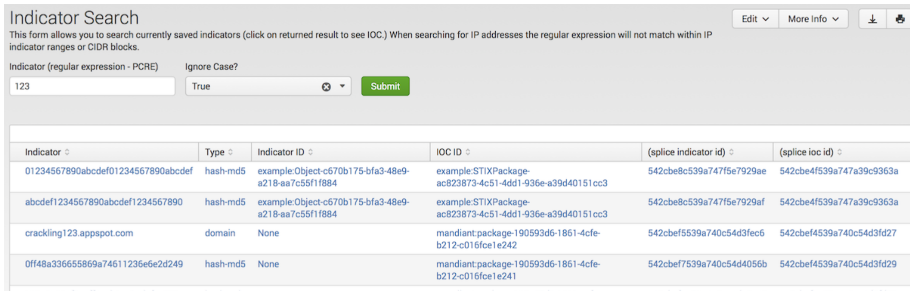
And atomic indicators look like:

```
# Example of possible atomic indicators
{
        "_id"          : ObjectId("542b716feaa76eb49c7be3e7
            ↪"),
        "raw_id"       : ObjectId("542b716feaa76eb49c7be3d4
            ↪"),
        "ioc_id"       : "opensource:Package-c6afaad1-92e7-4
            ↪c01-b18f-eb3fdca0247d",
        "indicator_id" : "opensource:File-fa253ff2-05fa-40b7
            ↪-9f75-092691234a0c",
        "value"        : "config.bin",
        "value_type"   : "filename",
}
{
        "_id"          : ObjectId("542b716feaa76eb49c7be3e8
            ↪"),
        "raw_id"       : ObjectId("542b716feaa76eb49c7be3d4
            ↪"),
        "ioc_id"       : "opensource:Package-c6afaad1-92e7-4
            ↪c01-b18f-eb3fdca0247d",
        "indicator_id" : "opensource:Address-50bb69cc-7b76
            ↪-4444-8175-dad1b4f54062",
        "value"        : "23.252.120.143",
        "value_type"   : "ipv4-addr",
}
```

SPLICE Forms

Indicator Search

The Indicator Search form can be used to search for the existence of indicators within SPLICE. The form is searching for the saved indicators within MongoDB. The search comes in the form of a regular expression along with an option to ignore case. You may click the results to view the IOC in the IOC Viewer form.



*Please note that the search is a PCRE Expression and in case of IP address searches, the address will not be searched across subnets. However, searches for IP addresses will work if the saved indicator is a single IP address.

IOC Viewer

The IOC Viewer allows you to view your stored IOCs. To retrieve an IOC, you may use any of the following fields: IOC ID (`ioc_id`), Indicator ID (`indicator_id` ↪), SPLICE Indicator ID (`indicator_raw_id`) and SPLICE IOC ID (`ioc_raw_id`). Besides seeing the raw IOC text, you can also view IOC key-value pairs.

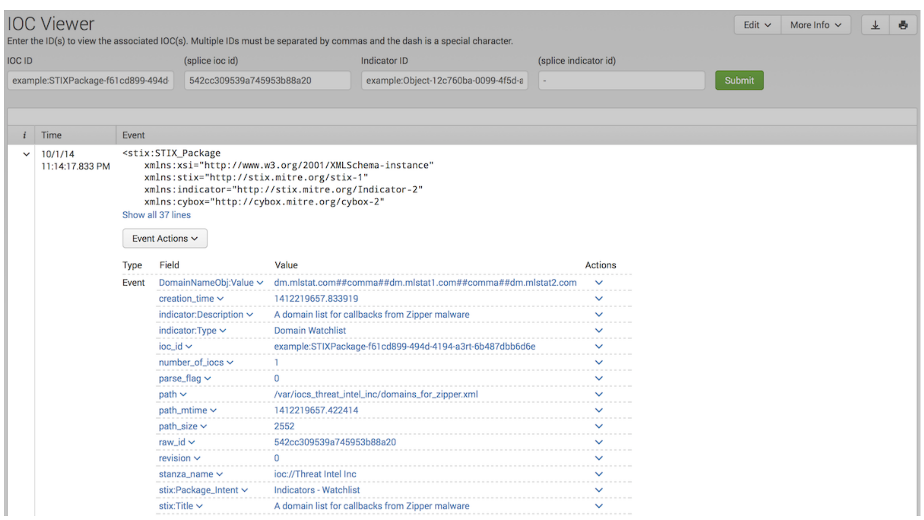


Figure 7: ioc_viewer

SPLICE Dashboards

There are four pre-built dashboards included in SPLICE. These dashboards utilize many of the custom search commands you'll see in the next sections of this document.

1. `IOC Alerts` includes information on recent correlated indicator alerts. The alerts are by default saved in the `ioc_search_results` index and all of the alert dashboards utilize this index.

2. `IOC Alert Trending` focuses on alert trends and statistics.

3. `IOC Sources` provides information on IOC data for different IOC sources. The sources are retrieved by the name of the Data input.

4. `IOC Consumption` provides information on IOCs, Indicators and SPLICE system events. This information is retrieved from MongoDB and logs.

IOCSearch

The IOCSearch command is designed to search your data for the atomic indicators that were extracted from the ingested IOCs. Those atomic indicators are read from the MongoDB at search time.

The syntax is as follow:

```
.. your search.. | iocsearch map="<fields mapping to use>"
```

The `iocsearch` command appends 2 new fields to your events: `ioc_indicators_count` ↪ and `ioc_indicators_json`. The first one will contains the number of matching atomic indicators while the second will contain the details of each atomic indicator that matched.

To correctly parse the results contained in the field `ioc_indicators_json`, a macro has been created to align the field names regarding the workflow action expectations (see example below). The macro will produce the following fields:

- `indicator_object_id` : The ID of the atomic indicator.
- `indicator_value` : The real indicator that has been searched (for example, 'evildomain.tld' or '127.0.0.1')
- `indicator_value_type` : The type of data. Refer to type mapping below.
- `indicator_type` : either `value` or `regex`
  - `value` : strict matching of the content, the `indicator_value`.
  - `regex` : `indicator_value` is evaluated as a PCRE regex.

The `ioc_indicators_json` can contain one or multiple indicators results, depending on your IOCs and the mapping for your search.

Type mapping

The mapping (option `map`) is used to assign a *type* to fields in order to search the corresponding atomic indicators.

The format is: `field_name:field_type` where `field_type` belongs to any of the following:

| field_type | meaning | iocsearch specifics | |————|————|———————| | ipv4-addr | IPv4 address| searched across ipv4-addr, ipv4-range and ipv4-cidr | | ipv4-range | IPv4 netblock with format 'from'-'to' (ex: 1.2.3.4-1.2.8.9)| searched across ipv4-range and ipv4-cidr | | ipv4-cidr | IPv4 netblock with CIDR notation (ex: 1.2.3.4/16)| searched across ipv4-range and ipv4-cidr | | domain | a domain name (host.tld)| - | | url | a url (http://host.tld/page)| - | | filename | name of the relevant file| - | | hash-*<type>* | hash of the relevant file (hash-md5, hash-sha1, hash-sha256, etc)| strict match in lowercase ('a'=='a') | | hash | generic type that refers to all known hashes types (hash-md5, hash-sha1, hash-sha256, etc). This should only be used when the searched fields are of an unknown or multiple hash type. **NOTE:** You could likely get duplicate alerts from this and the hash-*<type>* scheduled searches. You may want to tune the searches to only search relevant hash data based on your environment.| - |

A map can be constructed from one or more associations (see example below). However, it is not possible to type multiple times the same field within the same map (ex: `src:ipv4-addr,src:domain` will fails). If you want to do so, you have to split your map into multiple searches.

Sample usage

Here is an example from a search of proxy logs:

```
index=proxy | iocsearch map="cs_Referer:url,cs_host:domain,
    ↪c_ip:ipv4-addr,src:ipv4-addr"
```

In this example, the field `cs_Referer` is designed as an URL, the field `cs_host` is designed as a domain name while the `src` and `c_ip` fields are designed as IPv4 addresses.

The next example will filter the results of an iocsearch to display only events having indicators identified and use the macro to correctly parse the output.

```
..your search.. | iocsearch map="<your-mapping>"| search
    ↪ioc_indicators_count>0 | `parse_ioc_indicators_json`
```

Workflow Actions: from atomic indicator to IOCs

The workflow actions require specific field names to appear in your search results. This is the case with the fields `ioc_id`, `indicator_id`, `indicator_raw_id` and `ioc_raw_id`. When these fields exist, a new drop-down menu appears and lets you display the corresponding IOCs associated with this/those object_id(s).

In the `Event Actions` or in the `field event action` a new action named `Display ↪associated IOCs` allows you to drill down from raw results (atomic indicators) to the complete IOCs that included those indicators.

Figure 8: IOCSearch Workflow Action

CIM mapping

While the iocsearch command can use any field name, it's **highly recommended** to rely on Common Information Model (CIM) field names (src_ip, dest_ip, etc). Those will ensure compatibility with Enterprise Security and permit an easy integration in SPLICE Dashboards.

For more information on the CIM, please see CIM and the CIM App

Default Scheduled Searches

Out of the box, SPLICE provides default scheduled searches relying on CIM fields: `ioc_default_search_domain`, `ioc_default_search_filename`, `ioc_default_search_hash`, `ioc_default_search_ipv4-addr`, `ioc_default_search_ipv6 ↪-addr` and `ioc_default_search_url`. Those searches are run every hour (+ 10 minutes) on the previous hour. You may also want to create some additional scheduled searches that would run periodically and look further back just for any new indicators (for example, search back 30 days).

If any match is found, the following actions are performed :

- Storage of the matching event in the default `ioc_search_results` index. A field `marker` will be added to events to let you know which saved searches triggered.
- The RSS feed will be populated (ex: http://<*your splunk host*>/en-US/rss/ioc_default_search_ipv4)

- An Alert will be triggered (see saved searches section for details)

Those default searches are designed to be very generic and can be deactivated using the regular `Settings > Searches, reports, and alerts > App context (` `↪Splice)` menu.

Tip: If you choose to use searches, it is highly recommended that you customize and tune them. For example, you should try to narrow the search for each one to only where matching data could exist. Some commons ways of doing this would be to restrict the search to only certain indexes and/or sourcetypes based on what you are searching for in each individual saved search .

Understanding SPLICE Scheduled Searches

Here are the minimal steps to follow to successfully automate the searching and recording of IOCs in your data.

The first step is to understand how the `iocsearch` command works. Please refer the to IOCSearch chapter.

Minimal steps

The following steps are an illustration of the automation of searches of IOCs across proxy logs. The workflow is as follow:

1. Periodically search logs using the `iocsearch` command
2. Store the results in a dedicated index (log lines having at least one match to any indicator). You should use the index `ioc_search_results` created by SPLICE as this index is also used by SPLICE activity dashboards.
3. Once your matching events are indexed, you can create dashboards, alerts, etc, that fit your needs and context.

Example

Here is an example of such a search (points 1 and 2 from above).

```
index=proxy_logs
  | iocsearch map="cs_uri:url,cs_host:domain,src:ipv4-addr"
  | search ioc_indicators_count > 0
  | `parse_ioc_indicators_json`
  | rename _raw as raw_
  | collect index=ioc_search_results marker="marker=\"proxy
    ↪\""
```

Explanation:

- Line 1-4: this is standard call to the `iocsearch` command using the appropriate macro to correctly format fields. Please refer to the IOCSearch chapter for more details.
- Line 5: rename the `_raw` field in order to keep all the extracted fields and make sure they are stored with the data (next line).
- Line 6: collect all the data and store it in the index `ioc_search_results`. This index **must** exist before calling the `collect` command. Moreover, we add a marker (field: `marker`, value: `proxy`) to make it easier to later reference those events (for custom dashboards for example).

The result of this command should be rather ugly as we rename the `_raw` field. However, this step only is only intended to store the matching events, not necessarily to present them to the user.

Keeping the results

Finally, you can see the stored events in the index `ioc_search_results` by running the regular query `index=ioc_search_results`. However, to make it look nicer, you may want to rename the field `raw_`:

```
index="ioc_search_results" | rename raw_ as _raw
```

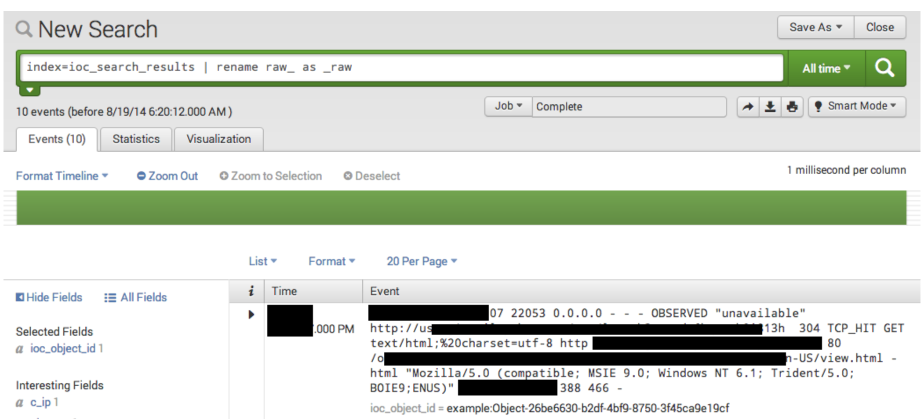


Figure 9: ioc_search_results

By following those steps you will store the SPLICE fields with the data, which are required by the workflow actions. In other words, the workflow actions associated to this field are available from the index `ioc_search_results`. Please refer to the workflow actions for more details.

Be aware *that if you run the same search multiple times using the* `collect` *command, you will append duplicate results to the existing index. In other words, if you don't take caution about the time period you are searching before using the* `collect` *command, you will be storing multiple instances of the very same results in the index* `ioc_search_results`*. This will lead to incorrect SPLICE dashboards and the frequency of the individual indicators will be skewed.*

A triggered alert (Activity > Triggered Alerts > Splice) will looks like the following:

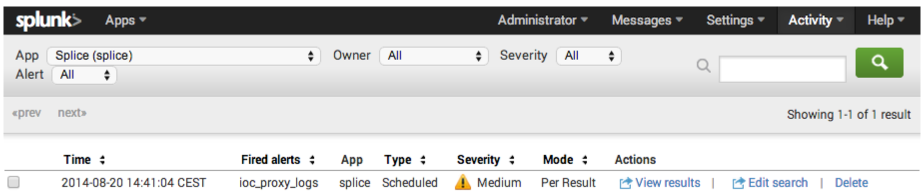


Figure 10: triggered alert

Fields sourcetype, source, index and host

The collect command will rename a few fields. For example, the field sourcetype will become `stash` instead of your original sourcetype. However the original information is not lost and is kept in renamed fields that start with `orig_` like `orig_sourcetype`.

To replace the `stash` sourcetype by the original one you can run the following command:

```
index=ioc_search_results
| rename raw_ as _raw
| rename orig_sourcetype as sourcetype
```

IOCStats

The IOCStats command is a command dedicated to retrieve simple statistics about IOCs stored in your mongoDB instance in order to populate SPLICE dashboards. To list the available statistics, use the `list` option as follow:

```
| iocstats stat=list
```

Note that this command is a generating command and therefore must be the very first argument in the Splunk search query.

For example, to get the number of stored IOCs simply use:

```
| iocstats stat=number_of_iocs
```

IOCFilter

The Splunk command `iocfilter` has been created to let you search the stored atomic indicators. This command includes the parameter `regex` which is the pattern (PCRE) to search for and a flag `ignorecase` that turns on or off the case sensitivity. Note that this flag is optional at search time and is set to `False` by default. Morevover, as SPLICE can (de)activate specific indicators, `iocfilter` let you decide if you want to display disabled indicators through the `displaydisabled` boolean flag.

```
| iocfilter regex="<PCRE Pattern>" [ignorecase=True|False] [
    ↪displaydisabled=True|False]
```

Here is an example that searches for the word `micro` and ignores the case (can match `Micro`, `MIcro`, `micRO`, `micro`, etc).

```
| iocfilter regex="MICRO" ignorecase=True
```

When using metacharacters in the pattern like `\d`, `\s`, `\w`, etc, you must escape them. As a general rule, the backslash must be escaped (or doubled) like in the following example that searches for the pattern `2\d1`:

```
| iocfilter regex="2\\d1" ignorecase=True
```

In case you forgot to do so, the following error will appears:

```
Error in 'iocfilter' command: command="iocfilter", Poorly
    ↪formed string literal: <your-pattern>
```

IOCExportCSV

The `iocexportcsv` command is designed to export atomic indicators as CSV and has been especially created to work with the Enterprise Security 3+ Threat List framework.

This command expects 4 parameters:

- `value_type` : the type of atomic indicator to export
- `alias` : the alias to rename the csv header
- `directory` : a directory where splunkd would be able to write the CSV file
- `filename` : exported filename

At this time, no append mode exists and the command exports all atomic indicators at once. This command also appends the extension '.csv' if it was not included in the parameter `filename`.

```
| iocexportcsv value_type="<type of atomic indicator to
    ↪export>" alias="<header name>" directory="<path>"
    ↪filename="<csv filename>"
```

Example:

```
| iocexportcsv value_type="ipv4-addr" alias="ip" directory
    ↪="/tmp" filename="myIpList.csv"
```

As an output you should see a log message similar to this one:

```
file /tmp/myIpList.csv created with 311 entries
```

The produced CSV file will look like the following example, where the field description will be filled with the atomic indicator unique id extracted from the original IOC. If a single value is referenced by multiple atomic indicators, only one will be outputed.

```
# head /tmp/myIpList.csv
description,ip
example:Address-9c915242-671d-47e4-bcb5-17035e1902d7
    ↪,61.129.33.35
fireeye:object-b7f34877-3ba5-44e9-9a36-c16ba7d7ad94
    ↪,122.200.124.57
fireeye:object-80ee612f-d300-462c-af4a-883baaaeeea8
    ↪,98.126.211.218
fireeye:object-afe13da0-2b49-4f7d-a0e4-46b9b8eaceec
    ↪,98.126.211.219
example:Address-8c2716f3-e09a-4f1b-9a5d-88f2ef1f1a21
    ↪,124.232.135.84
fireeye:object-142748e8-8000-4029-8291-31cd191a48ce
    ↪,219.90.112.203
fireeye:object-4ce029f3-4805-4ace-9b0c-43dfa12cb06f
    ↪,74.208.56.101
fireeye:object-4090671d-0e98-4768-935b-32c2c9de6ed3
    ↪,121.41.129.179
fireeye:object-40ad5a6c-e776-4cbb-a423-e266bc6d940a
    ↪,204.74.215.58
```

IOCDisplay

This command has been created to allow one to retrieve one or more IOCs based on the submitted object IDs (atomic indicator IDs) or IOC IDs. This command will give you access to the raw content of the IOCs.

The usage is as follow:

```
| iocdisplay object_id="<comma separated list of object IDs
    ↪>"
```

or

```
| iocdisplay ioc_id="<comma separated list of IOC IDs>"
```

Note also that the `ioc_id` parameter admit a star as input value (`ioc_id="*"`) to retrieve all the IOCs stored in the mongo.

Sample usage:

```
| iocdisplay object_id="example:Address-7e3827ad-9019-494a-
    ↪b8ae-2c24c3749442,example:Object-1980ce43-8e03-490b-863
    ↪a-ea404d12242e"
```

IOCToggle

This command has been created to change the state of atomic indicators based on any provided identifier (ioc_id, ioc_raw_id, indicator_id, indicator_raw_id). When calling this command with an IOC ID, the state (enabled/disabled) is toggled for all of the indicators that associated to this IOC. The toggle sets indicators to `enabled` when they are `disabled` and to `disabled` when they are `enabled`.

Disabling an indicator means that this indicator remains in the database but will no more be returned by the `iocsearch` command. This is generally used to reduce false positives or improper indicator definitions.

The usage is as follow:

```
| ioctoggle ioc_id="<comma separated list of object IDs>"
```

Note that you can use `ioc_id`, `ioc_raw_id`, `indicator_id` and `indicator_raw_id` (and you can combine them if needed).

Sample usage:

```
| ioctoggle ioc_id="mandiant:package-e33ffe07-2f4c-48d8-b0af
    ↪-ee2619d765cf"
```

The command `ioctoggle` also supports an optional parameter, `all`. When used, the current state of the referenced indicators are disregarded and are all set to the specified value (`enabled` or `disabled`).

Sample usage:

```
| ioctoggle ioc_id="mandiant:package-e33ffe07-2f4c-48d8-b0af
    ↪-ee2619d765cf" all="disabled"
```

SPLICE Parsers

SPLICE v1.2 introduces the notion of **parsers** by externalizing the code that actually transforms a supported indicator (STIX, CybOX, OpenIOC) to a Splunk Indicator from the core of SPLICE .

The parsers (python scripts) are stored in the `$SPLUNK_HOME/etc/apps/splice/` ↪`bin/splice/parsers/` directory.

Parsers have been externalized as modules to let users write their own parsers: maybe you have IOCs with currently unsupported indicators and you want to add support for them, maybe you want to edit the provided default parsers to enhance them, etc.

- Parsers need to have the very same name (case sensitive) as the indicator they refer to (ex: `AddressObjectType.py` or `FileItem.py`).
- Parsers are expecting input and are expected to return data in a normalized way (see details below).
- Parsers must implement the Base Class in order to integrate smoothly (and inherit few useful functions!) within SPLICE core.

SPLIndicator

`SPLIndicator` is a crucial piece in SPLICE parsers because they represent the atomic indicator in a way that Splunk can understand them through SPLICE.

The signature of such an indicator is as follows:

```
SPLIndicator(indicator_id, type, value_type, value)
```

- `indicator_id` is the identifier found for the the indicator (could be `None`).
- `type` must be either `value` (strict matching, equality) OR `regex` (evaluated as a PCRE). The `type` field defines how the `value` field will be evaluated by SPLICE during a search.
- `value_type` is a normalized type of data (`ipv4-addr`, `domain`, `url`, etc - please refer to data types section in this documentation).
- `value` is the searchable value extracted for the atomic indicator (`192.168.1.1` ↪, `www.google.com`, `32`, etc).

Parsers for STIX, CybOX

A module must be named like the CybOX object they refer to. For example, to create a module for the object `AddressObjectType`:

1. Create a python script named `AddressObjectType.py` in the parsers directory
2. Create a class `AddressObjectType` in that script that inherits from the `BaseObjectType` class
3. Instantiate the `parse()` method

As input, the class will receive:

1. The identifier of the atomic indicator (`object_id`)
2. All the properties of the CybOX object (`properties`)

As output, the parser **must** return an array of `SPLIndicator`

Here is the skeleton of a `AddressObjectType`module:

```
from .. import splindicator
from .. import common
from .. import errors

from BaseObjectType import BaseObjectType

__author__  = "<YOUR NAME>
__version__ = "1.0.0"
__email__   = "<YOUR MAIL>"

class AddressObjectType(BaseObjectType):

    def parse(self):

    if self.properties == None:
        return []

    self.log("INFO", "This is an example to call the logger
        ↪")

    # ... define how to parse the object here ...

    return [] # an array of SPLIndicator _MUST_ be returned
```

You'll notice that a logger is also provided by the Base class. The severity provided to the logger should be `INFO`, `WARNING` or `WARN`, `ERROR` or `DEBUG`. Note that the `self.log()` function will automatically append the `object_id` to the provided message in order to give the end-user relevant logs messages.

Parsers for OpenIOC

A module must be named like the OpenIOC `document` they refer to. For example, to create a module for the object `FileItem`:

1. Create a python script named `FileItem.py` in the parsers directory
2. Create a class `FileItem` in that script that inherits from the `BaseItem` class
3. Instantiate the `parse()` method

As input, the class will receive:

1. The identifier of the atomic indicator (`object_id`)
2. The properties of the `IndicatorItem` (`indicator_item`)
3. The properties of the `content` item (`content`)
4. The properties of the `context` item (`context`)

As output, the parser **must** return an array of `SPLIndicator`

Here is the skeleton of a `FileItem` module:

```
from BaseItem import BaseItem

__author__  = "<YOUR NAME>"
__version__ = "1.0.0"
__email__   = "<YOUR MAIL>"

class FileItem(BaseItem):

    def parse(self):
        return self.generic_parser()
```

Note that OpenIOC Indicators are very similar even if they describe various things (`FileItem`, `DnsEntryItem`, etc). You are completely free to instantiate the `parse()` method as you wish but in most cases the default `generic_parser()` method may be enough.

Just like STIX/CybOX parsers, a `log()` method is also defined.

Events and Logging

Logging schema

The Modular Inputs as well as the custom search commands rely on the Splunk SDK.

They both use the provided loggers and functions to write log messages. In other words, they don't know where or how the log messages are written to as they use functions provided by the SDK.

There are two kinds of messages:

- Log messages, like info, warning, error messages, generated by the custom search commands. They translate a state of the command and are written to `$SPLUNK_HOME/var/log/splunk/splunkd.log` They can be accessed through the `_internal` index.
- Events, which are only generated by the Modular Inputs. They aren't log messages and they are written to the `main` index. Those events can be compared to log lines read from a regular log file. So, when an IOC is read, an Event "this IOC has been read" is generated (this is different from a log message).

Please note that the logging scheme has a known bug

Internal Events

Those events can be accessed with the following search:

```
index=_internal source="*/splunkd.log" splice
```

Here are a few samples of log lines:

```
08-13-2014 00:11:04.823 +0200 INFO  ExecProcessor - message
    ↪from "python /opt/splunk/etc/apps/splice/bin/ioc.py"
    ↪event_type="new ioc detected" ioc_file="/var/iocs/
    ↪STIX_sample.xml" mtime="1407881460.51" size="266268"
    ↪status="ioc successfully stored"
08-13-2014 00:55:51.953 +0200 INFO  ExecProcessor - message
    ↪from "python /opt/splunk/etc/apps/splice/bin/ioc.py"
    ↪event_type="modified ioc detected" ioc_file="/var/iocs/
    ↪STIX_sample.xml" mtime="1407884150.92" size="266268"
    ↪status="new size (is:266268, was:266269)"
08-13-2014 00:56:26.981 +0200 INFO  ExecProcessor - message
    ↪from "python /opt/splunk/etc/apps/splice/bin/ioc.py"
    ↪event_type="modified ioc detected" ioc_file="/var/iocs/
    ↪STIX_sample.xml" mtime="1407884184.15" size="266268"
    ↪status="new modification time (is:1407884184.15, was
    ↪:1407884150)"
08-07-2014 01:31:02.182 +0200 ERROR ExecProcessor - message
    ↪from "python /opt/splunk/etc/apps/splice/bin/ioc.py"
    ↪ERRORcould not connect to localhost:27017: [Errno 111]
    ↪Connection refused
```

History

SPLICE v1.2 - 2014/11/17 - Feature Enhancements

- STIX/CybOX parser externalization
- New OpenIOC parsing logic (compatible with OpenIOC v1.0 and v1.1)
- OpenIOC parser externalization
- Core code reorganization
- python-libtaxii upgraded to 1.1.104 (was: 1.1.102)
- python-stix upgraded to 1.1.1.2 (was: 1.1.1.0)
- python-cybox upgraded 2.1.0.8 (was: 2.1.0.6)
- Dashboard menus renamed (backend only)
- Documentation updates
- Ability to (de)activate atomic indicators
  - `iocfilter` command new parameter `displaydisabled`
  - Form `Indicator Search` new drop-down choice
  - `iocsearch` command modified to exclude deactivated indicators
  - new command `ioctoggle` to change the state of indicators
- TAXII dual authentication support with pem/key cert files
- [fix] wrong timezone used for the TAXII feeds
- `iocsearch` command now supports a generic hash type `hash`
- The previous individual scheduled hash searches were replaced with a scheduled universal hash search called `ioc_default_search_hash`
- CIM mapping for the `ioc_default_search_ipv4-addr` and `ioc_default_search_ipv6` ↪`-addr` scheduled searches changed from `src` to `src_ip` and `dest` to `dest_ip`

SPLICE v1.1 - 2014/10/08 - Public release

- Bug fixes
- New features

SPLICE v1.0 - 2014/09/01 - Restricted access release

- Initial release

Known Bugs

Logging Issue

> SPLICE relies on the Splunk SDK for some pre-built functions like the one used for logging events or error messages. It has been observed that some error messages are not correctly written to the logger when using the search_command template from the SDK.
>
> A workaround is to set the `handlers` to `file` in the `logging.conf`. The messages will be written in a chosen file instead of the regular `$SPLUNK_HOME/var/log/splunk/splunkd.log`. Further investigation is in progress.

The external search command '<command>' did not return events in descending time order, as expected.

> This error is most of the time related to an improper input like an incorrect IOC (example: defining an IPv4 with a digit above 255 like 192.168.256.4). This error is also related to the logging issue (see above) and will be fixed in a later release of SPLICE.

Using the fields command after iocsearch empty the field ioc_indicators_json

When using the command fields after the output of iocsearch, the field ioc_indicators_json is erased/cleaned. Results are lost.

```
...  | iocsearch map="src:ipv4-addr" | fields
    ↪ioc_indicators_json
```