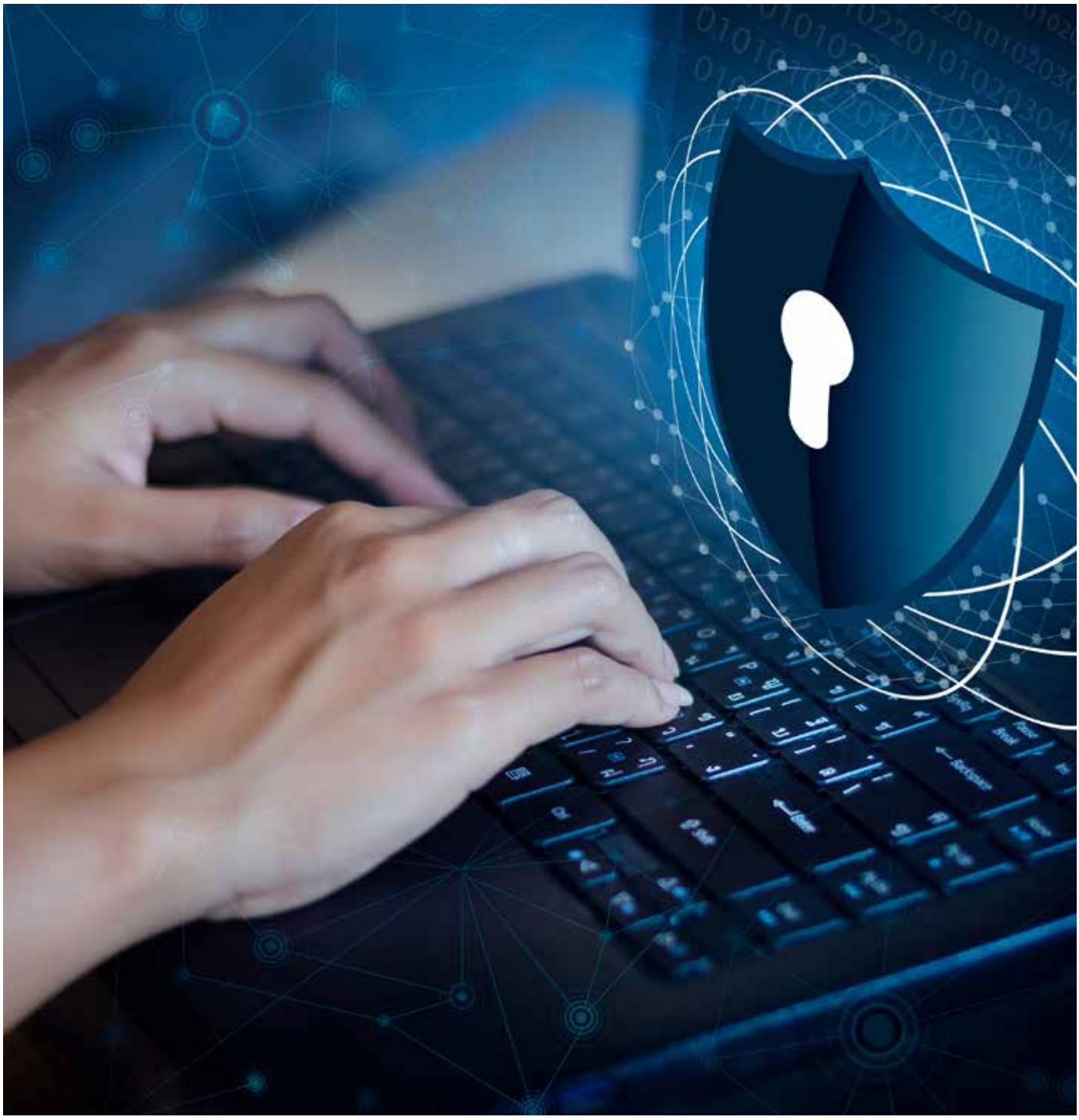
Mphasis
The Next Applied

A Framework for Code Vulnerability Detection

Whitepaper by Abinaya Mahendiran, Assistant Manager, NEXT Labs



Contents

1.

Abstract

Software vulnerabilities pose a serious security risk that can lead to denial of service attack, information leaks, and exploitation of the overall system. Fixing the bugs found in the software is done during the testing as well as the maintenance phases of the software development life cycle. In this paper, we explore how machine learning (ML) and deep learning techniques can be used to assist developers in detecting bugs faster. Common security vulnerabilities are reported almost every day and are maintained in the Common Vulnerabilities and Exposures (CVE) database. ML and deep learning techniques can be deployed on the open source code base and the CVE database to develop a standard framework for detecting code vulnerabilities. Deep learning architectures can be used for feature representation while ML algorithms can be used for classification of good code from bad code.

2.

Introduction

Thousands of security vulnerabilities are discovered every day by researchers and maintained in the Common Vulnerabilities and Exposures (CVE)[3] database. Patches for such vulnerabilities are released by the developer community for both open source and proprietary codes. Fixing such security flaws is essential to avoid exploitation of software systems. Usually, static and dynamic analyzers are used by the developers to detect bugs or vulnerabilities in the software. However, just these methods are inadequate to detect all the security vulnerabilities. ML and deep learning techniques can be coupled with traditional static and dynamic analyzers to detect vulnerabilities in the code. In this paper, we explore how deep learning and ML can be leveraged to automatically detect software vulnerabilities[1].

3.

System Architecture

The following diagram (Fig. 1) gives the overall system architecture of the code vulnerability detection framework. The source code is taken as input to the system. The feature extraction module extracts relevant features while the feature engineering module adds additional features through deep learning models, and language-specific grammars and compilers.

Deep learning models such as RNN (Recurrent Neural Networks), CNN (Convolutional Neural Networks) and GAN (Generative Adversarial Networks) are trained, and the source code is classified as either good code (code without vulnerability) and bad code (code with vulnerability).

The following sections describe in detail the different modules and steps involved in the framework.

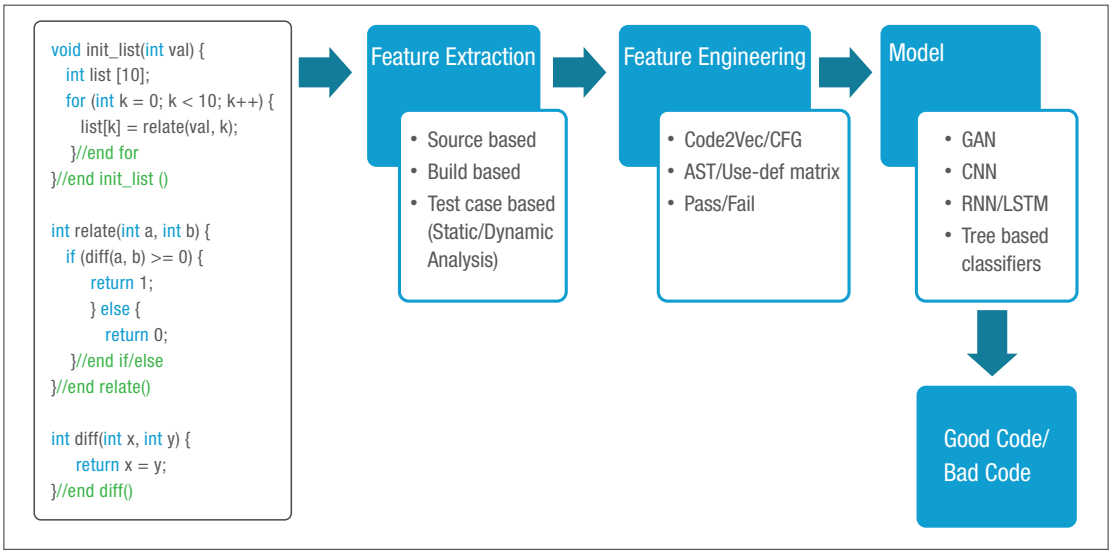


Fig. 1 - System Architecture

4.

Data

A. NIST – Juliet Test Suite

National Institute of Standards and Technology provides Software Assurance Reference Dataset (SARD) dataset that contains known security flaws[6]. The research and the developer community can use this to enhance the quality of the software assurance tools and frameworks that they develop. Juliet Test Suite contains known security vulnerabilities and test cases for programming languages such as C/C++, Java, c#, PHP and web applications. This source code, with or without the security vulnerabilities, can be used as a training and testing set for the framework that we are exploring in this paper.

B. Github

The open source repository like Github[5] can be mined for source code in different technologies such as C/C+, Java, Python and several others. It can be used for training and testing the framework under development.

5.

Data Preparation

For training purposes, the source code from the above-mentioned sources is broken down into functions. Several methods can be combined to label the functions as either good (without any vulnerability) or bad (with vulnerability). A static analysis and dynamic analysis tool can be used to test the source code for any bugs during compile time and run time respectively. Tools such as FlawFinder[4] can be used to check for vulnerabilities. For the open source code taken from Github, NLP techniques can be used on the commit messages to understand if any bugs exist. As an additional step, manual labelling can also be done to cross-check the consistency and accuracy in labelling the functions. Labelled functions become the data to be consumed by deep learning and ML models.

6.

Feature Extraction

From the source code functions, the following features can be extracted that can be used by deep learning and ML algorithms.

A. Source-based features

Features from the source code can be directly extracted using Natural Language Processing (NLP) techniques. A custom lexer can be built to tokenize the given source code. Specific lexers have to be built for different programming languages. The lexer can split the source code into tokens based on language specific keywords, and a dictionary can be built from it. Bag of words vector can be used to represent the tokenized source code.

B. Build-based features

Features can be extracted from the intermediate representation of the source code. Open source compilers like Clang and LLVM (C/C++)[5] can be used to get the Control Flow Graph (CFG) of the source code and the adjacency matrix. Language-specific tools can be used to obtain the CFG for respective programming languages and used as features. Use-def matrix and opcode vector obtained from the compilers can be used as features as well.

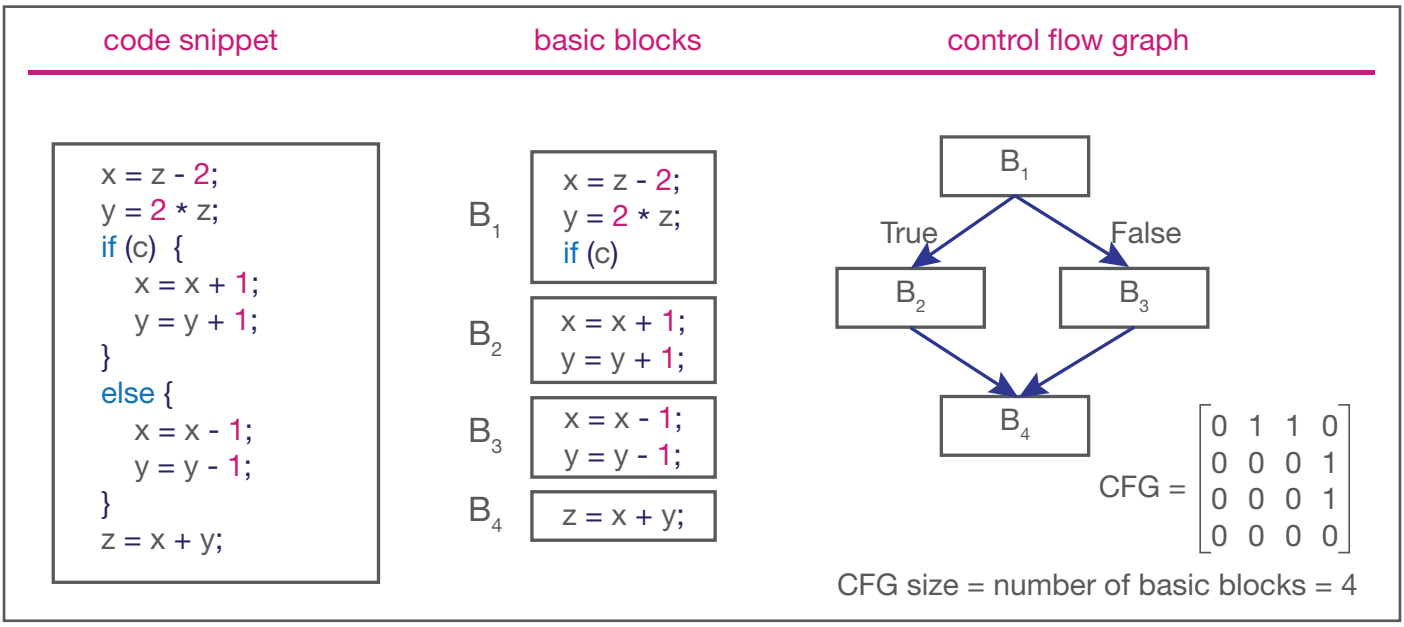


Fig. 2 - Control Flow Graph

C. Test case-based features

Traditionally, test cases are used to test the given code for bugs. Test cases are run against the source code and the status (pass/fail) is used as a feature for ML models.

7.

Feature Engineering

Deep learning techniques can be used to learn better representation of the features. Word2Vec, an unsupervised learning method is used to lean the word vector specific to a programming language. Once the functions are lexed, the tokens can be passed to Word2Vec algorithms which will output a vector of keywords that are contextually similar. For example, all the data type keywords will be close to each other in the vector space (Fig. 2). This Word2Vec representation can be used to explode the feature space representing the source code through the embedding layer in any deep learning architecture.

8.

Models

Once the features are obtained from the methods given above, the labelled source code is converted to sequences based on the lexer. The sequenced source code is then fed into the deep learning and the ML algorithms to predict its vulnerability.

In the following sections, let's look at the several types of models that can be used in this framework.

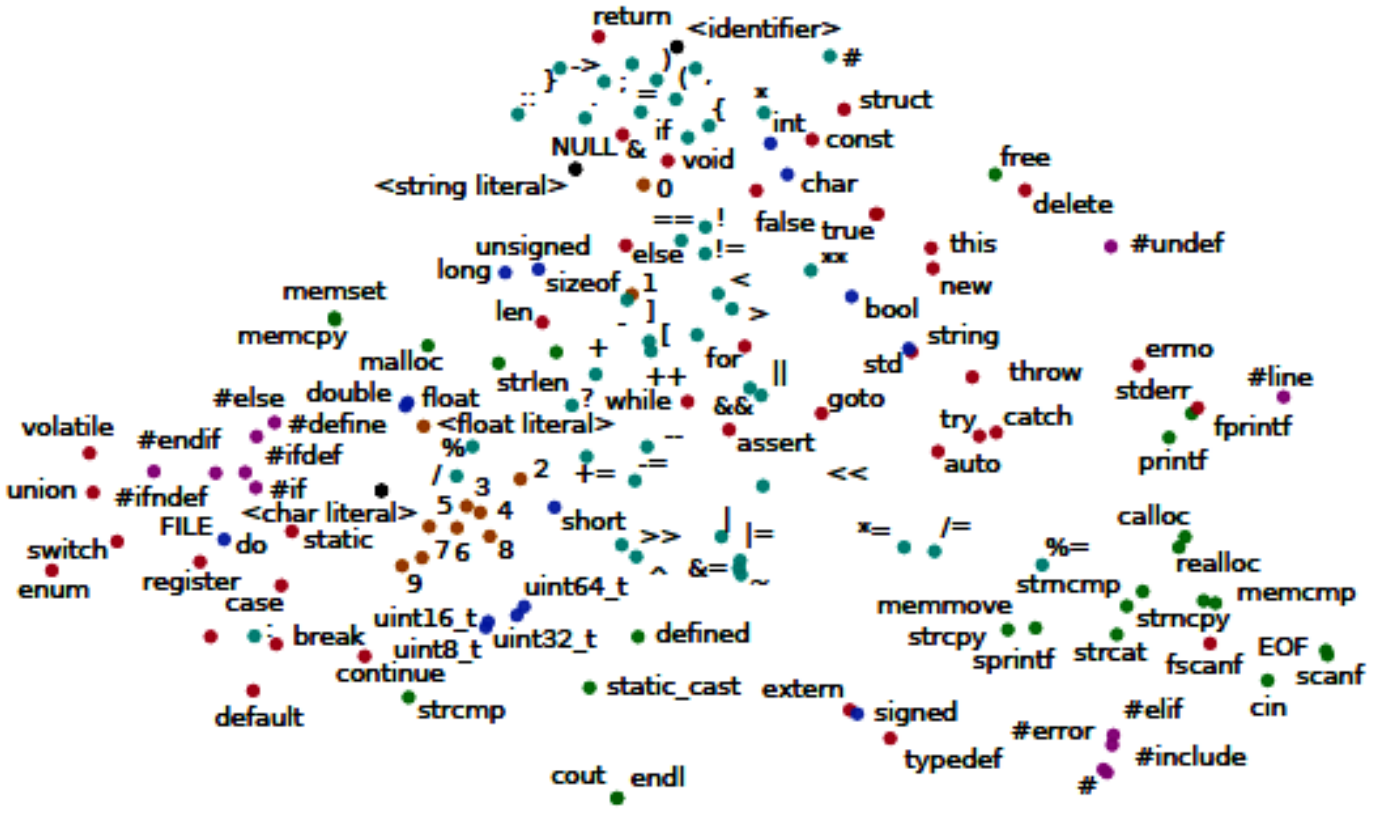


Fig. 3 - Word Vector for the C/C++ Source Code

A. Source-based models

The ‘Bag-of-Words’ and the Word2Vec vectors can be fed as input to a TextCNN[2] (Figure 3) model, which is developed basically for text classification. The CNN model will have an embedding layer that has the learned Word2Vec representation. The embedding layer is followed by several convolutional layers and a max pool layer which learns the succinct representation of the source code. The dense layer at the top of the max pool layer learns the key features.

The TextCNN model is followed by a tree-based supervised classifier like Random Forest which will classify the learned representation of the source code into one of the two classes, namely good code (without vulnerabilities) or bad code (with vulnerabilities).

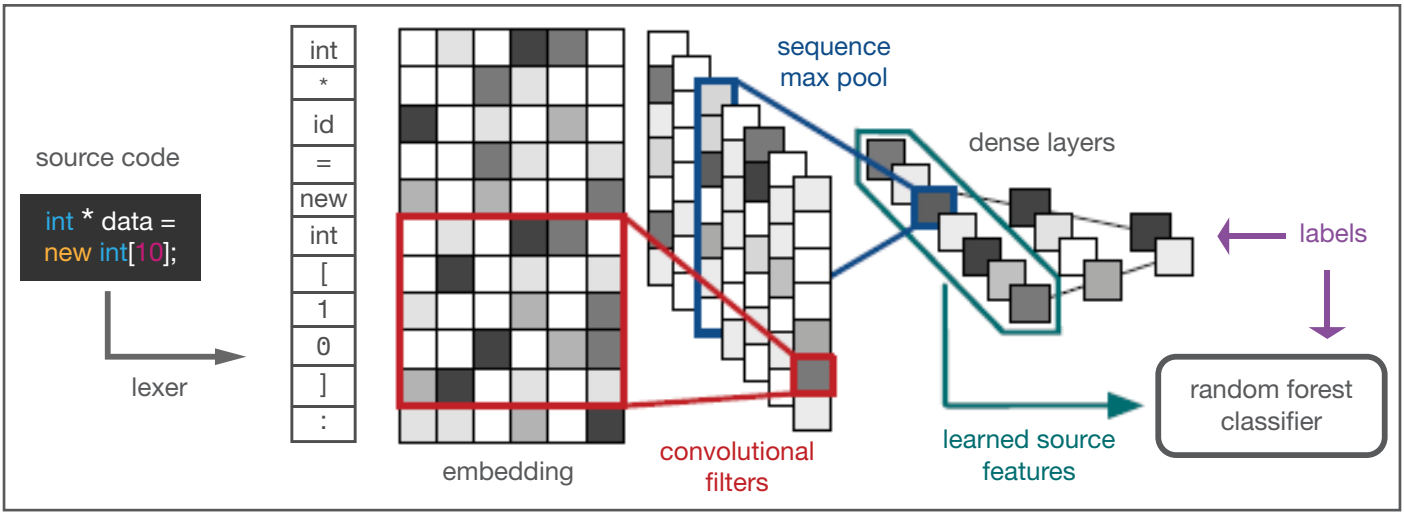


Fig. 4 - TextCNN Model

B. Build-based models

Another kind of model is the build-based one whose features are the CFG adjacency matrix, use-def matrix and the opcode vector that are obtained using the compiler. All these features are converted to vectors and then fed into a tree-based Random Forest classifier for classification.

9.

Conclusion

In this paper, we have outlined various deep learning and ML methods that can be used to detect vulnerability in software. This, in turn, can help reduce the time taken for bug fixing and testing. The methods mentioned have been tried by researchers and found to be efficient. The area of vulnerability detection is an active area of research and we see gradual increase in the contribution of solutions. The next logical step would be to fix the bugs by generating relevant patches for developers to choose from.

10.

References

1. Jacob A. Harer, Louis Y. Kim, Rebecca L. Russell, Onur Ozdemir, Leonard R. Kosta, Akshay Rangamani, Lei H. Hamilton, Gabriel I. Centeno, Jonathan R. Key, Paul M. Ellingwood, Erik Antelman, Alan Mackay, Marc W. McConley, Jeffrey M. Opper, Peter Chin, Tomo Lazovich, "Automated software vulnerability detection with machine learning," arXiv:1803.04497, 2 Aug 2018.

2. KIM, Y. Convolutional neural networks for sentence classification. CoRR abs/1408.5882 (2014).

3. MITRE. Common vulnerabilities and exposures. cve.mitre.org.

4. FlawFinder. https://dwheeler.com/flawfinder/.

5. CLANG. Clang static analyzer. https://clang-analyzer.llvm.org/.

6. Github. Github. https://github.com/.

7. NIST. Juliet test suite v1.3, 2017. https://samate.nist.gov/SRD/testsuite.php.

Author

Abinaya Mahendiran

Assistant Manager at Mphasis NEXT Labs

Abinaya Mahendiran is an Assistant Manager at Mphasis NEXT Labs. She holds a Master's degree in Computer Science with a specialization in Machine Learning and Deep Learning from International Institute of Information Technology Bangalore (IIIT-B). Her research areas include Natural Language Understanding/Processing, Machine Learning, Deep Learning and MLOps. She has an extensive software engineering and data science experience. At NEXT Labs, she has been building and productionizing NLU/NLP solutions for various clients both on premise and on cloud.

About Mphasis

Mphasis (BSE: 526299; NSE: MPHASIS) applies next-generation technology to help enterprises transform businesses globally. Customer centricity is foundational to Mphasis and is reflected in the Mphasis' Front2Back™ Transformation approach. Front2Back™ uses the exponential power of cloud and cognitive to provide hyper-personalized ($C = X2C^{2}_{TM} = 1$) digital experience to clients and their end customers. Mphasis' Service Transformation approach helps 'shrink the core' through the application of digital technologies across legacy environments within an enterprise, enabling businesses to stay ahead in a changing world. Mphasis' core reference architectures and tools, speed and innovation with domain expertise and specialization are key to building strong relationships with marquee clients. To know more, please visit www.mphasis.com

For more information, contact: marketinginfo.m@mphasis.com

USA
460 Park Avenue South
Suite #1101
New York, NY 10016, USA
Tel.: +1 212 686 6655

UK
1 Ropemaker Street, London
EC2Y 9HT, United Kingdom
T : +44 020 7153 1327

INDIA
Bagmane World Technology Center
Marathahalli Ring Road
Doddanakundhi Village
Mahadevapura
Bangalore 560 048, India
Tel.: +91 80 3352 5000

www.mphasis.com

NH 19/03/20 US LETTER BASIL6042