Pandora

The flexible monitoring platform

Simon Patarin

1 Overview

1.1 Introduction

Pandora is a general purpose monitoring platform. It offers a high level of flexibility, while still achieving good performance. Pandora's architecture has been described in a research paper *Pandora, A Flexible Monitoring Platform* to which you can refer for further details.

Each monitoring task executed by Pandora is splitted into basic and self-contained building blocks called *components*. These components are chained inside *stacks* to constitute high level tasks. Stack execution consists of components exchanging messages — data structures called "packets" — from the beginning of the stack, to the end.

Pandora provides a framework dealing with (among others) packet demultiplexing, timers, threads and communication. This allows programmers to concentrate on the precise functionnalities they want to implement and pomotes code reuse. During stack execution, components are created as necessary and the resources they use are collected after some — user-specified — time of inactivity.

There exists three different types of components in Pandora: standard, input and output. They differ by their number of input and ouput flows and by small modifications in their API. Packets are the data structures manipulated by components. Indeed, components may be seen as packet flows operator in the extent that they accept packets of some type and produce packets of possibly some other type.

The whole design of Pandora tends to make it easy to write or adapt components and packet types that suit your needs best. This can involve the conception of brand new objects or simply modifying existing ones.

1.2 Building and Installing Pandora

Pandora is made of several libraries. The `libpandora` is the core of the system and is used in conjunction with the others libraries which contains component and packet definitions. The other libraries (those found under the '`pandora_components`' directory in the distribution) contain the components and packets used by Pandora. Those are grouped by field: e.g. all components related to HTTP monitoring are in the `libphttp` library, and the packets they use are in the `libphttp_pkt` library. Programs using Pandora needs to be linked with the `libpandora`.

Important Note: Do not make Pandora setuid root for it loads dynamic libraries without any check. This makes local (and possibly remote) exploits really trivial.

You need GNU `gcc` and `g++` compilers to build Pandora. Provided those, basic installation process should be quite straight forward. Just uncompress the archive and type inside the '`pandora-x.y`' directory:

```
> ./configure
> make
> make install
```

Unfortunately, I do not have enough distinct platforms (especially not yours!) to guarantee that this will always work. See the '`README`' file for a list of platforms on which Pandora is known to build successfully.

You may pass several switches to the `./configure` command to customise the way Pandora is compiled. The most useful are:

`--disable-debug`
: suppress most debug checks, resulting in smaller a faster code

`--disable-shared`
: build static component libraries *instead of* shared ones

`--enable-static`
: build static component libraries *in addition to* shared ones

`--without-thread`
: remove any threading support from Pandora, resulting in much faster code (if you do not need this feature!)

Other options are available, see `./configure --help` for a complete list.

2 Running Pandora

2.1 Configuration

To configure Pandora, one needs to tell it the definitions of the stacks it will use. Additionally, the symbols that may be used by Pandora (in particular component and packet names) and the dynamic libraries from which they can be loaded should be also provided. By default, Pandora will look for these two files in the (environment variable) `PANDORA_HOME` directory with names, respectively, '`stacks`' and '`libconfig`'. However, you can specify any other location by using the '`-f`' and '`-l`' options, respectively, at run time.

2.1.1 Configuration File

The configuration file defines the set of stacks that Pandora will know about upon startup. Those definitions are written in a configuration language whose simplified specification in BNF form is the following:

```
stack          ::= '%'name '{' component+ '}'
component      ::= ('@'name ('[' options+ ']')? | '&'name ) modificator?
option         ::= '$'name '=' option_value
option_value   ::= (['"].*['"] | [+-][0-9]+('.'[0-9]*)? | true | false)
modificator    ::= ('(' | '|' | ')' | '<' | '>' )
name           ::= [0-9a-zA-Z_]+
```

Each stack must be given a name which is used to manipulate it during the execution of Pandora. A stack is composed of a number of components. Standard components are identified with their name. The modificator located after a component is used to specify the configuration of switch and demux components (cf. Components).

Options allow to set the parameters exported by the component mentioned in the stack definition. Values can be of numeric, boolean or string type. String type values may be processed by a function defined by the component developer.

2.1.2 Bindings File

The bindings file is generated automatically a compile time by a Perl script. It contains ressource descriptions (one by line). These may be of two types: library or symbol.

Comment lines start with a number sign (`#`) and extend up to the end of the line. No comment should appear in the middle of a line.

2.1.3 Complete Example

This is an example of a configuration file that defines an IP reassembly stack:

```
%defrag {
  @pcap [$filter = "ip"]
  @ipfragswitch (
    @demux2 [ $algo = "ipquad" ] <
      @ipreass [$timeout = 30]
    >
```

```
    )
    @printer ;
}
```

First, packets are captured via the the `libpcap` library, encapsulated into a `pcap` component. They are passed to `ipfragswitch` component that looks at the DF flag inside the IP header to determine whether a packet is fragmented or not. Fragmented packets are pushed inside the substack (the components enclosed in parenthesis), others are passed directly to the last `printer` component that just prints a trace on the console for each received packet. IP fragments are demultiplexed in the substack according to their source and destination addresses, their IP protocol number and their fragment identifier thanks to the `demux2` component. After this stage, every fragments of each packets (and only them) are passed to an `ipreass` component, that reassembles all of them to create the complete IP packet. Those reassembled packets are then passed to the next component which happens to be the same `printer` component as above.

This stack has been called "defrag" and three options have been set up. The first concerns the `pcap` component and tells it to filter out every non IP packets. The second one is a symbol, defining the function that should be used to demultiplex packets. The last one is a timeout that is used to clean `ipreass` components up if they have not finished the reassembly 30 seconds after the first fragment was received.

A minimum binding file needed to use this stack should contain these entries:

```
[library]       dir1/libpip.so.0        0
[library]       dir2/libpip_pkt.so.0    0
[library]       dir3/libpcore.so.0      0
[packet]        IPPacket                libpip_pkt.so. 0
[component]     PcapHandlerComponent    libpip.so.0
[component]     IPFragSwitchComponent   libpip.so.0
[component]     IPReassComponent        libpip.so.0
[component]     PrinterComponent        libpcore.so.0
```

2.2 Execution

2.2.1 Running Pandora

Once both configuration files have been set up, one can run Pandora in a console. The basic usage is simply to specify all stacks that should be run as arguments in the command line; to run the `htt` and `dns` stacks, one would type: '`pandora http dns`'.

By default, each stack to be run is executed in its own thread, the main thread being only waiting for those to finish. Yet, one can use the '`-s`' option to disable this behaviour. In such case each stack is run sequentially in the main thread.

The `SIGINT` (⟨CTRL⟩ *C*) signal is caught by Pandora. in order to let it exit cleanly. However, this happens to fail sometimes and you should use `SIGQUIT` (⟨CTRL⟩ \) to exit from Pandora. If you do not like this, you can disable this behaviour by using the '`-c`' option.

2.2.2 Usage

General Syntax

```
pandora [OPTION]... [STACK]...
```

Options

`-a, --assign=OPDEF`
: set option as specified in OPDEF, with OPDEF: stack.comp.option=value

`-c, --no-sighandler`
: do not install SIGINT handler

`-d, --dump-stacks`
: dump stack definitions, as known by Pandora at startup; if repeated, (‘`-dd`’) Pandora will also try to locate and lookup each component name

`-D, --dump-libraries`
: dump dynamic library config, as known by Pandora at startup

`-e, --exec=STRING`
: execute the command in Guile mode

`-f, --stack-uri=URI`
: read stack definitions from URI, recognized schemes for the URI are: `http`, `file` and `string`; if no scheme is specified, `file` is the default

`-h, --help`
: show help message

`-l, --library-uri=URI`
: read library config from URI, recognized schemes are the same as those for the stack URI

`-o, --get-options=COMP`
: display options and their default values for component COMP

`-s, --no-thread`
: do not create additional threads: this allows to run stacks sequentially or to let the control to the main thread when running a single stack

`-S, --print-stats`
: print statistics about components and packets creation and deletion

`-t, --profile`
: print stack execution time (this does not give intersting results unless -s is also specified)

`-u, --syslog`
: log messages with syslog facility

`-v, --verbose`
: increase verbosity level (may be repeated)

`-V, --version`
: print version information and exit

2.2.3 Usage

Pandora may be entirely controlled with Guile (`http://www.gnu.org/software/guile/`) (version 1.4 or higher). You may either run an interactive Guile interpreter from Pandora, or load Pandora from the standard Guile interpreter. Here is the syntax of the commands to use in each case:

- Pandora stack

```
> ./pandora -s guile
pandora>
```

- standard Guile

```
> guile
guile> (use-modules (pandora))
guile>
```

Inside the Guile interpreter one may use Pandora specific primitives in conjunction with every standard Guile ones. Each of these primitives starts with a `pandora-` prefix. See Appendix A [Pandora Guile Primitives], page 21, for a complete list of all available primitives.

3 Extending Pandora

Important Note: Since the version 0.3 of Pandora (to which this section refers), Pandora's API has evolved considerably. Hence, you should not take this into consideration for current (CVS) versions of the software. I will update it to reflect changes that have happened as soon as I find the time to do it. In the meantime the essential files you may look at to extend Pandora are: '`libpandora/component.h`', '`libpandora/inputcomponent.h`' and '`libpandora/packet.h`'.

3.1 Writing a New Component

Pandora provides a `Component` class that defines the component API and performs all the common functions they need. Each component type is represented by a class deriving from the `Component` class which implements the treatments specific to their type. Namely, there are the following four classes: `SimpleComponent`, `InputComponent`, `SwitchComponent` and `DemuxComponent`. Next every component used is defined as a class deriving from one of the above `Component` subclasses.

In this section, we will describe the `Component` class and all its subclasses, in order to show how to write one's own component.

3.1.1 The Component Class

Components are created and deleted by Pandora (specifically by one `Dispatcher`). For the sake of genericity, components constructors may not use arguments. Copy constructors are not needed since Pandora never duplicates components (it always create new ones or use pointers). Besides, every component should define and export a name that will be used for the stack definition in the configuration file. This achieved through two macros:

`INIT_COMPONENT(`*class*`,` *name*`,` *in_type*`,` *out_type*`)`
: This macro has to be placed inside the class definition (typically in the '`.h`' file). The *class* argument is the component class name used; *name* is the exported name which should be enclosed between double quotes. The two other arguments (*in_type* and *out_type*) specify which kind of packets this component is likely to accept and to produce, respectively. This is used to perform few basic sanity checks. See Section 3.2 [Writing a New Packet], page 11, to know how packet types are defined. In a first step, one can use the '`PKT_ANY`' type to bypass these checks.

`EXPORT_COMPONENT(`*class*`,` *name*`)`
: This macro must have a global scope and is usually located in the '`.cc`' file used to implement the class. In cases when all the class is implemented within its declaration, you must still create such a '`.cc`' file to insert this macro, otherwise the class will not be accessible to Pandora. The arguments used must be the same as the first two of the previous macro.

Basically, a component performs three tasks: accepting packets from previous components, pushing packets to next components and cleaning up itself when it is about to be deleted. These different actions are implemented in the following methods:

`bool add(Packet *)`
: This is the method called to pass a packet to the component. The argument is a pointer to a `Packet` which must be downcasted to the appropriate expected packet type before being used.

`void push(Component *, Packet *)`
: This is the most generic form of the method used to push a packet to a component. It cannot be overridden and should only be used by demux components. Other direct `Component` subclasses provide simplified methods which do not require to explicit the component to which the packet is pushed.

`void cleanup(void)`
: The `cleanup` method is called whenever the component is going to be deleted. At this point, the component should flush any packet it may still hold and reset itself so that it could be reused — we will come back on this point later.

`bool notify(Component *)`
: It is sometimes useful for a component to know when (one of) its successor(s) is cleaned up. In such cases, it has to override the `notify` method of the `Component` class. This method is called for the predecessor of a component going to be deleted. The argument passed is a pointer to the component about to be destroyed. If the return value of the function is set to a `true` boolean value, the destruction of the component will be *cancelled*.

Components also have the possibility to register *options* that will be set at run time via the configuration file. Options are standard class fields which can be of any type. Indeed Pandora treats options of type '`int`', '`float`', '`bool`', '`char *`' and '`void *`'. Registration must be performed inside the constructor with one of the following macros:

`registerOption(`*name*`, `*var*`)`
: This macro registers an option of name *name* whose value will be stored into the class fields *var*. The *name* should *not* be quoted, since this is done by the macro itself. Similarly *var* should be the plain name of the field (and not be a pointer to it).

`registerOptionPost(`*name*`, `*var*`, `*post*`)`
: With this flavor of the macro, the function *post* will be called immediately after the option has been set (which is the very last step of component initialization, meaning that everything except other options will have been setup previously). The prototype for the function is the following: `void post(void *)`.

`registerOptionFunc(`*name*`, `*var*`, `*func*`)`
: This macro is meant to deal with '`void *`' options. In such cases, the option value specified in the configuration file should be a string. This string is then passed as an argument of the *func* function, together with a pointer to the specified class field. Then the prototype of the function is `void func(char *, void *)`. It is the responsibility of the function to perform the affectation of the option.

`registerOptionFuncPost(`*name*`, `*var*`, `*func*`, `*post*`)`
: Finally, this macro combines the behavior of the two previous ones.

`void push(Packet *)`
: This method passes a packet to the next component in the stack, as defined in the configuration file.

The `Component` class defines one option which uses the integer *timeout* field. When set to a positive value, this represents the amount of inactivity time (in seconds) after which a component is collected. In this context, "inactivity" for a component means it does not receive any packet.

For the sake of performance, Pandora does not actually delete collected components. They are stored in a pool of unused components which may be recycled (reused) in a different context. This is why the `cleanup` method must perform a complete reset of the persistent state of the component. However, it is not necessary to restore the original values of the options if they have been altered during the life of the component since this is done by Pandora. If this is not achievable (or suitable) the component must set the *cacheable* flag to a '`false`' logical value inside its constructor.

3.1.2 Input Components

Input components are the components used at the beginning of a stack. Pandora is able to run several stacks concurrently. In such cases, a new thread is started for each of those stacks. Respectively, new threads can only be created along with stacks. Therefore, the `InputComponent` class is where threads are handled. It defines the following methods:

`void push(Packet *)`
: This method is very similar to the standard component one. It only produces some additional side effects needed by the timer services provided by Pandora.

`bool add(Packet *)`
: Input components are not likely to receive packets from other components. That why it defines an `add` method that will provoke an error when it is called. Nonetheless, input components may override this method if they need it.

`void start(void)`
: This pure virtual method must be implemented by every input component. This is the method called by Pandora to begin the execution of the stack. When the methods returns, the whole stack is cleaned up.

`void stop(void)`
: As stacks may be interactively started they may be stopped the same way. A default implementation is provided which just cancels the thread running the stack. Yet, the stack might remain in a inconsistent state and this should be avoided in most (if not all) cases, since it can make Pandora crash if the stack is restarted sometimes later.

By default, input components are set to be non cacheable because threads handling prevents them to be cached. Furthermore, it is very unlikely that they need to be collected.

3.1.3 Switch Components

Switch components are used to create branches in the stack execution path (for example to handle differently IP packets according to the transport protocol packet they encapsulate). These substack are eventually merged after specific processing is finished: one may

think of two branches that extract DNS information either from TCP or UDP packets, once this is done DNS packets can be further processed indistinctively. In switch components, the *next component* is the previously mentioned merging component (if it exists). The methods implemented by the `SwitchComponent` class are the following:

`void push(Packet *)`
: As for the previous component types, this method passes packet to the "next component" in the stack. With the above definition, this means that packets pushed with methods bypass all the branches defined. In particular, this can be used to introduce shortcuts in the execution path.

`void pushInside(Packet *, int)`
: This is the method used to push packets into specific branches. One just needs to give the branch number (starting from 0) as the second parameter of the function.

`void cleanup(void)`
: This method allows the cleaning process to be propagated to all the branches. Therefore, it is mandatory to call this method (via `SwitchComponent::cleanup()`) at the end of any overridden method in subclasses.

To ease the writing of switch components, Pandora provides a generic switch implementation: `GenSwitchComponent`, that provides the following interface:

`bool add(Packet *)`
: Pushes packets to the right branch according to the `_switch` result. This behaviour cannot be overriden, if something else is expected, one should use directly the `SwitchComponent` class.

`int _switch(Packet *) = 0`
: This is used to determine in which branch incoming packets should be pushed. If the return value is negative, the packet is discarded, if it is 0, it is pushed to the next component, else it goes to the specified branch number minus 1.

3.1.4 Demux Components

Demux components are basically used to group packets sharing a common characteristic. In other words, packets to be demultiplexed have a hash value and every packets that have the same hash value belong to the same group. Each time a new group appears, a fresh substack is dynamically created to which all packets of the group are pushed. Such substacks may be destroyed as they are not used anymore.

The `DemuxComponent` class does not export any API. Yet, in order to allow an easy implementation of packet demultiplexing, Pandora provides generic demux component classes. These classes make use of *packet keys*. A packet key represents the hash value discussed above and those generic demultiplexers make every incoming packets fill such a key. There are as many demultiplexer classes as distinct packet keys: they are simple instantiation of a template class. Finally, in order to be demultiplexed a packet needs only to define the method that will fill the key. For that purpose it must override the virtual functions named `fill*Key`, where the wild card is the actual name of the key used (see Section 3.2

[Writing a New Packet], page 11). Since one may have different demultiplexing schemes for the same kind of key, those methods take an additional integer parameters indicating which algorithm to use. This number is set via the generic demultiplexer classes option *algo*.

The existing keys are the following :

`SingleKey`
: It is made of a single long integer field, named *k1*.

`PairKey`
: It is made of two long integer fields, named *k1* and *k2* .

`QuadKey`
: It is made of four long integer fields, named *k1*, *k2*, *k3* and *k4*.

`StringKey`
: It is made of a single `String` field named *str*.

3.2 Writing a New Packet

Every packet used in Pandora should derive from the `Packet` class. Unlike components, packets are likely to be "copied" during their life cycle, therefore copy constructors and assignment operator should be implemented. Besides, each packet type must have an unique identifier (currently a 32-bit integer).This identifier (and other utilities) must be declared with the following macro:

`INIT_PACKET(`*class*`, `*id*`)`
: This macro has to be placed inside the class definition, in a public section. The first parameter refers to the name of class, whereas the second one is the identifier to be used.

Every packet identifier defined so far are in the '`Packet.h`' file with `#define` directives. It is not mandatory to do so for new identifiers, and should change in a near future.

The `Packet` class defines the following methods:

`Packet *clone(void)`
: This method returns a freshly allocated copy of the packet. Since it is a virtual method, it can be called on a `Packet *` pointer and still returns the right object.

`void print(ostream *)`
`void log(void)`
: These methods are used for printing packets in an ASCII format. The `log` method is only a shortcut for `print(`*&cout*`)`.

`size_t write(char *, size_t)`
`size_t read(const char *)`
: Packets that may be passed to other components via the network should implement these methods for marshalling/unmarshalling. Since Pandora stacks may be run on machines of different architecture and still communicate, these functions should be independent of the underlying byte-order and data type sizes.

`size_t write_fast(char *, size_t)`
`size_t read_fast(const char *)`

Since the portability required for the above methods may be time consuming, one has the possibility to write faster non-portable marshalling functions. These are meant to be used for stacks running on the same kind of machines.

`void fillSingleKey(SingleKey *, const int)`
`void fillPairKey(PairKey *, const int)`
`void fillQuadKey(QuadKey *, const int)`
`void fillStringKey(StringKey *, const int)`

These are the methods used in the generic demux component classes to produce suitable "hash" values.

4 Component Reference

In this section we describe (almost) all the component classes that have been implemented so far. For each of them, we give the name under which it can be accessed in a stack definition (or a dash if it cannot), a brief description of their use and a list of the options they may accept.

4.1 Core

4.1.1 Base Classes

`Component` `(-)`

Base class for every components. Options it exports are available for any component.

- `timeout` (int) — Sets inactivity timeout (in seconds): if no packets are passed to the component after the specified amount of time, the component is automatically cleaned up.
- `cache` (bool) — If set to `false`, prevents a component from being reused (i.e. when cleaned up, it is not cached).

`SimpleComponent` `(-)`

Base class for "Simple" components (i.e. one input and one output).

`InputComponent` `(-)`

Base class for "Input" components (i.e. simple components suitable at the first position of a stack).

- `nothread` (bool) — If set to `true`, forces the stack to be started in non threaded mode.

`SwitchComponent` `(-)`

Base class for "Switch" components.

`DemuxComponent` `(-)`

Base class for "Demux" components.

4.1.2 Inputs

`ReaderComponent` `(reader)`

Reads Pandora-crafted packets either from a file or from the network (see also WriterComponent).

- `file` (string) — Name of the file from which packets are to be read.
- `listen` (int) — Port number on which a listening socket is to be opened. If this number is positive, starts a TCP server, if this number is negative, starts an UDP server.

`PipeInputComponent` `(pipe)`

Stores packets it receives (via its `add` method) in a mutex protected FIFO while pushing them to its successor. This component may be used to split stack execution into several threads, or to merge the outputs of two stacks running in different threads (see also SynchroComponent for the latter use).

4.1.3 Switches

`GenSwitchComponent` `(-)`

Base class to ease the writing of Switch components.

`DupSwitchComponent` `(dup)`

Replicates every packets it receives to each of its substacks.

- `num` (int) — Number of substacks to which packets are to be replicated.

4.1.4 Demuxes

`GenDemuxComponent` `(-)`

Base class for generic Demux components.

- `algo` (int) — Index of the algorithm to use for demultiplexing packets.

`SingleDemuxComponent` `(dmx1)`

Demultiplexes packets according to a `SingleKey`.

`PairDemuxComponent` `(dmx2)`

Demultiplexes packets according to a `PairKey`.

`QuadDemuxComponent` `(dmx4)`

Demultiplexes packets according to a `QuadKey`.

4.1.5 Outputs

`DiscardComponent` `(discard)`

Discards any packet it receives.

`LinkerComponent` `(linker)`

Pushes packets to another running stack (inside the same instance of Pandora).

- `stack` (string) — Name of the stack to which packets are to be passed.

`PrinterComponent` `(printer)`

Prints packets in ASCII format in a terminal or a file, then destroy them.

- `file` (string) — Name of the file to which packet trace is to be printed.

`WriterComponent` `(writer)`

Writes packets in binary format into a file or on the network, then destroy them.

- `file` (string) — Name of the file to which packets are to be written.
- `host` (string) — Name of the host to which packets are to be sent, the format of string should be the following: '`hostname port`'. As for the ReaderComponent, if the port number is positive, a TCP connection is initiated, else if the port number is negative, UDP datagrams are sent.
- `fast` (bool) — Determines whether the `fast` Packet marshalling API is to be used, or not.

`prefix` (string) Triggers the use of rotating packets log file. All file names will be prefixed by this string, followed by the name of the host on which the capture took place and a time stamp for the moment at which it begun. You need to use one of `interval` or `size` options (or both) to specify when files are rotated.

`interval` (int) Rotates files after the specified amount of seconds.

`size` (int) Rotates files before they become bigger than the specified amount of bytes.

4.1.6 Utils

`CountComponent` `(count)`

Counts packets it receives and prints a summary when cleaned up.

`ReorderComponent` `(order)`

Reorders packets it sees according to their time stamp, then pushes them in that order.

`size` (int) Sets the size (in packets) of the queue in which packets are reordered. Packets are not pushed until this queue is filled up or the component is cleaned up.

`warn` (bool) Prints a warning when unordered packet is being sent (because the queue was too small).

`SynchroComponent` `(synchro)`

Locks with a mutex the operations of all its successors.

`index` (int) Index of the mutex to use among the `NB_MUTEXES` available (4 by default). Index ranges from 0 to `NB_MUTEXES`-1.

`TraceComponent` `(trace)`

Prints an ASCII trace of each packet and pushes them unchanged to the next component.

`tag` (string) Prefixes each packet trace with the string (to differentiate the outputs of several `trace` components within the same stack).

`life` (bool) Prints information about the number of packets/components created/active along with each trace.

`ts` (bool) Prints the time stamp at which the packet was received by the component (different from the packet time stamp) along with each packet trace.

4.2 Network Monitoring

4.2.1 Packet Capture

`PcapHandler` `(pcap)`

Encapsulates the `libpcap` library.

`filter` (string) Sets the packet filter, according to `tcpdump`'s syntax.

`device` (string) Sets the name of the interface on which are to be captured.

`rfile` (string) Sets the name of file (in `libpcap`'s dump format) from which packets are to be read.

`snaplen` (int) Sets the maximum length (in bytes) of snarfed packets. Packets exceeding this size are truncated.

`count` (int) Sets the number of packets to capture before exiting.

4.2.2 IP

`IPFragSwitch` `(ipfragswitch)`

Switches between fragmented and non-fragmented IP packets. Fragments are sent in the substack, while others skip it completely.

`IPReassComponent` `(ipreass)`

Reassembles IP fragments belonging to the same IP packet.

`IPProtoSwitch` `(ipprotoswitch)`

Switches IP packets according to their IP protocol number. Currently only TCP, UDP and ICMP are "switchable", packets from other protocols are silently discarded.

`tcp` (int) Index (starting from 1) of the branch to which IP packets encapsulating TCP are to be pushed. An index of 0 makes such packets discarded.

`udp` (int) Index (starting from 1) of the branch to which IP packets encapsulating UDP are to be pushed. An index of 0 makes such packets discarded.

`icmp` (int) Index (starting from 1) of the branch to which IP packets encapsulating ICMP are to be pushed. An index of 0 makes such packets discarded.

4.2.3 TCP & UDP

`TCPScanComponent` `(tscan)`

Extracts TCP packets from IP packets. IP packets that do not encapsulate TCP are silently discarded.

`UDPScanComponent` `(uscan)`

Extracts UDP packets from IP packets. IP packets that do not encapsulate UDP are silently discarded.

`TCPReorderComponent` `(tcporder)`

Resequences TCP packets belonging to the same connection (same source/destination addresses/ports 4-uple) according to their sequence numbers.

`TCPPortDemux` `(tpdmx)`

Demultiplexes TCP packets belonging to the same "transaction" (same source/destination addresses/ports couples, but possibly switched) according to their direction (e.g. from client to server and from server to client).

`TCPReqFlowComponent` `(trqf)`

Transforms TCP packets into (possibly oversized) UDP packets for protocols (like DNS or RPC) that usually use UDP but may initiate large data transfers with TCP. This requires that each transfer is prefixed by a 2-byte integer indicating its length (this is what the above protocols actually do).

`snaplen` (int) Maximum size (in bytes) of encapsulated protocols "interesting" headers.

4.2.4 Cache & HTTP

`HTTPScanComponent` `(nhscan)`

Analyses HTTP headers inside a (resequenced) TCP flow. Supports most of HTTP/1.1 features (keep-alive, pipeline).

`mime` (string) Name of the file where MIME types can be found.

`enc` (string) Name of the file where encoding schemes can be found.

`HTTPMatchComponent` `(hmatch)`

Matches HTTP requests with their corresponding responses within the same (keep-alive) "transaction".

`HTTPAnonComponent` `(anon)`

Anonymizes HTTP packets. **Warning** This component is going to be split in near future into two parts: the first one in charge of anonymizing generic IP packets and the other one in charge HTTP specificity (like URL hashing).

`key` (string) Sets the key with which every MD5 hash will be performed so that it is not easy to reverse when the start set is small (typically IP addresses).

`ipreject` (string) Name of the file where IP addresses not be logged can be found.

`depth` (int) Number of URL path elements to preserve while hashing. The resulting hash will of size: depth*16 bytes.

`level` (int) Anonymnization level to be performed: IP only, URL only, or both.

`URLDemux` `(udmx)`

Demultiplexes HTTP transactions according to their URL.

`CacheMatchComponent` `(umatch)`

Matches HTTP transactions seen before and after a proxy cache.

`addr` (string) IP address of the proxy cache.

`port` (int) Port number on which the proxy cache is accepting connections.

`SiblingMatchComponent` `(sibmatch)`

Identifies and merges cache transactions between cooperating proxy caches.

`CacheEvalScanComponen` `(cescan)`

Extracts from observed cache transaction the data suited for its evaluation.

`CacheEvalComponent` `(ceval)`

Computes and summarizes the evaluation of a proxy, based on collected data.

4.2.5 Basic Protocols

`ICMPScanComponent` `(icmpscan)`

Extracts ICMP packets from IP packets. IP packets that do not encapsulate ICMP are silently discarded.

`ICMPMatchComponent` `(icmmatch)`

Matches ICMP requests and responses according to their type and identifier.

`DNSScanComponent` `(dscan)`

Extracts DNS packets from UDP packets.

`DNSMatchComponent` `(dmatch)`

Matches DNS queries and answers according to their identifier.

`RPCScanComponent` `(rscan)`

Extracts RPC packets from UDP packets.

`RPCMatchComponent` `(rmatch)`

Matches RPC requests and responses according to their identifier.

`ICPMatchComponent` `(imatch)`

Extracts ICP packets from UDP packets.

`ICPScanComponent` `(iscan)`

Matches ICP queries and replies according to their identifier.

5 Available Packet Types

Here is the list of the classes implementing the various packet types used by components.

`IPPacket` IP packet.

`TCPPacket`
: TCP packet.

`UDPPacket`
: UDP packet.

`ICMPPacket`
: ICMP packet.

`ICMPTransPacket`
: ICMP transaction packet: merged request and corresponding response with transaction specific data (like round trip time).

`HTTPPacket`
: HTTP request or response.

`HTTPTransPacket`
: HTTP transaction packet: merged request and corresponding response.

`CacheTransPacket`
: Cache transaction packet: merged HTTP matching transactions before and after the proxy cache, with matching specific data (like the nature of the "transaction": hit or miss).

`CacheEvalPacket`
: Packet-like container for cache evaluation specific data.

`DNSPacket`
: DNS query or answer.

`DNSTransPacket`
: DNS transaction packet: merged query and corresponding answer.

`RPCPacket`
: RPC request or reply.

`RPCTransPacket`
: RPC transaction packet: merged request and corresponding reply.

`ICPPacket`
: ICP query or reply.

`ICPTransPacket`
: ICP transaction packet: merged query and corresponding reply.

`ValuePacket`
: Packet-like container for simple data types: integer, floats, booleans or strings.

6 Future Work

There is still a lot work to be done with Pandora. Items I can think of includes:

Documentation

Test suite

Deployment support

Dynamic component loading model refinement

Macro components

Fault tolerance

Load balancing

Stack structure cleanup

Configuration language improvement

Security

“Packet” types flexibility

More introspection

Automatic/semi-automatic stack building and checking

Tools easing dynamic configuration

GUI

Generalizing stacks to arbitrary graphs

Develop new components

Appendix A Pandora Guile Primitives

Appendix B Legal Notices

B.0.1

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public

License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further

restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICA-

BLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

B.0.2

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents