5Greplay User Manual

Montimage

Edgardo Montes de Oca
contact@montimage.com

5Greplay version 0.0.1
MMT-SDK version 1.7.0

26/07/2021

This user manual presents and explains how to use 5Greplay developed by Montimage's research team. The manual is suitable for basic users to use 5Greplay, developers' version is in progress.

For more information please contact us at `contact@montimage.com`.

Contents

1 Introduction

5Greplay is a 5G network traffic fuzzer that enables the evaluation of 5G components by replaying and modifying 5G network traffic by creating and injecting network scenarios into a target that can be a 5G core service (e.g., AMF, SMF) or a RAN network (e.g., gNodeB). The tool provides the ability to alter network packets online or offline in both control and data planes in a very flexible manner.

5Greplay relies in Montimage's Deep Packet Inspection Library **MMT-SDK**. MMT-SDK or MMT-DPI a C library that analyses network traffic using Deep Packet/Flow Inspection (DPI/DFI) techniques in order to extract hundreds of network and application based events[1] , including: protocols field values, data in messages and logs, network and application QoS parameters and KPIs.

This document is divided into 6 sections:

- Section 1 includes this introduction, the tool usage context, a description of the tool's global architecture and its main features.
- Section 2 explains how to install 5Greplay.
- Section 3 explains how to configure 5Greplay.
- Section 3 explains how to write and compile 5Greplay rules.
- Section 5 gives a roadmap for future development of the tool.

1.1 5Greplay Architecture

Global architecture: A high level architecture of the 5Greplay's frameworks is represented by Figure 1.

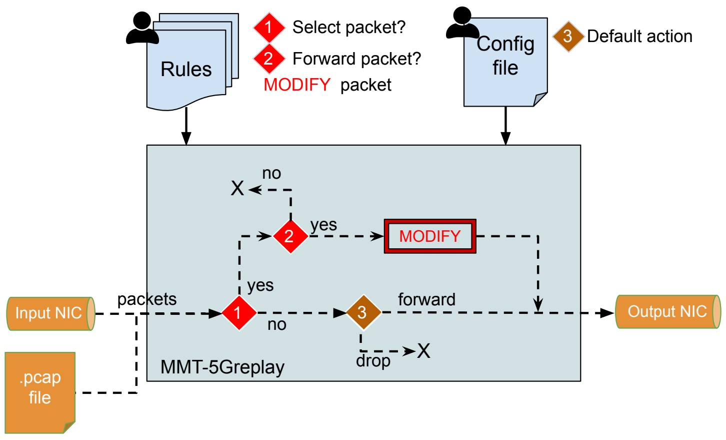
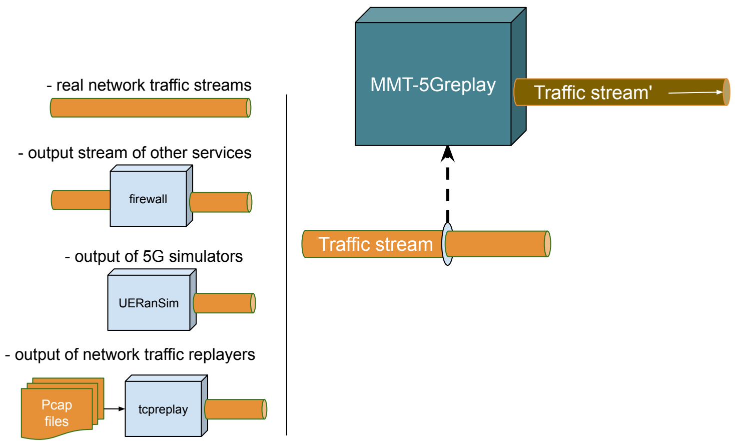


Figure 1: 5Greplay Architecture

5Greplay allows forwarding network packets from one network interface card (NIC) to another NIC with or without modification. It can be considered as a one-way bridge between the input NIC and the output one. Its behavior is controlled by (i) user defined rules and completed

[1] Note that the terms: "packet", "message", "log entry" and "event" can be considered as equivalent in this document. Here we will use either the term "packet" or "event" to denote an event in time that is perceived as structured data that can be extracted, classified and analysed.



- real network traffic streams
- output stream of other services
firewall
- output of 5G simulators
UERanSim
- output of network traffic replayers
Pcap files
tcpreplay
MMT-5Greplay
Traffic stream'
Traffic stream

Figure 2: 5Greplay in Service chaining

by (ii) a configuration file. The user defined rules decide explicitly which packet can be passed through the bridge and how a packet can be modified in the bridge. The config file provides a default option applying on the packets that are not selected by the rules, i.e., 5Greplay forward or drop these packets. The architecture of 5Greplay is represented in the Figure 1.

Workflow:

When a packet arrives at the input NIC, 5Greplay checks whether it is satisfied by one of the rules. If it is satisfied by no rules, 5Greplay applies the default action that is defined in the config file. The default action can be either (i) forwarding the packet to the output NIC without any modification, or (ii) dropping the packet. Otherwise, once the packet is selected by a rule, 5Greplay will apply the action defined in the rule on this packet. The action can be also either forward or drop. If the action is forward, then 5Greplay performs the modification action defined in the rule onto the packet before sending it to the output NIC.

1.2 Features

Main features: The main features provided by 5Greplay are:

1. *Granular traffic analysis capabilities*: MMT-SDK allows parsing a wide range of protocols and applications and to extract various network and application based traffic performance indicators. The extraction is powered by a plugin architecture for the addition of new protocols and applications.
2. *Application classification*: Prior to extracting protocol or application attributes, MMT-SDK uses DPI techniques for application identification and classification. This is essential when analysing applications that use non-standard port numbers (e.g., P2P, Skype).
3. *Rule engine*: 5Greplay has a rule engine that allows the detection of complex sequence of events that conventional monitoring tools does not detect. This is used to monitor: i) access control policies (e.g., that authorized users are authenticated prior to using a critical business application); ii) anomaly or attacks (e.g., excessive

login attempts on the application server); iii) performance (e.g., identification of VoIP calls with QoS parameters exceeding acceptable quality thresholds); and, much more.

4. *Online and offline solution*: 5Greplay is able to process incoming network traffic in real time, or pre-recorded traffic in pcap files.

5. *Service Chains:* 5Greplay is able to alter 5G packets in real-time and can be easily combined with other tools. Its input is a network traffic stream that may be the output of other tools or services. We list some possible combinations of 5Greplay with existing tools in Figure 2. As shown, 5Greplay can be placed after another network replay tool, such as, Tcpreplay to be able to provide more capabilities for altering packets. For example, the TCP/IP layer can be managed by Tcpreplay, while our tool can take care of managing the 5G protocol layers.

6. *Open-source solution*: 5Greplay aims to cover the is a lack of open-source solutions that enable to manually create or edit existing 5G network protocol packets and injecting them in a network, therefore the code is available in our website 5Greplay.

2 Installation

2.1 Pre-Requirement

2.1.1 Hardware

5Greplay can run easily on a (personal) computer to process low network traffic, i. e., less than 100 Mbps (mega bit per second). The minimal requirement is that the computer has at least 2 cores of CPU, 2 GB of RAM and 4GB of free hard drive disk. The computer must possess also one NIC (Network Interface Cards) on which 5Greplay captures network traffic. Higher network bandwidth requires stronger computer.

Network Interface Cards: 5Greplay needs at least 3 NICs: one for capturing network traffic, another for administrating, and a third one re-transmits the communication packets.

→ To achieve the best performance, the capturing NIC should be either Intel X710 or Intel X520 card. These are recommended because they support DPDK that will considerably improve the packet processing. For other hardware architectures, adaptations and tests would need to be performed.

CPU: For the best performance, the use of Intel Xeon class server system is recommended, such as Ivy Bridge, Haswell or newer. The larger the CPU cache, the better the performance obtained.

→ We recommend having at least 16 cores to process 1 Gbps of traffic. For higher bandwidths other server setups are needed, please contact us for more information.

RAM: We recommend using the fastest memory one can buy; and, having one DIMM per channel. One should avoid having 2 or more DIMMs per channel to make sure all memory channels are used to the fullest. It is more expensive to buy 2x16GB than 4x8GB, but with the later, memory access latency increases and the frequency and throughput decreases.

→ We recommend having at least 32 GB of RAM to process 1Gps traffic.

→ We recommend using a Solid State Drive with at least 20 GB of free space.

BIOS Settings: The following are some recommendations on BIOS settings. Different platforms will have different BIOS naming so the following is mainly for reference:

1. Before starting, consider resetting all BIOS settings to their default.
2. Disable all power saving options such as: Power performance tuning, CPU P-State, CPU C3 Report and CPU C6 Report.
3. Select Performance as the CPU Power and Performance policy.
4. Disable Turbo Boost to ensure the performance scaling increases with the number of cores.
5. Set memory frequency to the highest available number, NOT auto.
6. Disable all virtualization options when you test the physical function of the NIC, and turn on `VT-d` if VFIO is required.

2.1.2 Software

Ubuntu >= LTS 16.04: This Ubuntu version is recommended to run 5Greplay as it has been carefully tested on it. Other Ubuntu version can also run 5Greplay. To run 5Greplay on the other Linux distributions and Windows, please contact us. The example terminal commands used in this manual to prepare 5Greplay running environment is suitable for a machine running Ubuntu LTS 16.04.

libxml2-dev, libpcap-dev, libconfuse-dev libsctp-dev:

```
sudo apt-get install libxml2-dev libpcap-dev libconfuse-dev
    libsctp-dev
```

gcc, make, git:

```
sudo apt install gcc make git
```

2.2 Installation

Installing 5Greplay: To install 5Greplay do the following steps:

1. Clone the source code on your machine:
   ```
   git clone https://github.com/Montimage/5GReplay.git
   ```
2. Install MMT-SDK:
   ```
   cd 5GReplay; sudo dpkg -i lib/mmt-dpi*.deb; sudo ldconfig
   ```
3. Compile 5Greplay:
   ```
   make clean
   make
   ```
4. Compile sample rules existing in rules folder. For more details about the 5Greplay rules, see section 3:
   ```
   make sample-rules
   ```
5. Enable debug using gdby:
   ```
   make DEBUG=1
   ```
6. If you want to use Valgrind DRD or Helgrind, you should do :
   ```
   make DEBUG=1 VALGRIND=1.
   #The option VALGRIND=1 adds some instruction allowing
       Valgrind bypass atomic operations that usually causes
        false positive errors in Valgrind.
   ```
7. Refer to Section 3 to configure 5Greplay as need.

Start 5Greplay:

5Greplay command uses the following form: `command [option]`. Where command : is one of the following `compile, info, replay`. To get the different options of each command run:

```
 run "./mmt-5greplay command -h
```

1. **compile**: This command parses rules in .xml file, then compile to a plugin .so file. For example:

```
#to generate .so file
./mmt-5greplay compile rules/forward-localhost.so
    rules/forward-localhost.xml
#to generate code c (for debug)
./mmt-5greplay compile rules/forward-localhost.c
    rules/forward-localhost.xml -c
#To compile all rules existing in the folder rules,
    use the following command:
make sample-rules
```

2. **info**: This command prints information of rules encoded in a binary file (.so).

```
#print information of all available plugins
./mmt-5greplay info
#print information of rules encoded in 'rules/nas-smc
    -replay-attack.so'
./mmt-5greplay info rules/nas-smc-replay-attack.so
```

3. **replay**: This command can analyze either real-time traffic by monitoring a NIC, or traffic saved in a pcap file. The verdicts will be printed to the current screen.

```
#online analysis on eth0
./mmt-5greplay replay -i eth0
#to see all parameters, run ./mmt-5greplay replay -h
#verify a pcap file
./mmt-5greplay replay -t ~/pcap/5G-traffic.pcap
```

3 Configuration File

5Greplay requires a configuration file for setting different options:

1. Online or offline mode
2. Default action to be taken with the non-filtered packets

By default, 5Greplay will try to load the configuration from `./mmt-5greplay.conf`. A configuration file can be given to 5Greplay by using `-c` parameter, for example:

```
./mmt-5greplay -c /home/tata/probe.conf
```

→ Note: An attribute in the configuration file can be overrided by `-X` parameter. Multiple `-X` parameters are accepted. For example, the following command will override `probe-id` and `source` attribute of `input` block to the coresponding values given after `=` sign.

```
./mmt-5greplay.conf -c /home/tata/probe.conf -X input.source
    =/tmp/a.pcap
```

To list the attributes that can be overriden, run:

```
./mmt-5greplay.conf -x
```

A comment line inside a configuration starts by `#` sign.

The options are listed in the following:

`input`: This block configures the input of 5Greplay.

- `mode` can be either `ONLINE` or `OFFLINE` to indicate that 5Greplay will analyze respectively either traffic in realtime from a NIC or the traffic being stocked in a pcap file.
- `source` indicates the source of traffic to be analyze
- `snap-len` indicates maximal size of an IP packet, by default 65355 bytes.

The options `mode` and `source` need to be specified according to the requirements.

- `mode = ONLINE` allows *near* real-time analysis of network traffic. In PCAP mode, the NIC's network interface name needs to be identified.

  For example:

  ```
  input{
     mode = ONLINE
     # For PCAP it is interface name
     source = "eth0"
  }
  ```

- `mode = OFFLINE` allows analysis of a PCAP trace file. The `source` identifies the name of the trace file. The offline analysis is only available for the PCAP mode. For example:

  ```
  input{
     mode    = OFFLINE
     source = "/home/tata/wow.PCAP"
  }
  ```

`output`: This block configures general output parameters.

- `enable`: set to `false` to disable, `true` to enable
- `output-dir`: path to the folder where output files are written
- `sample-interval`: period in seconds to create a new sample file
- `report-description`: true to include rule's description into the alert reports, otherwise it will be excluded (thus rules's descriptions will be an empty string in the reports).Excluding rules's descriptions will reduce the size of reports

For example:

```
file-output {
    enable        = true
    output-dir    = "./"
    sample-interval = 5  #a new sample file is created each 5
         seconds
    report-description = true
}
```

`engine`: The engine configuration block allows configure functionalities related with the 5Greplay rules, detailed in Section 3.

- `thread-nb`: The option indicates the number of threads that 5Greplay will use for processing packets. It must be a positive number. Use 0 to have only one thread to read then analyze packets. Use x to have one thread to read packets and x threads to analyze the packets.
- `exclude-rules`: indicates the range of rules to be excluded from the verification.

  The range of rules is in BNF format: `exclude-rules = "x,y-z"`, in which `x,y, z` are positive numbers.

  For example, `exclude-rules = "1,3-5,7,50-100"` will exclude the rules having id $1, 3, 4, 5, 7, 50, 51, \ldots, 100$.
- `rules-mask`: It indicates the range of rules should be distributed to a specific analysis thread.

  By default, the rules will be distributed increasingly to each thread. For example, given five rules having id 1, 5, 6, 7, 8 and two analysis threads, then, the first thread will analyzes rules 1, 5 while the second one analyzes rules 6, 7, 8. This can be represented by:

  `rules-mask = "(1:1,5)(2:6-8)"`

  Generally, the `rules-mask` uses the following BNF format:

  `rules-mask = (thread-index:rule-range)`

  in which, `thread-index` is a positive integer; `rule-range` is either a positive integer, or a range of numbers (see `exclude-rules`).

  For example, if we have `thread-nb = 3` and `rules-mask = "(1:1,4-6)(2:3)"`, then, thread 1 verifies rules 1,4,5,6; thread 2 verifies only rule 3; and thread 3 verifies the rest.

  → Note: if we have `thread-nb = 2` and `rules-mask = "(1:1)(2:3)"`, then only rules 1 and 3 are verified, the other rules are not.
- `ip-encapsulation-index` this option selects which IP layer will be analyzed in case there exist IP encapsulation. Its value can be one of the followings:

  - `FIRST`: first IP in the protocol hierarchy
  - `LAST`: last IP in the protocol hierarchy
  - i: i^{th} IP in the protocol hierarchy.

  For example, given ETH.IP.UDP.GTP.IP.TCP.VPN.IP.SSL,

  - `FIRST`, or 1, indicates IP after ETH and before UDP
  - `LAST`, or any number $>=3$, indicates IP after VPN
  - 2 indicates IP after GTP and before TCP

- `max-instances = 100000`: maximum number of instances of a rule

For example,

```
engine
{
    thread-nb       = 0
    exclude-rules   = "(50-100)"
    rules-mask      = ""
    ip-encapsulation-index  = LAST
    max-instances   = 100000
}
```

`mempool`: The mempool configuration block sets the maximum elements of a pool.3.

- `max-bytes`: This parameter set the Maximum bytes of a pool. For example, for 2 GBytes write 2000000000.
- `max-elements`: Max number of elements in a pool.
- `max-message-size`: Maximum size, in bytes, of an input.
- `smp-ring-size`: Number of reports can be stored in a ring buffer.

For example,

```
mempool
{
  max-bytes = 2000000000
  max-elements = 1000
  max-message-size = 3000
  smp-ring-size = 1000
}
```

`forward`: The forward configuration block configures parameters related with the forwarding capability of 5Greplay.3.

- `enable`: Set to `false` to disable, `true` to enable.
- `output-nic`: Output network interface to forward the network traffic.
- `nb-copies`: Number of copies of a packet to be sen
- `snap-len`: Specifies the snapshot length to be set on the handle.
- `promisc`: Specifies whether the interface is to be put into promiscuous mode. If promisc is non-zero, promiscuous mode will be set, otherwise it will not be set.
- `default`: Default action when packets are not selected/satisfied by any rule. Either FORWARD to forward the packets or DROP to drop the packets.

- `target-protocols`: List of transport protocols over which the forwarded packets will be sent.Either SCTP, if the forwarded packets work over SCTP, UDP if the forwarded packets work over UDP or both is both types of packets are present.
- `target-hosts`: List of IP addresses to forward the packets. Each address matches with the transport protocol selected before.
- `target-ports`: List of ports to forward the packets. Each address matches with the transport protocol selected before.

In the following example, STCP traffic will be send over 127.0.0.5:38412 and UDP traffic over 127.0.0.7:2152.

```
forward
{
  enable      = true
  output-nic  = "lo"
  nb-copies   = 2
  snap-len    = 0
  promisc     = 1
  default     = DROP
  target-protocols = { SCTP, UDP}
  target-hosts     = { "127.0.0.5", "127.0.0.7" }
  target-ports     = { 38412, 2152 }
}
```

4 5Greplay rules

4.1 Description

The 5Greplay rules are intended for formally specifying events on the network that denotes packets to be forwarded, dropped or modified. They rely on LTL (Linear Temporal Logic) and are written in XML format. This has the advantage of being a simple and straight forward structure for the verification and processing performed by the tool. In the context of this document, we use the terms of properties and rules interchangeably.

When defining a rule, users must indicate in the rule:

- which packet will be process (filtering)
- which action will be used (dropping, forwarding, or modifying)
- how to modify the packet (definition of the value to be changed in case of the action to be taken is a modification)

Description: A 5Greplay rule is a XML file that can contain as many events as required to filter the desired packets to apply the actions defined by the properties (i.e forwarding or dropping it, with or without modification)

The file needs to begin with a `<beginning>` tag and end with `</beginning>`. Each property begins with a `<property>` tag and ends with `</property>`. A property is a "general ordered tree", that can be graphically represented as shown in Figure 3:

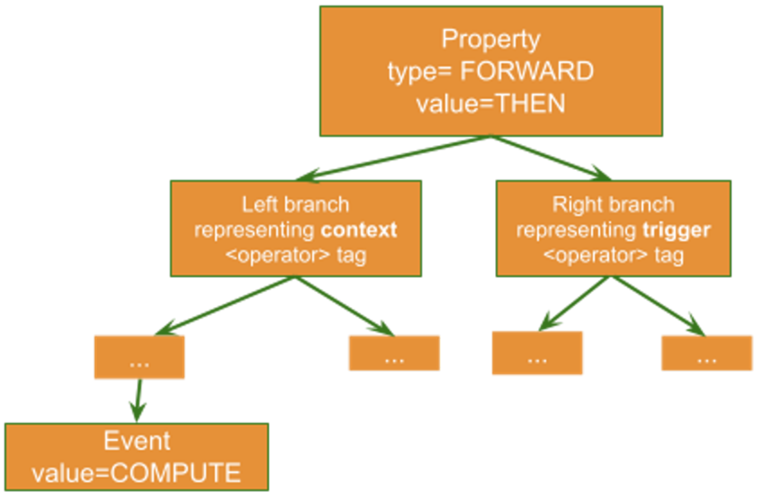


Figure 3: 5Greplay property structure

Property validation: The nodes of the property tree are: the property node (required), operator nodes (optional) and event nodes (required). The property node is forcibly the root node and the event nodes are forcibly leaf nodes. In general, the left branch represents the context and the right branch represents the trigger. This means that the property is found valid when the trigger is found valid; and the trigger is checked only if the context is valid.

The following example rule selects only the packets that come from 192.168.0.15. These packets will not be forwarded to the output NIC.

Thus, if 5Greplay is configured with the default action in the config file is forward, then it forwards the traffic that does not come from 192.168.0.15.

```
<property property_id="100" type_property="FORWARD"
    description="Drop any IP traffic that comes from
    192.168.0.15"  if_satisfied="#drop()">
    <event description="Not interested traffic"
        boolean_expression="(ip.src == ’192.168.0.15’)"/>
</property>
```

Property attributes: The `<property>` tag contains several attributes, some required, some optional:

Table 1: Property tag attributes

Attribute	Accepted values	Req./opt., default value	Description
property_id	Integer	required	Property id number, should go from 1 to n where n is the total number of properties in the XML file. In one or different XML files, two different properties must have two different `property_id`
value	`"THEN"`	required	A property is a tree and each node can have one or more branches. The <property> tag here represents the root of a tree that can have 2 branches: the context and the trigger. The value attribute indicates that the left branch representing the context needs to be valid before, after or at the same time we check the right branch representing the trigger, depending on the delays defined for the node.
type_property	`"FORWARD"`	required	This value is always `"FORWARD"` .
`delay_min`	Integer	optional, default value 0	Defines the validity period ([delay_min, delay_max]) of the left branch (e.g. context). If we have [a,b] with a=b=0 then the left branch needs to occur in the same time as the right branch. If we have a<b<=0 then the right branch needs to have occurred in the past time interval with respect to the left branch. If we have 0<=a<b then the right branch needs to occur in the future with respect to the left branch. If the time runs out before detecting the events concerned then we are in a TIMEOUT condition, i.e., the context occurs but the trigger never arrived in the specified time interval. This would mean that the property failed. Note that in the case that an event should not occur during a certain time interval, we refer to it as TIMEIN instead of TIMEOUT. The default unit of time used is the second but this can be changed using the attribute delay_units.
delay_max	Integer	optional, default value 0	For delay_min, a + sign before the value (e.g., “+0”) means strictly greater than the value and for delay_max a - sign means less than.
delay_units	Integer	optional, default value is “s” (seconds)	Defines the time units used in delay_min and delay_max attributes. If default value is not wished then this has to be defined before these two attributes. Possible values are Y,M,D,H,m,s(default),ms or mms.
description	String	required	The text that clearly explains what the property is about.

Table 2: Property tag attributes (cont.)

Attribute	Accepted values	Req./Opt., default value	Description
if_satisfied	String	optional	Defines what action should be performed if the property is satisfied. The string gives the name of the function that should be executed (see Reactive Functions 4.2).
keep_state	One or more event_id"s separated by commas	optional	Allows indicating what event states should be kept. This is used when a property, for which the context becomes valid, should be able to detect all the triggers that occur in the defined time interval and not be "consumed" by the first one.

Operator attributes: The < operator> tag in a property contains several attributes, some required, some optional:

Table 3: Operator tag attributes

Attribute	Accepted values	Req./Opt., default value	Description
value	`"THEN"`, `"OR"`, `"AND"`, `"NOT"`	required	Operators are used to combine different events and build complex events. "THEN" operator is used to describe ordered events, "AND" operator is used to describe two events without any order and "OR" operator is used to describe the occurrence of a least one of the events. "NOT" negates the underlying sub-tree.
description	String	optional	Gives the text that clearly explains what the complex event is about.
delay_min	Integer	optional, default value 0	Same as for the <property> tag. Note that these attributes are not to be used for the "OR" and "NOT" operators.
delay_max	Integer	optional, default value 0	
counter_max	Integer	optional, default value 0	
counter_min	Integer	optional, default value 0	
repeat_times (under development)	Integer	optional, default value 1	Allows detecting the repetition of a complex event occurrence a number of times.

Event attributes: Properties indicate the sequence of events that need to be observed. Events indicate the conditions that need to be verified on a packet or a set of packets for the event to hold. The <event> tag in a property contains several attributes, some required, some optional:

Table 4: Event tag attributes

Attribute	Accepted values	Req./Opt., default value	Description
event_id	Integer	required	This field allows identifying each event and starts from 1 to n where n is the total number of events in the current property.
description	String	required	Gives the text that clearly explains what the event is about.
value	`"COMPUTE"`	required	Means that the events need to resolve a Boolean expression that is equal to `true` or `false` depending on the attributes values
boolean_expression	expression	required	A Boolean expression, similar to a Boolean expression in the C language (explained in the following paragraph).

Boolean expressions: The `boolean_expression` is a logical combination of "<protocol name>.<field name>" and "<number>" with the following operators: "&&", "||", ">", ">=", "<", "<=", "==", "!=", "+", "-", "/", "*". Blanks and tabs are ignored. Note that the "&&" operator needs to be written in XML as "&amp;&amp;", "<" needs to be written as "&lt;", etc.

A simple `boolean_expression` can be of the form:

```
(<expression> <operator> <expression>)
```

in which, `<expression>` is one of the followings:

- `<protocol_name>.<attribute_name>`
- `<protocol_name>.<attribute_name>.<event_id>`
- `<numeric_value>`
- `<string_value>`
- `true`
- `false`
- embedded function

The Boolean expressions must respect some syntax rules:

- The building bloc needs to start with a "(" and end with a ")". Note that to avoid any ambiguities the quotes should explicitly define how the expression is to be resolved, e.g, A && B || C should be written as ((A && B) || C) or (A && (B || C)).
- An attribute needs to be identified with its `protocol_name` and its `attribute_name` separated by a dot ".".
- If the attribute refers to an attribute value from another event, then the name needs to be followed by a second "." and the `event_id` number of the event concerned. For instance we can have:
  - `(arp.ar_op == 0)` means that the field `arp.ar_op` should be equal to zero.

- `(arp.ar_op == arp.ar_op.1)` means that the field `arp.ar_op` should be equal to the field `arp.ar_op` of the event in the same rule with the `event_id` equal to 1.

Example of a Boolean expression: For example; the following expression means that the event is valid if the packet received corresponds to the ARP protocol; the `ar_op` is 2; `ar_sip` is the same as `ar_tip` of an event 1; and, `ar_sha` is different with `ar_sha` of event 2. Where events 1 and 2 occurred before or will occur after depending on the order that the events should occur (defined by the time intervals specified).

```
"(((arp.ar_op == 2)&amp;&amp;(arp.ar_sip == arp.ar_tip.1))
&amp;&amp;(arp.ar_sha != arp.ar_sha.2))"
```

The following table gives a more detailed explanation about the information that is used in the Boolean expression:

Table 5: Boolean expressions

Name	Type	Description
protocol_name	String	Indicates the protocol of the packet that we need to inspect.
attribute_name	String	Indicates the field of the protocol that we will use for verifying a condition in the event.
numeric_value	Integer	Gives the value that will be used to compare with the packet field value. Note that the `protocol_name` and the `attribute_name` allow uniquely identifying the type of value a field contains (e.g. number, IP address, MAC address in binary format). Two specific constants, `true` and `false`, represent respectively a non-zero and zero number.
string_value	String	Gives a chain of characters, enclosed by ' and ', e. g., `'GET'`
reference	Integer	Indicates that we are to use the data obtained for the field from an event that occurred before (a previously received packet that validated the previous event). The name is a number that clearly identifies the event and needs to be exactly the same as the value given in the `event_id` attribute of the corresponding <event> tag. Note that one should refer to events that occurred beforehand in the property.

Extensions: Embedded functions can be used to extend the elements used in the Boolean expressions (see section 4.2).

Property compilation: After creating the XML properties, we need to compile them by using the executable `compile_rule`:

```
./compile_rule <rule_name>.so <property>.xml
```

4.2 Embedded Functions

Embedded functions are functions that allow implementing calculations that are too complicated to define using only classical operators on fields in the Boolean expressions of security properties. One can use existing embedded functions or implement a new function. In both cases, they can be used in the Boolean expressions by using the syntax:

#<name_of_function>(<list of parameters>)

For instance:

`(#em_is_search_engine( http.user_agent ) == true)`

where `http` is the protocol name and `user_agent` is the attribute name (i.e., packet meta-data).

Implement new Embedded Functions: The embedded functions must respect the C syntax and are used in Boolean expressions using the syntax described above. Embedded functions are implemented inside `<embedded_functions>` tag of the XML rule files and will be compiled together with the rules. The input parameters are either of the type `const char *` or `int val`. For instance, the `em_is_search_engine` above could be implemented as follows:

```
<embedded_functions><![CDATA[
//code C
static inline bool em_is_search_engine(const char *agent){
    if( strstr( agent, "google" ) != NULL )
        return true;
    if( strstr( agent, "bing" ) != NULL )
        return true;
    return false;
}]]></embedded_functions>
```

→ Note: : The function name should be prefixed by `em_` to avoid any confusion with other functions in the system.

One can also implement code into 2 other functions to initialize and free common resource:

- `void on_load(){ ... }` is called when the rules inside the XML file being loaded into 5Greplay
- `void on_unload(){ ... }` is called when exiting 5Greplay

Pre-implemented Embedded Functions: There exist several embedded functions that have been implemented inside 5Greplay.

1. `is_exist( proto.att )` checks whether an event has an attribute of a protocol, e. g., `is_exist( http.method )` will return `true` if the current event contains protocol HTTP and attribute `method` has a non-null value, otherwise it will return `false`.

   Normally 5Greplay has a filter that allows an event in a rule to be verified only if any proto.att used in its boolean expression contains value. If one of them has not, the rule will not be verified. This allows to reduce number of verification of boolean expression, thus increases the performance.

   For example, given an event having the following boolean expression:

   `((ip.src != ip.dst) && (#em_check_URI(http.uri) == 1))`

   This expression is verified only if `ip.src` and `ip.dst` and `http.uri` are not null, hence only HTTP packets are verified (it does not verify every IP packets).

   However, if one use the following expression, that is totally having the same meaning with the previous one:

   ```
   ((ip.src != ip.dst) && ((is_exist(http.uri) == true )
            && (#em_check_URI(http.uri) == 1)))
   ```

5Greplay needs to verify the expression against any IP packet as the `is_exist` function tell the engine to exclude `http.uri` from its filter.

2. `is_empty( proto.att )`, e.g., `is_empty(http.uri)` checks whether the string value is empty, i.e., its length is zero.
3. User can use *any standard C functions* as embedded function, e. g., `(#strstr( http.user_agent, 'robot') != 0)` to check if `http.user_agent` contains a sub-string `"robot"`.

→ Note: Before using a C function, the library containing that function need to be included. The following libraries have been pre-included:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "mmt_lib.h"
#include "pre_embedded_functions.h"
```

Thus when using a function that does not defined inside these libraries, one need to include its library. For example:

```
<embedded_functions><![CDATA[
#include <math.h>

static inline bool function em_check( double port ){
    double x = sqrt( port );
    ...
}
]]></embedded_functions>
```

Reactive Functions: Reactive functions allow user perform some action when a rule is satisfied. The functions will be called each time their rules are satisfied. When a security and attack rules are satisfied, they will give `not_respected` and `detected` verdicts respectively.

To implement and use a reactive function, one need to

1. implement a C function inside `<embedded_functions>` tag. The function has the following format:

```
typedef void (*mmt_rule_satisfied_callback)(
  const rule_t *rule,//rule being validated
  int verdict,       //DETECTED, NOT_RESPECTED
  uint64_t timestamp,//moment (by time) the rule is
      validated
  uint64_t counter,  //moment (by order of message) the
      rule is validated
  const mmt_array_t * const trace  //messages that
      validate the rule
);
```

2. put the function name in attribute `if_satisfied` of the rule you want to react. For example: `if_satisfied="em_print_out"`

```
<beginning>
<embedded_functions><![CDATA[
static void em_print_out( const rule_info_t *rule, int
    verdict, uint64_t timestamp, uint64_t counter, const
    mmt_array_t * const trace ){
   const char* trace_str =
       mmt_convert_execution_trace_to_json_string( trace,
        rule );
    printf( "detect rule %d\n%s\n", rule->id, trace_str );
}]]></embedded_functions>

<property value="THEN" delay_units="s" delay_max="0"
    property_id="10" type_property="EVASION"
    description="HTTP using a port different from 80 and
        8080." if_satisfied="em_print_out">
    <event value="COMPUTE" event_id="1"
        description="a port different from 80 and 8080"
            boolean_expression="((http.method != '')&amp;&
                amp;((tcp.dest_port != 80)&amp;&amp;(tcp.
                dest_port != 8080)))"/>
    <event value="COMPUTE" event_id="2"
            description="HTTP packet"
            boolean_expression="(ip.src != ip.dst)"/>
</property>
</beginning>
```

4.3 Example

4.3.1 Property without Embedded Functions:

In the following example rule, it selects any packets that satisfy the following IF-THEN condition: IF a packet is a 5G authentication response, and it contains information about uplink NAS transport and IF it is from any UEs having RAN id $<$ 10, being the two conditions described in the`boolean_expression` of the events 1 and 2 respectively; THEN before being forwarded to the output NIC, the selected packets will be updated via update function defined in `if_satisfied`: increase the RAN id attribute of the packet by 100.

```
<property value="THEN" property_id="101" type_property="
    FORWARD"  description="Increase RAN IDs of packets with
    RAN ID < 10"
        if_satisfied="#update(ngap.ran_ue_id, (ngap.ran_ue_id
            .2+100))">
    <event event_id="1"
        description="Authentication response, Uplink NAS
            Transport"
        boolean_expression="((ngap.procedure_code == 46) &amp
            ;&amp;  (nas_5g.message_type == 87))"/>
    <event event_id="2"
        description="A range of UEs"  boolean_expression="(
            ngap.ran_ue_id < 10)"/>
</property>
```

4.3.2 Property with Embedded Functions:

The following example presents a complex modification of packets that is done by using the embedded function. The rule will send to the output NIC 2 packets: the first one with increasing UE's RAN ID by 100, and the second one with updating its AMF and RAN UE's ID.

In the example, the rule filters the same packets as in the example before. In this case the field `if_satisfied` will call the embedded function `em_update_then_forward`.

```
<beginning>
<embedded_functions><![CDATA[
extern void forward_packet();

static void em_update_then_forward( uint64_t ran_ue_id,
    uint64_t amf_ue_id ){
```

```
    set_attribute_value( PROTO_NGAP, NGAP_ATT_RAN_UE_ID,
        ran_ue_id + 100 );
    forward_packet();
    set_attribute_value( PROTO_NGAP, NGAP_ATT_AMF_UE_ID,
        amf_ue_id + 200 );
    set_attribute_value( PROTO_NGAP, NGAP_ATT_RAN_UE_ID,
        ran_ue_id + 100 );
}
]]></embedded_functions>

<property value="THEN" delay_units="s" delay_min="0"
    delay_max="0" property_id="103"
        type_property="FORWARD"
        description=""
        if_satisfied="#em_update_then_forward(ngap.ran_ue_id,
            ngap.amf_ue_id)">
    <event event_id="1"
        description="Authentication response, Uplink NAS
            Transport"
        boolean_expression="(((ngap.procedure_code == 46) &
            amp;&amp;  (nas_5g.message_type == 87))"/>
    <event event_id="2"
        description="A range of UEs"  boolean_expression="(
            ngap.ran_ue_id < 10) "/>
</property>
</beginning>
```

4.4 Compile rules

5Greplay rules are specified in plain text following an XML format. The rules need to be compiled into a dynamic C library used by the tool.

The compiled rules must put in `./rules`. The former has higher priority and only one of them will be taken into account by 5Greplay.

Specifically, if 5Greplay found `./rules` folder in the current folder executing, it will use the rules inside this folder.

Compile rules: To compile all rules existing in the folder `./rules` , use the following command:

```
make sample-rules
```

To compile a rule individually, 5Greplays' command `compile` parses rules in .xml file and compile them to a plugin .so file or c code for debugging purposes. To generate .so file do:

```
./mmt-5greplay compile rules/[output\_file].so rules/[
    property\_file].xml
```

And to to generate code c (for debug) do:

```
./mmt-5greplay compile rules/[output\_file].c rules/[property
    \_file].xml -c
```

where:

1. `output_file`: is the path of file containing the result that can be either a .c file or .so file.
2. `property_file`: is the path where the property file can be found.
3. optional parameters:
    - `-c`: will generate only the C code. This option allows manually modifying the generated code before compiling it. After

generating the C code, the tool prints out the command that needs to be executed for compiling it.

For example, the following command compiles a rule that the user defined in `rules/forward-localhost.xml`

```
./mmt-5greplay compile rules/forward-localhost.so rules/
    forward-localhost.xml
```

Obtain information from the compiled rules:

To get some basic information about the compiled rules (such as, ID, description) 5Greplay provides the command `info`.

- The tool can be used to inspect a specific compiled rule by giving the rule path as parameter, for instance:

```
./mmt-5greplay info rules/nas-smc-replay-attack.so
Found 1 rule.
1 - Rule id: 4
  - type            : attack
  - events_count    : 3
  - variables_count : 7
  - variables       : arp.ar_op (30.5), arp.ar_sha (30.6)
    , arp.ar_sip (30.7), arp.ar_tip (30.9), ethernet.
    dst (99.2), ethernet.packet_count (99.4099),
    ethernet.src (99.3)
  - description     : IPv4 address conflict detection (
    RFC5227). Possible arp poisoning.
  - if_satisfied    : (null)
  - if_not_satisfied: (null)
  - create_instance : 0x7f5063e7be70
  - hash_message    : 0x7f5063e7bcd0
  - version         : 1.1.2 (67fa423 - 2017-5-19 18:14:34
    ), dpi version 1.6.8.0 (b3e727b)
```

5 Conclusions

In this document, we have introduced our 5G replay tool to address the lack of an open-source tool to perform security testing in 5G networks. To achieve such purpose, the tool provides a flexible way to modify 5G protocol packets before injecting them to the target services to be tested. The techniques presented in this tutorial can be applied for the implementation of a wide set of security test cases.

For further information, please send an email to `contact@montimage.com`.