Light-Weight Platform for Attack Validation in LTE Network

Weiqi Wang and Hai Li, *Member, IEEE*

***Abstract*—Within the scope of Long Term Evolution (LTE) security research, open-source implementations of LTE stack are generally used to validate attacks. Since analysis of the code requires advanced understanding of both specifications and programming, validating attacks is tiring and time-consuming. To address the difficulty, we develop LTE Evaluation Assistant Platform (LEAP), an extension of OpenAirInterface (OAI). It provides a Python interface to monitor and manipulate traffic inside OAI. LEAP adopts a novel architecture to reduce complexity. We also investigated the tradeoff between high level customization, usability, and latency. The evaluation result shows LEAP could effectively simplify attack validation process.**

***Index Terms*—LTE, attack valildation, OpenAirInterface, distributed architecture.**

I. Introduction

With billions of subscribers, Long Term Evolution (LTE) is one of the most fundamental communication technologies in modern society. It has thus become the target of various attacks. Generally, current research on LTE security includes attack detection and validation. Whereas the former may differ because of different methodologies, researchers usually leverage open-source projects to validate attacks [1]–[8] because of the high cost of commercial stack implementations and inaccessibility to facilities of mobile operators.

The open-source community witnessed a large scale emergence of mobile network stack implementations in 2G period such as OpenBTS [9] and OpenBSC [10], In LTE era, OpenAirInterface (OAI) [11] and srsLTE [12] are widely used. Because such projects aim to run mobile networks on general purpose processors rather than conduct security tests, using them to validate attacks can be difficult. On one hand, such projects typically contain huge amount of code and use lower-level languages. An example is [1], they analyzed more than 90K lines of C++ code of OpenLTE and srsLTE and implemented 3,470 lines of code for the testbed. On the other hand, the modification done by one research cannot be used by another because the attacks are different. Besides, official documentation is more related to telecommunication and does not depict code structure or how to perform security test.

Therefore, we developed LTE Evaluation Assistant Platform (LEAP) to address these difficulties (available at https://github.com/pmcrg/LEAP). LEAP is based on OAI which provides the most comprehensive simulation of all key LTE network elements (UE, eNodeB, MME, HSS, and SPGW). LEAP can help users easily monitor and manipulate messages inside OAI. It also uses a series of methods to reduce latency so as to avoid violating the timers. The subtle balance between customization, usability, and latency allows LEAP to substantially reduce the difficulties in validating attacks.

The remainder of this letter is organized as follows. Section II describes the architecture. Section III compares LEAP with existing testbeds and explains the advantages of LEAP. Section IV evaluates the performance of LEAP by using it to validate existing attacks. The conclusion is given in Section V.

II. Architecture

LEAP adopts a Client/Server architecture as shown in Fig. 1. The server part (denoted by LEAP_S) is embedded in OAI and is written in C language. The client part (denoted by LEAP_C) is a Python library. The connection is based on SCTP protocol and utilizes two kinds of messages. LEAP message (denoted by LEAP_{msg}) is used to transfer intercepted message in OAI. LEAP command (denoted by LEAP_{cmd}) is used to instruct OAI to execute certain procedure.

A. LEAP Server

- **Instrumentation:** To monitor and manipulate traffic in OAI, users need to instrument a probe statement in OAI where they want to change the execution of OAI, indicating what message to intercept and how the procedure is changed. The statement is provided by LEAP_S and can be instrumented in any network elements.
- **Functions:** This module contains encapsulated message handle functions in OAI. When a LEAP_{cmd} is processed, either message sending functions are called to inject messages or message retrieving functions are called to send context information to LEAP_C.
- **LEAP_{cmd} parsing:** This module parses incoming byte stream into LEAP_{cmd}. LEAP_{cmd} is then analyzed and essential arguments are passed to `Functions` module.
- **LEAP_{msg} packing:** This module analyzes intercepted OAI message, extracts important information to generate LEAP_{msg}, and sends it to LEAP_C.

B. LEAP Client

- **User defined script:** This is a Python script implemented by users. It describes the attack procedure.
- **Shared libraries:** The procedures that requires heavy computation are implemented in low-level language and

Manuscript received May 10, 2020; revised July 31, 2020; accepted September 9, 2020. Date of publication September 18, 2020; date of current version December 7, 2020. The associate editor coordinating the review of this article and approving it for publication was N. Passas. *(Corresponding author: Hai Li.)*

The authors are with the School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, China (e-mail: weiqiwang@bit.edu.cn; haili@bit.edu.cn).

Digital Object Identifier 10.1109/LNET.2020.3025019

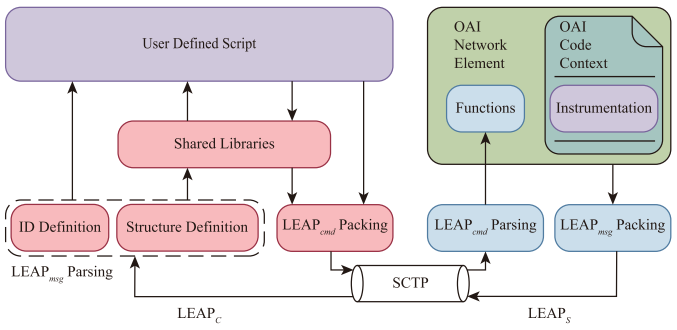

Fig. 1. Architecture of LEAP. $LEAP_S$ is marked in blue. $LEAP_C$ is marked in red. The parts that are implemented by users are marked in purple.

8 7 6 5 4 3 2 1	
message_id	octet 1
ue_id	octet 2
length	octet 3
message[0]	octet 4
…	…
message[length-1]	octet 4+(length-1)

Fig. 2. LEAP message information element.

compiled to shared libraries. They are loaded in user defined script to offload heavy computation procedure.

- **ID definition:** This module contains the mapping between readable names and ID of OAI messages. Readable names are used in user defined script. They are mapped to ID when being transferred in socket.
- **Structure definition:** This module defines OAI message structures in Python. If `message` field of a $LEAP_{msg}$ is filled, this module is used to convert it to structure.
- **$LEAP_{cmd}$ packing:** This module parses user defined script, generates $LEAP_{cmd}$ and sends to $LEAP_S$.

C. Communication

The $LEAP_{msg}$ and $LEAP_{cmd}$ information element is coded as shown in Fig. 2 and Fig. 3.

- **message_id/command_id:** This field is internal identification of $LEAP_{msg}$ and $LEAP_{cmd}$.
- **ue_id:** In $LEAP_{msg}$, this field represents which user equipment the message belongs to. In $LEAP_{cmd}$, it indicates which user equipment to send message to.
- **cause:** This field is used to transfer the `cause` field of certain messages such as AUTHENTICATION_REJECT.
- **length:** This field represents the length of `message` field. It's set to zero if there is no content in `message` field.
- **message:** This field is optional. If needed, it could be used to transfer additional data. The memory of this field dynamically allocated.

The interaction between $LEAP_S$ and $LEAP_C$ is as follows:

1) OAI message is intercepted by probe statement.
2) $LEAP_S$ generates $LEAP_{msg}$ and sends it to $LEAP_C$.
3) $LEAP_C$ parses $LEAP_{msg}$. If $LEAP_{msg}$ contains encoded message, it is passed to shared libraries to be decoded. Otherwise it is directly passed to user defined script.

8 7 6 5 4 3 2 1	
command_id	octet 1
ue_id	octet 2
cause	octet 3
length	octet 4
message[0]	octet 5
…	…
message[length-1]	octet 5+(length-1)

Fig. 3. LEAP command information element.

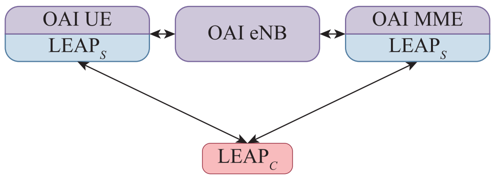

Fig. 4. Deployment example.

4) $LEAP_C$ generates $LEAP_{cmd}$ according to user defined script and sends to $LEAP_S$. Shared libraries may be needed to encode the message during the process.
5) $LEAP_S$ parses $LEAP_{cmd}$ and calls relevant functions to execute indicated procedures.

III. Comparision

In this section, we compare the architecture of LEAP and existing testbeds. Then we describe how we reach the balance between high level customization, usability and low latency.

A. Distributed Architecture

The architecture of LEAP introduces a great reduction in system complexity. Rupprecht *et al.* adopts integrated architecture, implementing a hierarchical set of controllers in OAI [13]. The framework can only be used as a whole and lacks scalability. Fang and Yan uses distributed architecture but they embed the reinforcement learning algorithm at server side (OAI) [14]. Therefore, the research-oriented part is not separated and the testbed cannot be used in other research. We use distributed architecture and optimized the distribution of tasks between client and server. $LEAP_S$ simulates LTE network and is embedded with a light-weight SCTP server. All research-oriented parts are placed at $LEAP_C$. When changing attack procedure, only client script needs to be modified. Furthermore, $LEAP_S$ and $LEAP_C$ can be deployed independently as shown in Fig. 4. When validating an attack exploiting UE and MME, eNB only serves as a message relay point and requires no deployment. Besides, since one socket connection is regarded as an object in $LEAP_C$, multiple network elements can be controlled by one script.

B. Customization

1) Test Scope: Existing testbeds usually focus on certain procedures or layers to reduce complexity such as [13] and [14]. However, LEAP allows users to specify test scope. The layer is specified by instrumentation and attack procedure is specified by user defined script. Thus the scope of LEAP is not constrained to certain research.

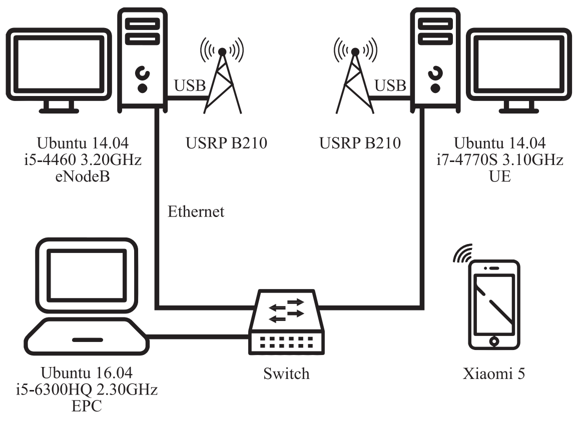

Fig. 5. Evaluation environment setup.

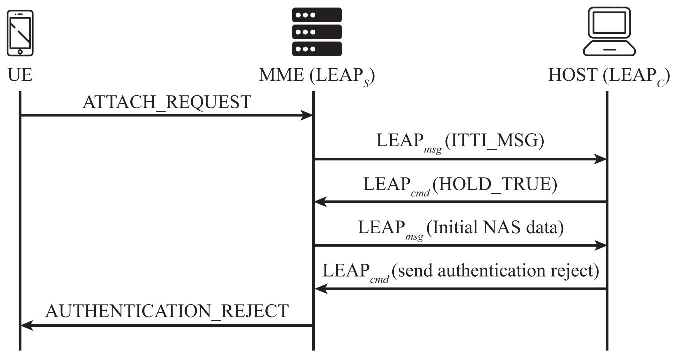

Fig. 6. Message flow of numb attack validation.

2) Accessibility of Message Content: We analyzed OAI message structures and rewrote them in Python using ctypes library. Users can extract members from OAI message structures. In coding the script, users can either print the message to generate logs, which is the same monitoring interface as [13] and [14], or use message contents in logical operation.

C. Usability

The user interface of LEAP is Python language, which is famous for its simplicity. The amount of code needed to use LEAP is about ten lines and it only requires basic programming knowledge such as loop and condition statements.

D. Low Latency

Low latency and usability is in contradiction because in our case, ease of use means high-level programming language and higher latency. We managed to increase speed by offloading computation and cut down network traffic cost. First, the heavy computation is offloaded to shared libraries implemented in C language. Second, we transfer indication in SCTP socket most of the time instead of the whole message. In most cases, `command_id`, `ue_id`, and `cause` fields in LEAP$_{cmd}$ are enough to indicate a need to send certain message. The message generation is then executed in C language at OAI, which is faster than Python. Third, if the whole message needs to be sent over SCTP socket, `message` field is provided to store the message. Its memory is dynamically allocated according to actual length of the message, which is efficient in reducing memory and traffic cost.

```
le = leap("127.0.0.1","7897")
while True:
  message_id, ue_id, length, msg = le.recv(buf_size)
  if message_id == LEAP_ITTI_MSG:
    le.hold(True)
  elif message_id == LEAP_INITIAL_NAS_DATA:
    nas_msg = le.nas_message_decode(msg)
    message_type = nas_msg.contents.header.message_type
    if message_type == ATTACH_REQUEST:
      ppstruct(pointer(nas_msg.contents.attach_request.additionalguti.imsi))
      le.send_authentication_reject(ue_id)
      le.exit_loop()
  else:
    le.exit_loop()
```

Script 1. Script example of numb attack validation.

IV. Evaluation

In this section, We used LEAP to validate three attacks exploiting the attach procedure–IMSI catch, Numb, and Authentication Synchronization Failure (denoted by ASF) [7]. First, we give an illustration of our evaluation environment. Second, we give the validation example of Numb Attack and ASF to show LEAP's support for both small-scale and large-scale message interaction. Finally, we compare attack validation process with and without LEAP.

We build the evaluation environment with the real-world setup of OpenAirInterface. As shown in Fig. 5, three machines simulates EPC, eNB, and UE respectively. Two USRP B210s [15] simulates real-world air interface. A COTS UE: Xiaomi 5 is also used to test the response of real device.

A. Numb Attack Validation

1) Validation Procedure: The message flow of this attack is shown in Fig. 6. The validation consists of OAI instrumentation and user defined script. In MME, two `leap_send()` are instrumented where ATTACH_REQUEST is processed (openair-cn/src/nas/nas_mme_task.c:66). The first one intercepts `ITTI_MSG` which indicates the start of ATTACH_REQUEST processing. The second intercepts NAS message and sends to LEAP$_C$. Script 1 is an example of user defined script. After receiving `ITTI_MSG` indication, it responds with `HOLD_TRUE` which pauses current procedure of OAI. Then as LEAP$_S$ sends NAS message, LEAP$_C$ decodes and examines the contents. Here, IMSI can be printed to perform IMSI catch. Finally, LEAP$_C$ instructs OAI to respond with an AUTHENTICATION_REJECT.

2) Latency Analysis: We performed the validation three times and recorded the time used t_i of major procedures. The result is shown in Table I. The average total time is 3.214ms, which remains within proper timer range according to 3GPP TS 24.301 T3410 [16] of which the value is 15 seconds.

B. ASF Attack Validation

1) Validation Procedure: The message flow of this attack is shown in Fig. 7. Script 2 is a script example. This attack utilizes a malicious UE to send ATTACH_REQUEST multiple times so as to increase SQN in HSS and therefore cause sync failure on victim UE when it starts attaching. To intercept and replay ATTACH_REQUEST, `leap_send()` and a for

TABLE I
TIME CONSUMED IN ATTACK VALIDATION PROCEDURE

Procedure	t_1 / ms	t_2 / ms	t_3 / ms	$\bar{t}$ / ms
Retrieve LEAP$_{msg}$	0.481	0.218	0.334	0.344
Retrieve EMM Security Context	0.164	0.236	0.180	0.193
NAS message decode (Offloaded)	0.002	0.002	0.002	0.002
NAS message decode (Total)	0.325	0.581	0.306	0.404
Send LEAP$_{cmd}$	0.002	0.002	0.002	0.002
Attack validation (Total)	3.776	2.776	3.089	3.214

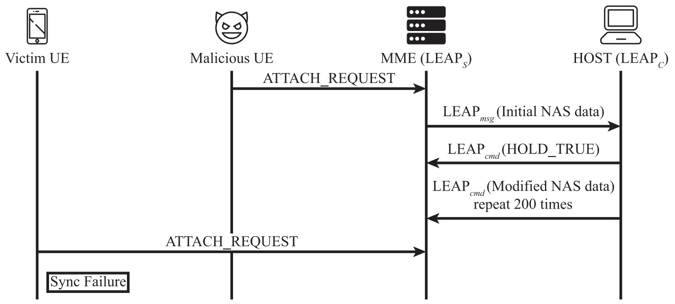

Fig. 7. Message flow of ASF attack validation.

```
le = leap("127.0.0.1","7897")
flag = 0
while True:
  message_id, ue_id, length, msg = le.recv(buf_size)
  if message_id == LEAP_INITIAL_NAS_DATA:
    if flag == 0:
      le.hold(True)
      for y in range(200):
        modified_msg = msg[:13] + chr(0x80>>random.randint(0,7)) + \
        chr(0x80>>random.randint(0,7)) + msg[15:]
        le.send(modified_msg)
      flag = 1
    else:
      le.hold(False)
  else:
    le.exit_loop()
```

Script 2. Script example of ASF attack validation.

TABLE II
COMPARISON ON MANUAL AND LEAP VALIDATION

Compare contents	IMSI catch		Numb		ASF	
	Manual	LEAP	Manual	LEAP	Manual	LEAP
C code analyzed (lines)	0	0	961	0	179	0
C code added (lines)	15	1	7	1	11	5
Python code added (lines)	0	3	0	4	0	12

loop is instrumented. In order to prevent HSS from dropping consecutive ATTACH_REQUEST, the encryption and integrity protection algorithms field of each ATTACH_REQUEST is set to random value. The modified message is then sent back to LEAP$_S$ for OAI to continue attach procedure. Such procedure is done 200 times to increase SQN in HSS. Finally, we attach the COTS UE and it receives sync failure.

C. Results

The evaluation results of three attack validation are shown in Table II. All three validation are done with the aim of minimizing code analysis and addition to present the minimal effort as beginners. As shown in Table II, LEAP could greatly reduce the work on analyzing C code and total code needed is about ten lines. Besides, LEAP isolates the output location with OAI log. The output location of LEAP is Python terminal. The information displayed is fully controlled by users and won't mix with log of OAI, thus easier to read. Furthermore, when changing attack procedure, manual validation requires users to modify OAI and compile again whereas LEAP only needs to modify Python script and no recompilation is required.

V. CONCLUSION

In this letter, we proposed a novel light-weight architecture for attack validation platform and investigated the tradeoff between customization, usability, and latency. Based on the architecture, we developed LEAP and tested it using existing attacks. The overall performance proves LEAP could greatly reduce the barrier on validating attacks using open-source implementation of LTE stack and thus accelerate the research on LTE security. In the future, we'd like to extend shared library to support more layers as well as integrate instrumentation into LEAP$_S$ so that only script is needed when validating attacks.

REFERENCES

[1] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the untouchables: Dynamic security analysis of the LTE control plane," in *Proc. IEEE Symp. Security Privacy (SP)*, 2019, pp. 1153–1168.
[2] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper, "Breaking LTE on layer two," in *Proc. IEEE Symp. Security Privacy (SP)*, 2019, pp. 1121–1136.
[3] R. P. Jover, "LTE security, protocol exploits and location tracking experimentation with low-cost software radio," 2016. [Online]. Available: arXiv:1607.05171.
[4] S. F. Mjølsnes and R. F. Olimid, "Easy 4G/LTE IMSI catchers for non-programmers," in *Proc. Int. Conf. Math. Methods Models Archit. Comput. Netw. Security*, 2017, pp. 235–246.
[5] M. Lichtman, R. P. Jover, M. Labib, R. Rao, V. Marojevic, and J. H. Reed, "LTE/LTE-A jamming, spoofing, and sniffing: Threat assessment and mitigation," *IEEE Commun. Mag.*, vol. 54, no. 4, pp. 54–61, Apr. 2016.
[6] A. Shaik, R. Borgaonkar, N. Asokan, V. Niemi, and J.-P. Seifert, "Practical attacks against privacy and availability in 4G/LTE mobile communication systems," 2015. [Online]. Available: arXiv:1510.07563.
[7] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "LTEInspector: A systematic approach for adversarial testing of 4G LTE," in *Proc. Netw. Distrib. Syst. Security (NDSS) Symp.*, 2018, pp. 1–15.
[8] R. P. Jover, J. Lackey, and A. Raghavan, "Enhancing the security of lte networks against jamming attacks," *EURASIP J. Inf. Security*, vol. 2014, no. 1, p. 7, 2014.
[9] *OpenBTS*. Accessed: Jul. 25, 2019. [Online]. Available: http://openbts.org/
[10] Osmocom. *OpenBSC*. Accessed: Jul. 23, 2019. [Online]. Available: http://osmocom.org/projects/openbsc
[11] Eurecom. *OpenAirInterface*. Accessed: Jul. 16, 2019. [Online]. Available: http://www.openairinterface.org
[12] S. R. Systems. *srsLTE*. Accessed: Jul. 19, 2019. [Online]. Available: https://github.com/srsLTE
[13] D. Rupprecht, K. Jansen, and C. Pöpper, "Putting LTE security functions to the test: A framework to evaluate implementation correctness," in *Proc. 10th USENIX Workshop Offensive Technol. (WOOT)*, 2016, pp. 1–12.
[14] K. Fang and G. Yan, "Emulation-instrumented fuzz testing of 4G/LTE android mobile devices guided by reinforcement learning," in *Proc. Eur. Symp. Res. Comput. Security*, 2018, pp. 20–40.
[15] *USRP B210*. Accessed: Jul. 18, 2019. [Online]. Available: https://www.ettus.com/product/details/UB210-KIT
[16] *Non-Access-Stratum (NAS) Protocol for Evolved Packet System (EPS); Stage 3*, 3GPP Standard TS 24.301, 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072