Blueliv.

Advanced search: IOA dork

Table of contents

Introduction

The dork “ioa” is one of the most powerful dorks we can find in the Malware Hunting section of Threat Context. This dork will permit to find malware with a specific behaviour, because the Indicator of Actor (IoA) section of the malware reports generated by our sandbox will be scrutinized when we use this dork. As it is possible to see in the IoA tab of a malware sandbox report, the kind of information we can find is:

- Domains
- IPs
- URLs
- Network connections
- Registry Keys: written, deleted, created, opened, enumerated and read.
- Paths: PDB paths, created/removed files and directories, written files, etc.
- Yara signatures
- Mutexes
- Binary Certificates
- Metadata: file metadata, different hashes (CRC32, PE Imphash, ssdeep), PEiD signatures, PE timestamp, etc.

The full list of available IoA keys can be found in “Appendix I. Full IoA JSON keys” and the relation between the Blueliv sandbox report and a given “ioa” dork in “Appendix II. Relation between sandbox reports and “ioa” dork”.

How to use the “ioa” dork

As we mentioned before, the IoA section of the JSON report is fully indexed, so we can use dots to refer to specific parts of it. For example:

```
ioa.metadata.crc32.original:"2E548B06" # Exact match
ioa.ip:"192.168.56.102" # Exact match
ioa.url:~"priceminister.com" # Fuzzy match (contains)
ioa.ip:^"8.8" # Starts with match
ioa.url:$".ru" # Ends with match
```

It is important to keep in mind that wildcard searches are “expensive” and can slow down the search.

In the case of searching for strings we have to consider that if the string contains certain characters Lucene, the library behind dork searches, will tokenize the words so we will not be able to search for a whole string if we don't escape the characters or we use an exact match. The characters that should be escaped are (link to Lucene documentation):

+ - && || ! ( ) { } [ ] ^ " ~ * ? : \ /

So, for instance, if we want to search for "g:\65_vc8\VBE6\legovbe\opt\VBE6.pdb" (872f05e70b26d6b4e0f2936db82cb94614143d9db4d67984fb7e9c47dd857fce), it is not possible to do:

```
ioa.path.pdb_path:"g:\65_vc8\VBE6\legovbe\opt\VBE6.pdb"
```

Instead, we could do:

- For exact match the following can be used (**keyword** is the key here):

```
ioa.path.pdb_path.keyword:"g:\65_vc8\VBE6\legovbe\opt\VBE6.pdb"
```

- In case of partial match we must escape some characters:

```
ioa.path.pdb_path.keyword:~"g\:\\65_vc8\\VBE6\\legovbe\\opt\\VBE6.pdb"
```

These are some useful dorks which can be used when performing Threat Hunting tasks:

Registry keys

- ioa.regkeys
  - ioa.regkeys.regkey_written
  - ioa.regkeys.regkey_created
  - ioa.regkeys.regkey_deleted

File paths

- ioa.path.filepaths
    - ioa.path.filepaths.file_created
    - ioa.path.filepaths.dll_loaded
    - ioa.path.filepaths.file_moved
    - ioa.path.filepaths.file_written
    - ioa.path.filepaths.directory_created

PDB path

- ioa.path.pdb_path

PE Compilation timestamp

- ioa.metadata.pe_timestamp

Mutexes

- ioa.mutex

IPs contacted

- ioa.ip

Domains resolved/contacted

- ioa.domain

URLs contacted

- ioa.url

Names of processes being executed

- ioa.process_name

Yara signatures triggering

- ioa.yara

Appendix I. Full IoA JSON keys

This section shows all the available keys which can be used together with the "ioa" dork and appearing in the JSON report the Blueliv sandbox generates:

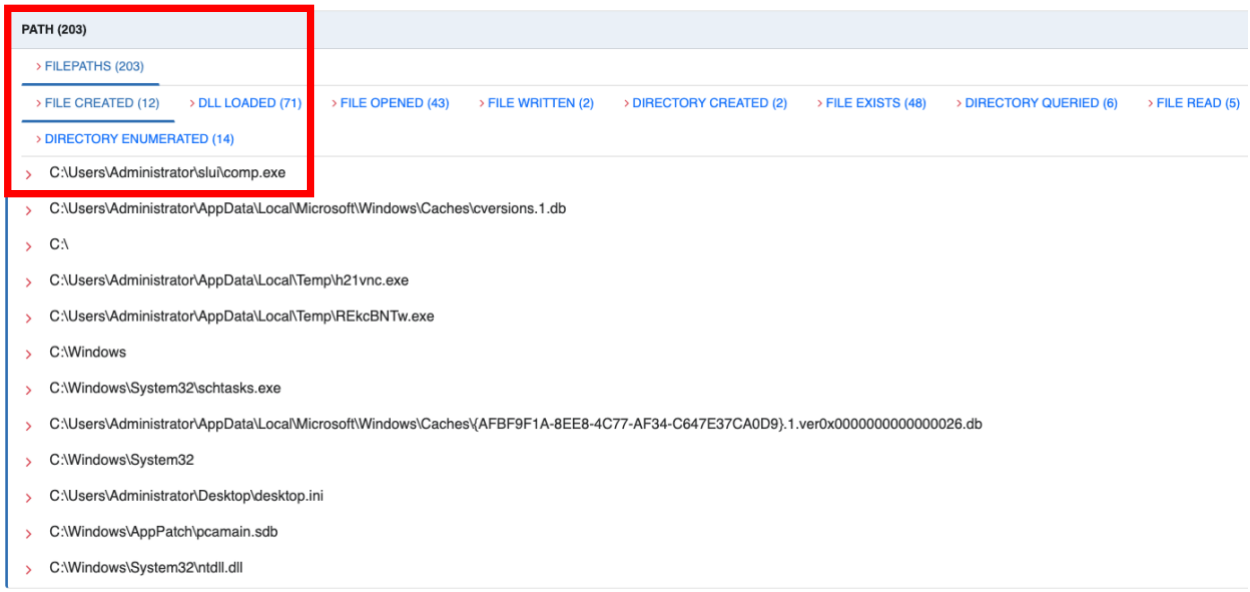
- "domain"
- "regkeys"
  - "regkey_written"
  - "regkey_opened"
  - "regkey_enumerated"
  - "regkey_deleted"
  - "regkey_created"
  - "regkey_read"
- "ip"
- "process_name"
- "connections"
  - "tcp_dead"
  - "udp"
  - "tcp"
- "host"
- "path"
  - "pdb_path,
  - "filepaths"
    - "file_created"
    - "dll_loaded"
    - "file_opened"
    - "file_moved,
    - "file_written"
    - "directory_created"
    - "file_copied,
    - "file_deleted,
    - "directory_removed,
    - "file_exists"
    - "directory_queried"
    - "file_read"
    - "directory_enumerated"
- "yara"
  - "generic"
  - "url,
  - "pre_analysis,
  - "misc"
    - "packer"
    - "misc"
    - "crypto"
  - "memory"
- "url"

- "ports"
    - "tcp_dead"
    - "udp"
    - "tcp"
- "mutex"
- "certificates"
- "email"
- "metadata"
    - "signing_date"
    - "crc32"
        - "original"
        - "unpacked"
    - "pe_imphash"
    - "names"
        - "title"
        - "original_filename"
        - "producer"
        - "locality"
        - "country"
        - "legal_trademarks"
        - "creator"
        - "legal_copyright"
        - "organizational_unit"
        - "company_name"
        - "internal_name"
        - "private_build"
        - "common_name"
        - "organization"
        - "author"
        - "subject"
        - "product_name"
        - "special_build"
    - "peid_signatures"
    - "file_type"
        - "original"
        - "unpacked"
    - "postal_code"
    - "ssdeep"
        - "original":
        - "unpacked"
    - "pe_timestamp"

Appendix II. Relation between sandbox reports and "ioa" dork

When a malware sample report is visualized, from the IoA tab we can see the following information:

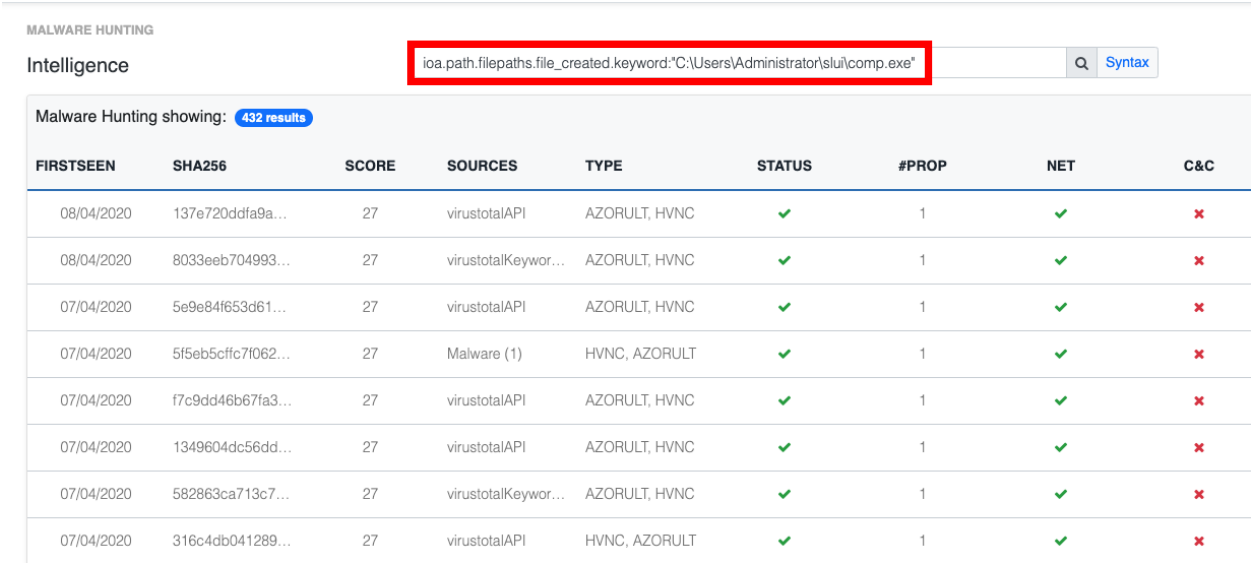
Clicking in the different categories we can find elements and subcategories like in the example PATH - FILEPATHS – FILE CREATED:



This categories and subcategories are a direct mapping from the keys that we can find in the JSON report that the Blueliv sandbox generates after analysing a file:


```
"path": {
  "pdb_path": [],
  "filepaths": {
    "file_created": [
      "C:\\Users\\Administrator\\slui\\comp.exe",
      "C:\\Users\\Administrator\\AppData\\Local\\Microsoft\\Windows\\Caches\\cversions.1.db",
      "C:\\",
      "C:\\Users\\Administrator\\AppData\\Local\\Temp\\h21vnc.exe",
      "C:\\Users\\Administrator\\AppData\\Local\\Temp\\REkcBNTw.exe",
      "C:\\Windows",
      "C:\\Windows\\System32\\schtasks.exe",
      "C:\\Users\\Administrator\\AppData\\Local\\Microsoft\\Windows\\Caches\\{AFBF9F1A-8EE8-4
      "C:\\Windows\\System32",
      "C:\\Users\\Administrator\\Desktop\\desktop.ini",
      "C:\\Windows\\AppPatch\\pcamain.sdb",
      "C:\\Windows\\System32\\ntdll.dll"
    ],
    "dll_loaded": [
      "C:\\Windows\\System32\\wshqos.dll",
      "C:\\Windows\\system32\\sfc.dll",
      "rasman.dll",
      "urlmon.dll",
      "kernel32",
      "API-MS-Win-Security-LSALookup-L1-1-0.dll",
```


We can use those keys (or categories/subcategories) in the JSON tree to search samples with specific behaviours just adding the categories and subcategories followed by dots, as you can see in the following image:



It is important to always add the keyword parameter at the end to find correctly the samples and avoid tokenizations of the library behind our dork search.