
**PhD in Information Technology and Electrical Engineering**

**Università degli Studi di Napoli Federico II**

# PhD Student: Pietro Liguori

**XXXIV Cycle**

## Training and Research Activities Report – Third Year

**Tutor: Domenico Cotroneo – co-Tutor: Roberto Natella**

## 1. Information

**PhD candidate**: Pietro Liguori
**Date of birth**: 07/09/1989
**Master Science title**: Master degree in Computer Engineering (cum laude), University of Naples Federico II
**Doctoral Cycle**: XXXIV
**Fellowship type**: PhD student grant (Grant Type: Academic)
**Tutors**: Prof. Domenico Cotroneo, Prof. Roberto Natella
**Year**: Third

I received my MS degree (cum laude) in Computer Engineering from the Univesità degli Studi di Napoli Federico II in July 2018.

My master thesis focused on anomaly detection in distributed and complex systems such as cloud computing infrastructures. To limit the massive number of false positives (due to the non-determinism of distributed systems), I developed a novel approach of anomaly detection based on a probabilistic model.

I'm currently in the third year of the Ph.D. program in Information Technology and Electrical Engineering (ITEE) at Federico II University of Naples, under the supervision of Prof. Domenico Cotroneo and Prof. Roberto Natella.

## 2. Study and Training activities

During the third year, I attended the following courses and seminars.

| Title | Type | Hours | Credits | Dates | Organizer | Certificate |
|---|---|---|---|---|---|---|
| International Workshop on Artificial Intelligence for IT Operations | Workshop (Seminar) | 8 | 1.6 | December 14, 2020 | University of Berlin, Huawei Munich Research Center and University of Coimbra | Yes |
| Statistical data analysis for science and engineering research | Ad-hoc | 12 | 4 | April 2021 | University of Naples Federico II | Yes |
| 2021 Spring School on Transferable Skills | Ph.D. school | 12 | 2 | May 4-5, 2021 | University of Naples Federico II | Yes |
| The First Workshop on Natural Language Processing for Programming | Workshop (Seminar) | 8 | 1.6 | August 6, 2021 | Association for Computational Linguistics (ACL) | Yes |
| The 32nd International Symposium on Software Reliability Engineering (ISSRE 2021) | Conference (Seminar) | 16 | 3.2 | October 25-28,2021 | IEEE | Yes |

I also have been local organizer for the University of Naples Federico II of ***CyberChallenge.IT***, the first Italian introductory training program in cybersecurity for high-school and undergraduate students.

| | Credits year 1 | | | | | | | | Credits year 2 | | | | | | | | Credits year 3 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | | | 1 | 2 | 3 | 4 | 5 | 6 | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | |
| | Estimated | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | Summary | Estimated | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | Summary | Estimated | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | Summary | Total | Check |
| **Modules** | 25 | 0 | 2,2 | 6 | 9 | 3,6 | 4,8 | **26** | 10 | 4,3 | 0 | 0 | 6 | 0 | 0 | **10** | 5 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | **6** | **42** | 30-70 |
| **Seminars** | 5 | 0,8 | 0 | 0,5 | 3,8 | 0,8 | 0 | **5,9** | 5 | 0 | 3,2 | 0 | 0 | 0 | 0 | **3,2** | 5 | 1,6 | 0 | 0 | 0 | 1,6 | 3,2 | 0 | 0 | **6,4** | **16** | 10-30 |
| **Research** | 30 | 9,2 | 7,8 | 3,5 | 0 | 5,6 | 5,2 | **31** | 45 | 5,7 | 6,8 | 10 | 4 | 10 | 10 | **47** | 50 | 6,4 | 8 | 4 | 6 | 6,4 | 4,8 | 8 | 4 | **48** | **125** | 80-140 |
| | 60 | 10 | 10 | 10 | 13 | 10 | 10 | **63** | 60 | 10 | 10 | 10 | 10 | 10 | 10 | **60** | 60 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 4 | **60** | **183** | 180 |

## 3. Research activity

### *Failure Mode Analysis in Cloud Computing Infrastructures*

In cloud computing infrastructures, the failure mode analysis is a difficult and time-consuming task, due to the size and complexity of failure data. Moreover, failure mode analysis is hindered by the non-deterministic behavior of cloud systems, which causes random variations in the timing and the ordering of events in the system, thus introducing noise in the failure data. Therefore, failure mode analysis techniques must be robust to noise in the failure data.

The adoption of unsupervised machine learning techniques, such as clustering and anomaly detection, comes to the rescue but still faces some limitations. These techniques require the preliminary selection and transformation features (*feature engineering*), to make the failure data more amenable for analysis. This effort requires deep domain knowledge and represents a significant up-front cost.

During this year, I developed a novel approach for efficiently identifying recurrent failure modes from failure data. The approach leverages deep learning for unsupervised machine learning, to overcome the challenges of noise and complexity of the feature space. The approach saves the manual efforts spent on feature engineering, by using an autoencoder to automatically transform the raw failure data into a compact set of features. The approach transforms the data by jointly optimizing for the reconstruction error (i.e., the transformed features are still representative of the sample) and inter-cluster variance (i.e., to make it easier to identify groups of similar failures).

I evaluated the proposed approach on a dataset of thousands of failures from OpenStack, a popular platform used in several private and public cloud computing systems, and the basis of over 30 commercial products. As an additional contribution to this work, we publicly released this dataset for the research community. We compare the proposed approach to a manually fine-tuned clustering technique. The results demonstrate that the proposed approach can identify clusters with accuracy similar, or in some cases, even superior, to the fine-tuned clustering, with a low computational cost.

### *Runtime Failure Detection of the Failures in Cloud Computing*

Nowadays, the cloud infrastructures are considered a valuable opportunity for running services with high-reliability requirements, such as in the telecom and health-care domains. Unfortunately, residual software bugs in cloud management systems can potentially lead to high-severity failures, such as prolonged outages and data losses. These failures are especially problematic when they are silent, i.e., not accompanied by any explicit failure notification, such as API error codes, or error entries in the logs. This behavior hinders the timely detection and recovery, lets the failures to silently propagate through the system, and makes the traceback of the root cause more difficult, and recovery actions more costly (e.g., reverting a database state).

To face these issues, more powerful means are needed to identify these failures at runtime. A key technique in this field is represented by *runtime verification strategies*, which perform redundant, end-to-end checks (e.g., after service API calls) to assert whether the virtual resources are in a valid state. For example, these checks can be specified using temporal logic and synthesized in a runtime monitor, e.g., a logical predicate for a traditional OS can assert that a thread suspended on a semaphore leads to the activation of another thread. Runtime verification is now a widely employed method, both in academia and industry, to achieve reliability and security properties in software systems. This method complements classical exhaustive verification techniques (e.g., model checking, theorem proving, etc.) and testing.

I proposed a lightweight approach to runtime verification tailored for the monitoring and analysis of cloud computing systems. I used a non-intrusive form of tracing of events in the system under test, and I build a set of lightweight monitoring rules from correct executions of the system in order to specify the desired system behavior. I synthesized the rules in a runtime monitor that verifies whether the system's behavior follows the desired one. Any runtime violation of the monitoring rules gives a timely notification to avoid undesired consequences, e.g., non-logged failures, non-fail-stop behavior, failure propagation across sub-systems, etc. The approach does not require any knowledge about the internals of the system under test and it is especially suitable in the multi-tenant environments or when testers may not have a full and detailed understanding of the system.

I investigated the feasibility of our approach in the OpenStack cloud management platform, showing that the approach can be easily applied in the context of an ``off-the-shelf'' distributed system. In order to preliminary evaluate the approach, I executed a campaign of fault-injection experiments in OpenStack. The experiments show that the approach can be applied in a cloud computing platform with high failure detection coverage.

### *Automatic Software Exploit Generation from Natural Language Description*

In the context of software security, a solid understanding of offensive techniques is increasingly important. Well-intentioned actors, such as penetration testers, ethical hackers, researchers, and computer security teams are engaged in developing exploits, referred to as proof-of-concept (POC), to reveal security weaknesses within the software. Offensive security helps us understand how attackers take advantage of vulnerabilities and motivates vendors and users to patch them to prevent attacks.

Among software exploits, code-injection attacks are the trickiest. They allow the attacker to inject and execute arbitrary code on the victim system. Since the injected code frequently launches a command shell, the hacking community refers to the payload portion of a code-injection attack as a **shellcode**. Writing code injection exploits is a challenging task since it requires significant technical skills. Shellcodes are typically written in assembly language, affording the attacker full control of the memory layout and CPU registers to attack low-level mechanisms (e.g., heap metadata and stack return addresses) not otherwise accessible through high-level programming languages. Another challenge for shellcodes is modern antivirus (AV) and intrusion detection systems (IDS), which actively look for malicious payloads to block attacks.

To elude detection, shellcode writers weaponize their shellcode by implementing an encoding/decoding strategy. In other words, writers have to develop encoders (typically, using Python) to obfuscate the original shellcode without altering its functionality, and decoders} (typically in assembly language, as the shellcode) to revert to the payload once it is loaded (and then executed) on the victim system.

In collaboration with the **University of North Carolina at Charlotte** (UNCC), in the United States, I developed an approach for exploit writing based on natural language processing. The approach aims to support both beginners and experienced researchers, by making exploits easier to create and flattening the learning curve. In the approach, a machine learning system that learns about exploit writing from a dataset,

containing both real exploits and their description in the English language. Then, the writer describes the exploit using the English language and lets the machine learning system translate the description into assembly and Python code. The leverages recent advances in *neural machine translation* (NMT) to automatically generate code from natural language descriptions using recurrent neural networks.

NMT has emerged as a promising machine translation approach, and it is widely recognized as the state-of-the-art method for the translation of different languages. NMT has been adopted in many different areas, to generate programs in the Python language, OS commands for the UNIX Bash shell, commit messages for version control, code completion, test cases from security requirements, and more. However, NMT techniques have not heretofore been applied in the field of software security in the manner described in the proposed approach.

I also released substantive datasets containing exploits collected from shellcode databases and their descriptions in the English language. Such data is valuable to support research in machine translation for security-oriented applications since the techniques are data-driven.

# 4. Products

## 4.1 Publications

During the third year, I have produced the following products.

**Conference Paper**

1. Cotroneo, D., De Simone, L., Liguori, P., Natella, R., and Scibelli, A. (2020, December). Towards Runtime Verification via Event Stream Processing in Cloud Computing Infrastructures. In *International Conference on Service-Oriented Computing* (pp. 162-175). Springer, Cham.
2. Liguori, P., Al-Hossami, E., Cotroneo, D., Natella, R., Cukic, B., and Shaikh, S. (2021, August). Shellcode_IA32: A Dataset for Automatic Shellcode Generation. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)* (pp. 58-64). Association for Computational Linguistics.
3. Liguori, P., Al-Hossami, E., Orbinato, V., Natella, R., Shaikh, S., Cotroneo, D., and Cukic, B. (2021, October). EVIL: Exploiting Software via Natural Language. In *2021 IEEE 32th International Symposium on Software Reliability Engineering (ISSRE)* (pp. XX-XX). IEEE.
4. Liguori, P., Improta, C., De Vivo, S., Natella, R., Cotroneo, D., and Cukic, B. Can NMT Understand Me? Towards Perturbation-based Evaluation of NMT Models for Code Generation. In *The 1st Intl. Workshop on Natural Language-based Software Engineering (NLBSE 2022)*, Submitted for Initial Review

**Journal Paper**

1. Cotroneo, D., De Simone, L., Liguori, P., and Natella, R. (2021). Enhancing the analysis of software failures in cloud computing systems with deep learning. *Journal of Systems and Software*, 181, 111043.
2. Liguori, P., Al-Hossami, E., Cotroneo, D., Natella, R., Cukic, B., and Shaikh, S. Can We Generate Shellcodes via Natural Language? An Empirical Study. In *Automated Software Engineering (AUSE)*, Resubmitted after minor revisions

# 5. Conferences and Seminars

## 5.1 Presentations made

During the third year, I have presented the following research products at international conferences:

1. Towards Runtime Verification via Event Stream Processing in Cloud Computing Infrastructures. In *International Conference on Service-Oriented Computing*, December 14, 2021
2. Shellcode_IA32: A Dataset for Automatic Shellcode Generation. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, August 6, 2021
3. EVIL: Exploiting Software via Natural Language. In *2021 IEEE 32th International Symposium on Software Reliability Engineering (ISSRE)*, October 27, 2021
4. Enhancing the analysis of software failures in cloud computing systems with deep learning. *Journal of Systems and Software*, presented at the ISSRE 2021 "Journal First, Conference Second" track, October 28, 2021

## 5.2 Session Chair

At the Workshop on Software Certification (WoSoCer) 2021, co-located with the 32nd International Symposium on Software Reliability Engineering (ISSRE 2021), I have been **session chair** of the session on Design Models, Test Cases, Software Requirements (Session 3, October 25, 2021).

# 6. Activity abroad

Since January 2020 to December 2020, I have been at the **University of North Carolina at Charlotte** (**UNCC**), North Carolina, USA, for my abroad period. At UNCC, I have worked on a new and innovative research area under the supervision of Dr. Bojan Cuckic and Dr. Samira Shaikh. This new activity focuses on the automatic generation of **software exploits** starting from natural language description by using the **Neural Machine Translation** (**NMT**) techniques. The collaboration with the UNCC is still ongoing.

# 7. Tutorship

During the third year, I have been teaching assistant for the course of "Impianti di Elaborazione", ING-INF/05 (Academic Year 2020/2021 and 2021/2022), prof. Domenico Cotroneo.

I have been co-advisor of the following thesis:

1. "*Leveraging Neural Machine Translation to Automatically Generate Software Exploits*", Simona De Vivo, MSc Thesis, Academic Year 2020/21