CAKe: **C**EF **A**rmy **K**nife **e**xperiment

A versatile CEF manipulation and generation tool

cardinal_gaetan |at| yahoo.fr

Introduction

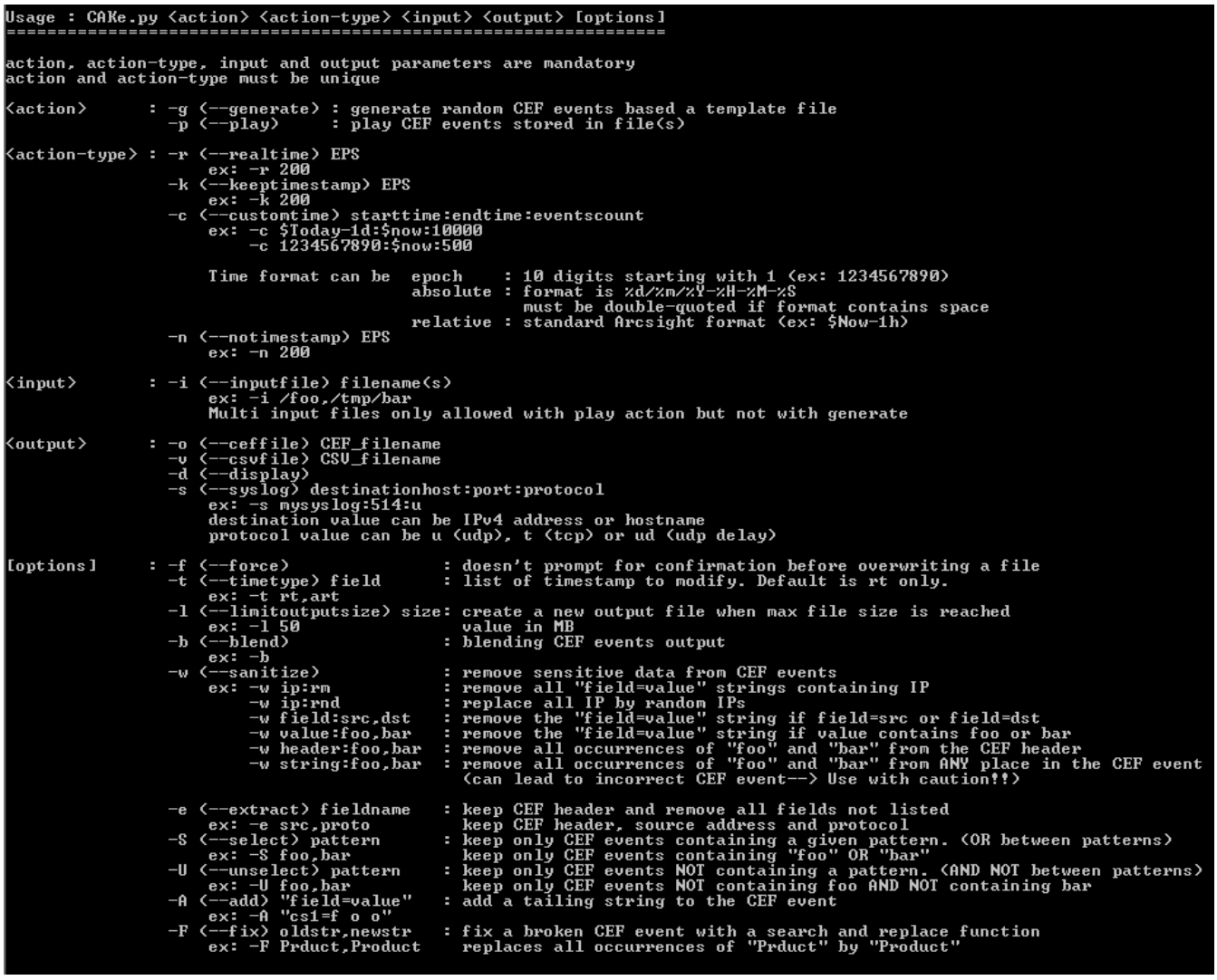
There are quite a few situations where it can be interesting to generate traffic based on CEF events. It can happen when:

- testing the amount of storage required for a certain type of events
- simulating CEF traffic flowing through a network
- measuring the maximum traffic load a connector can handle when receiving CEF events
- optimizing an hardware/software implementation in order to get the best performance
- testing some specific content you've built in ESM.
- creating a test lab simulating a production environment

There are some solutions available to simulate such traffic but we found the creation of realistic CEF events pretty difficult and time consuming. Of course it is possible to replay some real events but then you are facing issues with the timestamps which are in the past. Moreover you don't necessarily want to see your real events containing sensitive information in a testlab outside of your production network. The original idea behind CAKe was to answer to these problems by providing a way to easily modify CEF events timestamps while replaying existing CEF events and/or easily generating your own CEF events with pretty powerful customization options. Sanitization options have also been added to protect sensitive data from being disclosed.

Syntax

Usage



```
Usage : CAKe.py <action> <action-type> <input> <output> [options]
================================================================

action, action-type, input and output parameters are mandatory
action and action-type must be unique

<action>      : -g (--generate) : generate random CEF events based a template file
                -p (--play)     : play CEF events stored in file(s)

<action-type> : -r (--realtime) EPS
                    ex: -r 200
                -k (--keeptimestamp) EPS
                    ex: -k 200
                -c (--customtime) starttime:endtime:eventscount
                    ex: -c $Today-1d:$now:10000
                        -c 1234567890:$now:500

                    Time format can be  epoch    : 10 digits starting with 1 (ex: 1234567890)
                                        absolute : format is %d/%m/%Y-%H-%M-%S
                                                   must be double-quoted if format contains space
                                        relative : standard Arcsight format (ex: $Now-1h)
                -n (--notimestamp) EPS
                    ex: -n 200

<input>       : -i (--inputfile) filename(s)
                    ex: -i /foo,/tmp/bar
                    Multi input files only allowed with play action but not with generate

<output>      : -o (--ceffile) CEF_filename
                -v (--csvfile) CSV_filename
                -d (--display)
                -s (--syslog) destinationhost:port:protocol
                    ex: -s mysyslog:514:u
                    destination value can be IPv4 address or hostname
                    protocol value can be u (udp), t (tcp) or ud (udp delay)

[options]     : -f (--force)               : doesn't prompt for confirmation before overwriting a file
                -t (--timetype) field      : list of timestamp to modify. Default is rt only.
                    ex: -t rt,art
                -l (--limitoutputsize) size: create a new output file when max file size is reached
                    ex: -l 50                value in MB
                -b (--blend)               : blending CEF events output
                    ex: -b
                -w (--sanitize)            : remove sensitive data from CEF events
                    ex: -w ip:rm           : remove all "field=value" strings containing IP
                        -w ip:rnd          : replace all IP by random IPs
                        -w field:src,dst   : remove the "field=value" string if field=src or field=dst
                        -w value:foo,bar   : remove the "field=value" string if value contains foo or bar
                        -w header:foo,bar  : remove all occurrences of "foo" and "bar" from the CEF header
                        -w string:foo,bar  : remove all occurrences of "foo" and "bar" from ANY place in the CEF event
                                             (can lead to incorrect CEF event--> Use with caution!!)

                -e (--extract) fieldname   : keep CEF header and remove all fields not listed
                    ex: -e src,proto         keep CEF header, source address and protocol
                -S (--select) pattern      : keep only CEF events containing a given pattern. (OR between patterns)
                    ex: -S foo,bar           keep only CEF events containing "foo" OR "bar"
                -U (--unselect) pattern    : keep only CEF events NOT containing a pattern. (AND NOT between patterns)
                    ex: -U foo,bar           keep only CEF events NOT containing foo AND NOT containing bar
                -A (--add) "field=value"   : add a tailing string to the CEF event
                    ex: -A "cs1=f o o"
                -F (--fix) oldstr,newstr   : fix a broken CEF event with a search and replace function
                    ex: -F Prduct,Product    replaces all occurrences of "Prduct" by "Product"
```

Overview

CAKe is divided in **5** parts:

- **<Action>** is where you define if you want to use existing CEF file(s) as a starting point or if you want to generate new CEF events based on a customizable template file

- **<Action type>** is where you define how to deal with the timestamps and the EPS of the CEF events being sent to the output

- **<Input>** is the place where you define your input file(s) which can be CEF file(s) or a customizable template file.

- **<Output>** is where you define what you want to do with the CEF events you have modified or generated. You can store them in a file, convert them in CSV file format, display them to the screen, send them to a syslog server or any combination of these options

- **[Options]** is where you define extra actions to be taken like sanitization, adding extra CEF fields, un/selecting specific CEF events, limiting file output size, and so on …

<Action>

--generate (-g)

Use this action when you want to generate new CEF events. These events are going to be created based on a customizable template file passed via the -input parameter.

--play (-p)

Use this action when you want to use existing CEF events as a source. These CEF events will be read from one or multiple files passed via the -input parameter and can be modified by CAKe according to your needs.

Examples:

- CAKe.py -g -n 100 -i /opt/mytemplate -o /opt/myCEFfile
  generates 100 CEF events per second and saves the output to myCEFfile. CEF events are formatted according to information provided in mytemplate

- CAKe.py -p -n 100 -i /opt/CEFfile1,/opt/CEFfile2 -o /opt/myCEFfile
  reads (play) 100 CEF events per second from CEFfile1 and CEFfile2 and sends the output to myCEFfile

<Action-type>

--realtime (-r)

This action-type is using near real-time timestamps with timestamps being spread equally overtime (timestamp used by default is 'rt', see --timetype option for more detail). For instance for an EPS (Events Per Second) of 1000, the first event will use the current time and there will be a difference of 0.001 second between two consecutive events. It's worth noticing that CAKe will resync with real time from time to time which will cause inconsistencies if the EPS is higher than what the system can deliver. EPS supported by a system can depend on multiple factors such as the HW used, the parameters used (for instance sanitizing is taking more time) or the output(s) chosen (syslog is usually slower than writing to file). During our tests we have been able to consistently reach 17000 eps for a single instance of CAKe on a standard workstation.

Use Case : This action-type can be used to simulate a real flow of events sent to a connector or a logger in real-time.

Examples:

- CAKe.py -g -r 5000 -i /opt/mytemplate -o /opt/myCEFfile
  - generates 5000 events per second and modifies the timestamp to spread these 5000 events equally over a second. The scripts will generates events at the same rate until CTRL-C is pressed

--keeptimestamp (-k)

This action-type keeps existing timestamps unchanged. If no timestamp exists in the original event, the script won't add any timestamp.

Use cases: Keeping existing timestamps can be useful when:

- you just want to play events as they are in the input file with no modification
- you have manually specified in the template the timestamps to be used when generating new CEF events and don't want them to be overwritten
- played or generated events contain no timestamp and we want to leave them like that in order to let the connector stamp them at the arrival time. This scenario is similar to --realtime but events are not spread equally over a second.

Examples:

- CAKe.py -g -k 5000 -i /opt/mytemplate -o /opt/myCEFfile
  - generates 5000 events per second and keeps the timestamp(s) (if any) defined in the template

--customtime (-c)

This action-type calculates the time interval between two consecutive events by splitting a user defined timeframe by a given number of events. For instance if a user choose 360000 events for a period of 1 hour, timestamp between two consecutive events will be 0.01 sec. Moreover the user can define the timeframe in the past or in the future if needed.

Accepted time formats:

- **EPOCH**: a ten digits serie starting with 1. For instance: 1378891029 (NB:13 digits format is not accepted as input but will be used in the output)

- **Relative time** :
  - Accepted variables: Now and Today (00:00:00 this morning)
  - Accepted variations:
    - m (minute)
    - h (hour)
    - d (day)
    - w (week)
    - M (month)
    - ex: Now-1h , Today-1M , Today+15m

- **Absolute time** :
  - the default format is %d/%m/%Y-%H-%M-%S but it can be modified with the timeformat variable in the "custom parameters" section of the script.
  - The column ":" sign is used as a separator and shouldn't be used in the format.
  - If the date format contains spaces, the whole time will have to be double-quoted to allow python to understand the argument properly. As a good practice we recommend always double quoting absolute time format.

- Different time types can be mixed together. For instance starting time could be in EPOCH time and end time in absolute format.

- Ending date must be posterior to starting date.

- Default timestamp modified is 'rt', see --timetype option for more details

Use cases: --customtime should be used when it's important to insert events at a precise moment in time :

- if events have to be inserted in the past → for instance if you want to populate quickly several days of events in an ESM DB for performance testing.
- if you want to generate a "flat" load over a given period of time. You could for instance decide that you want to have 1000 events per second for a full day. Having the number of events equally spread over a given period of time can be useful for performance testing.

Examples:

- CAKe.py -g -c Now-1h:Now:5000 -i /opt/mytemplate -o /opt/myCEFfile
    - generates 5000 events and spread the timestamp over a period of time ranging from Now-1h until Now.
- CAKe.py -g -c Now-1h:1523467890:5000 -i /opt/mytemplate -o /opt/myCEFfile
- CAKe.py -g -c “05/10/2013-14-57-00”:”05/10/2013-17-00-00”:1000 -i /opt/mytemplate -d

--notimestamp (-n)

This action-type removes the timestamp if it exists (‘rt’ removed by default, see –timetype option for more details)

Use cases: --notimestamp can be useful when

- events having an old timestamp need to be re-used.
- Some events need to be used multiple times. In this case they can be sent without timestamp and the connector will use the current time to create the timestamp

Example:

- CAKe.py -p -n 100 –i /opt/myCEF –d
    - reads 100 CEF events per second from myCEF file, removes rt timestamp and displays the result on screen

<Input>

Input needs to be specified for both --generate and --play <actions> but the type of input provided is different.

--input (-i) with --play (-p) action:

As we want to play existing CEF events, we must provide a file or a list of files containing those:

- You can specify as many input files as you want with the play action. CAKe will read files in the specified order
- if CAKe needs to read more CEF events than the sum of CEF events contained in all input files, it will restart from the first file and so on …
- CAKe will discard lines in the file which are not considered as CEF events. Valid format for CEF events can be changed in the "custom parameters" section of CAKe.py by modifying the regular expression in the validCEFevt variable.
- CAKe has been written to allow using huge input files without consuming too much memory.

Example:

- CAKe.py -p -c Now-1h:Now:5000 -i /tmp/myCEFevents1,/tmp/myCEFevents2 -o /tmp/myOutputFile
    - plays both files myCEFevents1 and myCEFevents2 in the specified order. If the sum of CEF events contained in both events is <5000, the script will continue to read events from the first file and so on…

--input (-i) with --generate (-g) action:

Here we want to generate CEF events based on a template file which contains the fields and values to be used in order to generate a CEF event:

- You can only specify a single template file as input for the generate action.
- Template files need to follow a predefined format, refer to the "Template for CEF generation" section for more details

Example:

- CAKe.py -g -r 5000 -i /tmp/myTemplate -o /tmp/myCEFfile
    - generates 5000 EPS according to the parameters defined in myTemplate file.

<Output>

Output indicates to CAKe what you want to do with the CEF events that you have generated or played. These CEF events can be stored file(s), converted in .CSV format, send via syslog, displayed on the screen or any combination of these options.

--outputfile (-o)

Stores your CEF events in a file. You can only specify one output file but it's possible to have a rotation mechanism when the file reaches a predefined size. See -l (--limitoutputsize) option for more details.

Example:

- CAKe.py -g -r 5000 -i /tmp/mytemplate -o /tmp/CEFoutput
    - generates 5000 EPS in realtime <action-type> and stores the output in /tmp/CEFsource

--csvfile (-v)

Parses your CEF events and stores them in CSV file. This can be convenient when you need to get human readable format for your CEF events.

- You can only specify one output file but it's possible to have a rotation mechanism when the file reaches a predefined size. See -l (--limitoutputsize) option for more details.
- The CSV file should be readable in Excel from any system whatever the separator defined in the regional settings. Should this be necessary, you can modify the delimiter in the Custom Parameters section of the script with the variable csvdelimiter.
- CEF Values are double quoted in order to allow those to contain the delimiter character. Should this be a problem, it is possible to modify the qualifier character in the Custom Parameters section with the variable csvqualifier
- If for some reason you want to remove all the delimiter values (like comma) from the CEF values stored in the CSV, you can replace them by a character of your choice. For this you need to set the variable allowdelimiterinvalue to false (true by default) and to define the replacement string in replacementcsvdelimiter
- CAKe creates first a temporary file before creating the final CSV file. This means that twice the file size of the final CSV file should be available on the disk.

Example:

- CAKe.py -g -r 5000 -i /tmp/template -c /tmp/CEFoutput.csv
    - generates 5000 EPS and stores the output in /tmp/CEFsource.csv with all values sorted and separated by a comma

--syslog (-s)

This output allows sending the CEF events to a syslog server (usually a syslog Arcsight connector). It should be noted that CEF events are sent as is, meaning there is no syslog header added.

Parameters to specify:

- A **destination** which can be
  - A hostname
  - An IPv4 address
- A **port**
- A **protocol** which can be:
  - **u** for udp
  - **t** for tcp
  - **ud** for udp delay. As UDP is unreliable by nature, there is a risk of packets/events loss when the number of events generated per second becomes high. The udp delay parameter allows preventing this by artificially creating a delay between two batches of syslog events. By default ud will add a 0.001 second delay every 5 events but you can change the default behavior by editing both variables udpdelay (delay in seconds) and delaybatchsize (number of events in a batch) in the "Custom Parameters" section of CAKe. udp delay should be seen as a way to limit packet loss when using high EPS in UDP but TCP should still be the preferred option when possible.

Examples:

- CAKe.py -p –k 5000 -i /tmp/input -s mysyslogsrv:514:u
  - plays 5000 CEF EPS from /tmp/input and send them to mysyslogserver on port 514/udp
- CAKe.py -g –n 5000 -i /tmp/template -o /tmp/output -s mysyslogserver:514:ud
  - Generates 5000 CEF EPS based on /tmp/template
  - removes the default timestamp ('rt') during generation
  - saves a copy of the generated CEF events in /tmp/output
  - sends the generated CEF events to mysyslogserver on port 514/udp but waits for 1msec every 5 events.

--display (-d)

This output displays the CEF events on the screen. The main interest is to provide an easy way to visualize the CEF events generated/manipulated by CAKe in order to verify they are formatted in the expected way. When you are satisfied with the result, you should disable this output as it will slow down the other outputs used (see multi output consideration section).

Multi output considerations

When multiple outputs are selected, they will all contain the same CEF events and the max EPS rate will never be faster than the slower output method. For instance, if you can reach 20000 eps with file output alone and if you can only reach 10000 eps with syslog ud option, you will not be able to get more than 10000 eps if you select both syslog ud and file output at the same time. The slowest output mechanism is the screen output followed by the csv file. Syslog and CEF file output should normally be pretty fast but some syslog parameters can heavily impact the speed (udp delay mode for instance).

Example:

- CAKe.py -p -k 10 -i ./input -s mysyslogsrv:514:t –o out.cef –v out.csv –d
    - read 10 CEF EPS from /tmp/input and
        - send them to mysyslogserver on port 514/tcp
        - save them in out.cef file
        - convert CEF to CSV format and save to out.csv
        - display output on the screen

[Options]

--force (-f)

This option can be used to bypass warning messages such as “the output file already exists, do you want to overwrite it ?”. This is helpful when automation is required via a script or if you don’t want to be bothered by warning messages. It can be seen as a “Stop bothering me, I know what I’m doing” so be sure you indeed know what you’re doing if you use it.

Example:

- CAKe.py -p -r 20000 -i /tmp/myCEF -o /tmp/mynewCEF -f
    - plays myCEF file at a 20000 EPS rate and let you overwrite /tmp/mynewCEF with no confirmation if it does already exist.

--timetype (-t)

By default all the action on timestamps only apply to the receiptTime (rt). It is possible to apply the changes to another timestamp or to multiple timestamps with this option. If you want to permanently change the timestamp modified by default by CAKe, you can go to the Custom Parameters section of CAKe.py and change the timelist variable.

Example:

- CAKe.py -p -r 1000 -i /tmp/myCEF -o /tmp/mynewCEF -t rt,art
    - plays 5000 CEF EPS from myCEF file and sets rt and art timestamps to real-time

--limitoutputsize (-l)

By default CAKe creates a single output file but it is possible to create a new file each time the predefined max size is reached.

- The file names are built like this “output_filename.current_time” .
- The time format used is the same than the one defined in the “custom parameters” with the timeformat variable.
- The size is specified in MB and the minimum size allowed is 1MB.
- It is possible to modify permanently CAKe default behavior by specifying a different value for the variable outputfilesize in the “Custom Parameters” section. By default there is no file size limit (0).
- This option only works with CEF file output (-o) but not with CSV file output (-v)

Example:

- CAKe.py -p -r 1000 -i ./myCEF -o ./mynewCEF -l 10
    - plays 1000 CEF EPS from myCEF file and creates a new output file called mynewCEF.$date each time the output file reaches a size of 10MB

--blend (-b)

If for some reason you need to have some kind of randomness when playing a CEF file, you can use this option to *partially* randomize the output.

- The randomization is only partial because it does not apply to the whole input file(s) at once but only to a certain number of CEF events at a time. By default the number of events is 5000.
- For instance you have an input file containing 10000 CEF events and you use the -b option with the default size value (5000). The first 5000 CEF events will be blended which means their order will be changed randomly but each event will only be used once in the output. The same process will happen again with the next 5000 events. This means that event number 100 in input file can become event 1-->5000 in the output file but will never become event 5001-->10000. This is why the randomization is only partial.
- The number of events which can be blended together can be modified in the “Custom Parameters” section with the variable maxCEFmem. However increasing this parameter will modify the number of events kept in memory before being sent to output file which increases the memory used

Example:

- CAKe.py -p -k 1000 -i ./myCEF -o ./mynewCEF -b
    - reads the maxCEFmem (5000 by default) first events from ./myCEF without modifying them but blends them together before saving them in ./mynewCEF. Then the operation is repeated with the next ‘maxCEFmem’ events and so on ...

--sanitize (-w)

It can be useful to sanitize CEF events for instance to remove sensitive information from real events you want to use outside of your production network. This option can be used in 6 different ways

- ip:rm (remove IPs)

All IPs values and the associated field names are removed from CEF events

NB: this option could also remove strings similar to IP addresses like product version number because there is no validation of the field name format.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF -o ./mynewCEF -w ip:rm

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1| cs1=CEF event

- ip:rnd (randomize IPs)

All IPs values are replaced by random IPs

The range of IPs being used for the random IP generation can be modified in the “Custom Parameters” section with the variables rndipmin and rndipmax. By default the ranges starts at 0.0.0.1 and ends at 223.255.255.255

NB: this option could also remove strings similar to IP addresses like product version number because there is no validation of the field name format.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF -o ./mynewCEF -w ip:rnd

mynewCEF could contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=112.0.1.72 dst=85.2.1.1 cs1=CEF event

- field:field1,field2

Can be used to remove one or several “field=value” pairs from CEF events based on the **field** name

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF -o ./mynewCEF -w field:src,cs1

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1| dst=4.3.2.1

- value:foo,bar

Can be used to remove one or several “field=value” pairs from CEF events based on the **value** name.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF -o ./mynewCEF -w value:CEF,dst

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1

- header:foo,bar

Can be used to remove one or several strings from the CEF **header**. Be cautious with this option, it could generate events violating the CEF standard

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CAKe event

CAKe.py -p -k 1 -i ./myCEF -o ./mynewCEF -w header:generator,CAKe

mynewCEF will contain:
CEF:0||CEF |1|100|warning|1.0|src=1.2.3.4 dst=4.3.2.1 cs1=CAKe event

- string:foo,bar

can be used to remove one or several strings from CEF events. This will remove the string(s) wherever they are located in the CEF event. Be cautious with this option, it could generate events violating the CEF standard. When possible it is safer to use another sanitization option.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF -w string:CEF

mynewCEF will contain:
:0|CAKe| generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1= event

--fix (-F)

Can be used to fix CEF events containing a small error or to replace a string by another. It will replace all occurrences of the string wherever it finds it in the event.

Example:

if myCEF contains:
CEF:0|CAKe|CAF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CAF event

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF -F CAF,CEF

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

--extract (-e)

Can be used to keep the CEF header and remove all the “field=value” pairs which haven’t been specified.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF -e dst,cs1

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1| dst=4.3.2.1 cs1=CEF event

--select (-S)

Can be used to only keep CEF events containing 1 or multiple patterns. When multiple patterns are provided, the **OR** operator is used. If you need to use a AND operator for multiple patterns, you can run the command multiple times with a single pattern.

NB: Because the select function is applied after the CEF events generation, the count of events does not reflect the number of events selected sent to the output but the count of events generated before the selection happens.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs2=ECF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs3=EFC event

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF -S EFC,ECF

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs2=ECF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs3=EFC event

--unselect (-U)

Can be used to discard CEF events containing 1 or multiple patterns. When multiple patterns are provided, the **AND NOT** operator is used which means only events containing all the specified patterns will be discarded.

NB: Because the unselect function is applied after the events generation, the count of events does not reflect the number of events sent to the output but the count of events generated before the selection happens

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs2=ECF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs3=EFC event

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF -U CEF,ECF

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs3=EFC event

--add (-A)

Can be used to add 1 or several extra “field=value” at the end of the CEF event.

Example:

if myCEF contains:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4

CAKe.py -p -k 1 -i ./myCEF –o ./mynewCEF –A “dst=4.3.2.1 cs1=CEF event”

mynewCEF will contain:
CEF:0|CAKe|CEF generator|1.0|100|warning|1|src=1.2.3.4 dst=4.3.2.1 cs1=CEF event

Template for CEF generation

The template file used to define parameters of the CEF events you can generate with the - -generate option is split in 3 sections for which following rules apply:

- Lines starting with == indicate sections start
    - These lines should not be modified
    - the sections order shouldn't be changed

Section1 : Global Parameters

Valid Section 1 example:

```
== SECTION 1 : Global parameters

separator:,
```

"separator" defines the separator to be used between multiple values. If missing this separator is set to comma.

Section2 : Header

This section contains the list of mandatory fields composing the CEF header.

- All these fields must be present in the template file but some can be empty
- Fields order must not be changed
- Unknown fields will be discarded

Valid Section 2 example:

```
== SECTION 2 : Header

cefversion:0
devicevendor:Check Point
deviceproduct:VPN-1 & FireWall-1
deviceversion:
signatureid:100,101
name:accept,drop,reject,decrypt
severity:2,3,5,4
```

Section3 : Optional CEF fields

This section contains the list of optional fields composing the CEF extension.

- You can use any field you want here as long as you follow the naming convention :
  - fieldname:value1,value2
- The field name will be created as defined here with no additional validation so ensure that:
  - field name is correct and/or exists
  - the data type (string,IPAddress,...) is respected
- If your value contains an equal "=" sign, you absolutely need to escape it with a backslash "\=" or the interpretation by the Arcsight CEF interpreter can lead to unexpected results

Valid Section 3 example:

```
== SECTION 3 : Optional CEF fields

eventId:[10000:50000],[60000:65000],30000
proto:TCP,UDP
categorySignificance:/Normal,/Informational,/Alert,/Warning,/Critical
spt:[1:65000]
dpt:22,23,80,443,8080,139,138,500,389,161,53,21,25,445,123,67,,135,137
_cefVer:0.1
src:[10.1.1.1:10.2.255.255],,172.16.1.1
dst:[10.1.1.1:10.2.255.255]
```

- It's possible to specify a *range of integers* from which a value will be selected randomly.
  - Syntax: [starting value:ending value].
  - Example: [1:500]

- It's possible to specify an *IPv4 range* from which a value will be selected randomly.
  - Syntax: [first IP from the range:last IP from the range]
  - Example: [1.0.0.0:1.255.255.255]

- You can use several ranges or/and mix them with regular values
  - Example: [1:500],,600,[700:1200]

- It is possible to define an *empty value* as a valid parameter.
  - Example: 1,,2 means 1 or empty or 2

- Frequency. It is important to understand that the random function applies on each of the comma separated values. For instance if your list of values looks like: **dpt:[1:4],6,[8:12]** , the value 6 will in average be chosen 33.33% of the time so this value will appear a lot more often than any other value. There is no integrated function in CAKe to let you increase/decrease the occurrence of a given value but there are some tricks which could be used. For instance, in our example, you could use:
  - [1:4],[1:4],[1:4],[1:4],6,[8:12],[8:12],[8:12],[8:12] → each allowed value should now have an equal chance to be chosen
  - [1:12] with the option --unselect spt=5,spt=7 . This would give an equal chance to any number between 1 and 12 to be chosen but if 5 or 7 are chosen, CAKe will discard these events.

Advanced Use Cases

CAKe is pretty versatile but it is not always possible to do everything with a single pass. Here are a few cases that you could meet where you will have to be a bit more creative when using the tool.

Generating realistic events

Generating realistic events can be a pretty difficult task and, depending on how realistic your events should be, you may have to use several templates and run several passes of CAKe. For instance, let's imagine you want to generate realistic Firewall events. If in your template you select proto=TCP,ICMP and dport=80,22 , you probably don't want your events with proto ICMP to have a destination port number.

Step1: define 2 template files

- 1 with proto=TCP and dport=80,22
- another one with proto=ICMP and no dport.

Step2: Use CAKe to generate one output file per template

- something like CAKe.py –g –n 10000 –i templatefile1 –o out1

Step3:

- use cat and shuf functions to concatenate and shuffle the content of your files (cat out1 out2 | shuf > final) and play the result with CAKe. (ex: CAKe.py –p –r 500 –i final –s syslogsrv:514:t)
- **OR** run 2 sessions of CAKe simultaneously, each of those playing a different file

Complex timestamps

It's possible to define multiple identical timestamps with the --timetype option but in some situation you might want to precisely define several different timestamps. In this case, the only solution is to run CAKe twice, once for each timestamp you want to define. For instance, let's imagine you want 5000 events to be generated with a receipt time corresponding to the last hour and the file modification time corresponding to receipt time – 1h

Step1: CAKe.py –g –c Now-1h:Now:5000 –i template –o tmpout

Step2: CAKe.py –p –c Now-2h:Now-1h:5000 –i tmpout –o finalfile –t fileModificationTime

Bugs, limitations, ongoing work, … all the stuff you won't like but you should know about

The script is provided as is with no warranty of any kind, use it at your own risk. Don't forget the author is not a professional coder, just an enthusiast who decided to share his work with the community. You are more than welcome to provide your feedback, your constructive criticism and even your feature requests. I'm not making any promise of any kind but I could take some time to improve CAKe in the future if I find the time and the will. That being said, for the price, I still believe CAKe is a bargain ;-)

Known bugs

- I'm not too sure how CAKe will behave with some of the options (-w value, -w field, --extract) if some CEF values contain an escaped equal sign (\=). I should definitely work on that at some point
- I'm not too sure how CAKe will behave with escaped pipe character (\|) contained in the header. I should also work on this at some point.
- Sometimes if you forget a mandatory parameter after a given option, the next option is passed as a parameter. I'm not too sure why this happens but it shouldn't be a big deal as long as you follow the syntax. Just be aware that it can happen.
- When using the select or unselect option, the count of events is incorrect. This happens because the selection/un-selection of events happens after they have been counted. I know how to fix this but it isn't straight forward and I couldn't find enough courage to fix it (yet?).
- Probably plenty of other bugs as CAKe has never been tested by anybody else than I. Feel free to shoot

Limitations

- If you generate your events in --realtime and if the EPS is higher than what your output can support, your timestamps won't be perfectly accurate. I didn't find any elegant solution to prevent this so just be aware this can happen.
- With --customtime (-c) device action , CurrentWeek and CurrentMonth variables are not accepted.
- With --customtime (-c) device action , relative timestamps can only contain 1 or 2 parts. For instance 'Now-1h' is ok while 'Now-1d+1h' is not.
- --blend option is not really mature and it would be nice to have a more powerful shuffling function.
- --limitoutputsize (-l) only works with CEF output file (-o) but not with CSV output file (-v). This is related to the fact that the –v output uses a temporary file before generating the final CSV file which makes impossible to know when the file reaches the limit size.
- CAKe has been coded and tested (more or less) on python 2.7.
- Did I mention I'm not a professional coder ;-) Don't be surprised to find some oddities, incongruities and non sense in the code.

Reporting a bug

If you want to report a bug, you can contact me at **cardinal_gaetan |at| yahoo.fr**

Please provide as many details as possible (version, the full command you launched, the error message you got, a clear explanation of the problem and if possible the template or source file you used as well as the resulting output.)

License

CAKe has been released under the MIT/X11 license which can be found hereunder. This provides a lot of flexibility on what can be done with the software. If you find the software useful and/or if you modify it, I would be pleased to hear from you and to get a copy of the modified version. Many thanks

The MIT License