Study on Different Code-Clone Detection Techniques & Approaches to MitigateCode Reuse Attacks

Bhaumik Tyagi, Avishek Chaudhary, Dr. Amrita

Department of Computer Science & Engineering, Sharda University

---***---

Abstract - *Code clones in software development are types of fragments of code that must be identified by using a clone detection tool. This paper discusses reviews on different code-clone detection techniques, code reuse or cold cloning issues, and ROP (Return Oriented Programming). Review of "clone detection precision using machine learning techniques" which thereby seeks to eliminate false-positive clone classes outlinedby a clone recognition tool. Pyclone: "A Python code clone test bank generator" which testifies a new tool that will take a kernel of 'source code'(python) & give rise to Type1, Type2, and Type3 code clones in python. After reviewing these papers, we have found some research gaps that are briefly mentioned in this review paper. Code clones or code reuse could create a serious issue when it comes to software maintenance, testing, and debugging also makes the system vulnerable so, it may be easily exploited by unauthorized people. Also, it culminates that there exist numerous types of research to identify type1, type2, type3, and type4 clones. However, there is a necessity to remodel new methodologies using proper tool support in order to discover all types of emulations cooperatively and mitigate code cloning and code reuse attacks. Moreover, it is also essential to propose more methodologies and techniques to streamline the expansion of Program Dependency Graph (PDG) while dealing with the recognition of type4 clones. Also, cloning issues like exploitation of code, and code reuse attacks are discussed along with approaches to mitigate code reuse attacks.*

Keywords* - *ML, Clone Detection, Decision Tree, AST, Code Reuse, ROP (Return Oriented Programming), Malware

I. Introduction

In the software development industry, the recent trend is to reuse existing code [17], libraries, components, etc. by replication,and fixing fragments of source code is ageneric pursuit. The outcome of theseactivities is replicated code i.e., code clones. Code clones introduce issues in the form of bugs, and complexity in software maintenance. The developer's habit of replicating code, rather than refactoring code, leads to code clones in programs. Cloning and reusing of code are similar terms in software development. The developer performing the code reuse task is not aware of further complexity, which is much more difficult to handle.

In this busy world everybody wants to complete their task on time and because of peer pressure developer also takes the easy route to accomplish the development of the software by using code cloning and code reusing. Also, nowadays most of the shared libraries and components are already available by the framework and compiler. If similar software is already developed in this case, we try to reuse existing code and introduce new software.

Above all the scenario is hands down to develop the software but it may introduce a large number of issues and vulnerabilities to the software where e hacker or another unauthorized person who is used to those libraries and know to flow of code can easily exploit the software. And that is not good practice for software development.

Recently, the log4j library, one of the most popular libraries of java, was exploited by a malicious code injection by a hacker. This negatively impacts development, and other java packages were also at high risk of disclosure.

Code reuse attacks and code cloning are one of the major concerns in software development practices nowadays. It is important to mitigate code reuse attacks and code cloning by using randomization techniques and early recognition of code replicas during the development of code.

This is a well-known research topic in software development intending to detect such replicated fragments of code in the software.

Also, if two code fragments are similar with minor modifications they are called code clones. These code clones can cause troublein the software maintenance and debugging process.

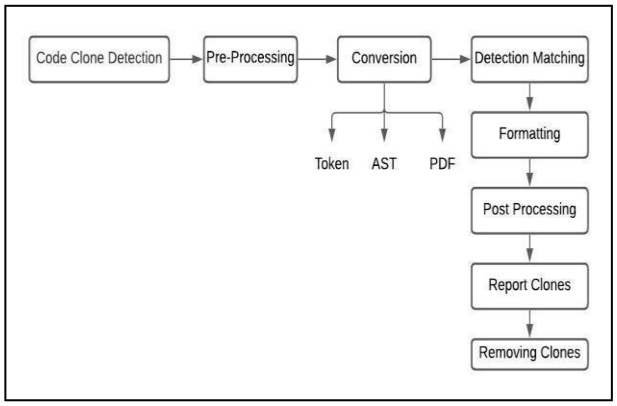

Fig 1: Life Cycle of Code Clone Detection

Types of clones:

Type-1: includes clones that are identical code snippets, neglecting whitespaces and comments.

Type-2: include clones with only disparities in using parameters & formatting style. (i.e., literals, type, literals, variable, function).

Type-3: include clones with variations including modified statements.

Type-4: include clones that don't comprehend the identical structure (syntactical more precisely) and still instigate similar functionalities.

Original code	**Type 1 code**	**Type 2 code**	**Type 3 code**	**Type 4 Code**

int x = 9;
int y = 0;
// Com
 	
ment
while(
y<=x)
{ y++;
}
	int x =9;
	int y
	=0;
	while(
	y<=x)
	{ y++;
	}
		int a =
		9;
		int b =0;
		
		while(a
		<=b) {
		b++;
		}
			doublea =
			9;
			doubleb =
			0;
			while(b
			<=a) {
			b +1 =
			1.0;
			}
				double
				a =
				(18/2);
				doubleb =
				0;
				while(b
				<=a) {
				
				b +1=
				1.0;
				}

In this paper, we will review two techniques of clone detection and mitigation of code reuse attacks and briefly discuss their flaws.

Beginning with Pyclone, which was composed to generate code clones based on a kernel of python files conceded to it. With the vision of mutation of AST (Abstract Syntax Tree) is created next to the kernel files.

In a machine learning-based technique, the author picked "19 clone class metrics" that depict diverse descriptions of cloned and non-cloned classes and a decision tree binary classifier to facilitate filtering out clone classes from the original clone result given by a clone detection tool. Here, A supervised learning algorithm "DecisionTree Algorithm" is used that produces decision nodes via the information gain attained from the value of each feature. The classification is made by accepting the data through the tree from the top to a leaf node.

II. Related Work

a) ***"Pyclone":*** This can create a documented no. of code clones centered at the objective of transformation of Abstract Syntax Tree (AST) created from the kernel records. In this paper, the very first step is to examine the kernel files & create ASTs for the same. "An AST is basically a tree representation of abstract code structure. If the code is dissimilar amid the two projects, the ASTs will also contrast."

In order to compile the python code into bytecode, python uses a compiler associated with AST. A Python package called Astor is also used by Pyclone. Astor permits Pyclone to develop ASTs based on justifiable Python files and alter the same.

class P:
 	
def_init_(self,
x):
 	
self_x = x
def get_x(self):
return self_x def
set_x(self, x):
 	
self_x = s
	class P:
	
	def_init_(self, x):
	
	self_x = x def
	get_x(self):
	return self_x

Fig 2: Cloning a Class Method

Here, the generation of Type 3, Type 2, and Type 1, has been accomplished by using the Abstract Syntax Tree method.

b)"Clone detection using Machine Learning",19 clone metrics are chosen that accumulate diverse emulated and non-emulated classes.

The methodology of this research consists of an "Experimental framework, Dataset and clone validation, error measures, Model Training, Tuning, and evaluation of the clone filter on clones in another language."

There are two Research Questions in this paper. The first is the effectiveness of a Machine Learning clone filter to advance clonedetection precision & the second is the effectiveness of applying a Machine Learning clone filter directed from one to another language.

The ML clone filter is efficient in refining clone recognition precision. When integrated decision tree filter into i-clones displays that it can improve i-clones accuracy from 0.94 to 0.98.

A large training data set is required in the direction to generate a more comprehensive clone filter. Also, the developed metrics may not efficiently capture the features of clones in a different language.

Original precision = 0.94, Filter's precision

= 0.98.

c)"Search-based software engineering (SBSE) in clone- detection optimization",

To find a set of parameter values, two approaches used SBSE that maximizes the contract amongst a collective of clone detection tools.[7][8] Also, Eva-Clone, is an approach using a genetic algorithm to discover the alignment space of clonedetection tools and to achieve the best parameter backgrounds. However, Eva- Clone gives undesirable results in terms of clonequality.

d) "Tree-based clone detection", creates an abstract syntax tree *(AST)* for each code fragment and then leverages tree matching algorithms to detect similar subtrees.[9]

e) "Deep-learning-based-research" *in software engineering,* researchers have recently used deep learning to solve problems in software engineering.[12],[13],[14],[15]

Nevertheless, they did not compare their methodology with a prevailing clone detection technique using any renowned clone benchmark.

III. Research Gap

In a study of Pyclone, the whole system works on the injection-based framework and due to the class of this framework, it could be probable that by injecting novel code or new lines of code or removing them, an accidental clone could be established with an unintended code fragment.

As, "Type 1 and Type 2 clones" are forthright, as they are nearly indistinguishable replicas of the novel code. Nevertheless, the detection of "Type 3 or Type 4 clones" might hinge on the performance of the tools for retrieving clones, and the specified tool may not define "Type 3 or Type 4" as similar as the way the tool has generated them, with thisclone may be misidentified or not caught at allin case of Type 3, Type 4 in case of using Pyclone.

In our study of the Machine Learning Technique, the author only focuses on replicas of Type 1 to Type 3 because of their syntactic similarity. But the main problem is with Type 4 which is spotless in this paper. Moreover, the finding of this very research is centered only on Python "open-source project" (Django) & Java "open-source project" (J Free Chart). Also, the efficiency of the filter is associated witha Decision Tree Model. Dataset or Datapoints should be increased to instruct and assess the model by using more balanced & larger ground reality data, moreprecise models are normally anticipated.

Instead of using a Decision Tree, more sophisticated techniques like Random Forest should be taken into consideration. It might present an improved implementation thanthe present model.

IV. Code Cloning Issues

a) ***Vulnerability of Code:*** Vulnerability is considered as the major concern in software development which is introduced in the code because of bad coding habits, practice, not following coding standards, and also using the code block without knowing the functionality of that block of code.

b) ***Code Reuse Attacks:*** Another major issuein software development is caused by the copying of malicious code or the libraries in the software. ROP (Return oriented programming) is a technique where an attacker introduces a set of instructions called gadgets to exploit the code. It is depending on the flow of memory design where code is executed.

c) ***Exploitation of Code:*** Malicious code leads to exploit the software by using exploitation techniques hacker hacks or theft the important information from the user's system. This is also another major issue that is caused by code cloning.

d) ***Complex to Manage Code:*** Code cloning is not only introducing the major issue or vulnerability in the system/software but also makes complex to manage the software in the future. Hard to debug and track the bug in the system. Cause the code is not written by the concerned developer.

V. Approach to Mitigate code Reuse Attack

1. ***Code Randomization:*** Code Randomization consist of two phases the first phase also consists of two-part one is used to extract the information from function blocks and flow control, and the second part is used to separate the code segment. The second phase makes shuffles the block of code and function.

2. ***Optimization of Code:*** It is a technique to modify the code to reduce the size code, and consume less memory during execution. By detecting the code cloning level, we can able to optimize and refactor the code so, complexity and vulnerability can be resolved.

3. Other techniques which are used to reduce the code reuse attacks are: ILR[18], MARLIN [17], STIR [19], XIFER [21], DROP [20], CFL [24], CCFIR, ROPDEFENDER [23]

VI. Conclusion

Type 3 - Type 4 clones are much more complex to weight and resolve as well. As, in Pyclone, the filter was fairly suitable for "Type 1 and Type 2" replicas in contrast to "Type 3 and Type 4". Also, using an injection-based framework is unsuitable if we want to avoid accidental or unintended clones. In, the ML technique, Pyclones illustrates that the filter was not efficient in the other programming languages & future work is required proceeding this problem.

Also, by using the Randomization technique and optimization technique which are mentioned above we can able to reduce the code reuse attacks and early detection of code cloning which helps to refactor the code and reduce the complexity.

By increasing the data points, we can get more accurate results, and also by using a more sophisticated technology we can increase the performance and accuracy of our model. In the software industry, there are tools out there that can catch Type 3 clones effectively but still Type 4 is a big issue in the software world.

VII. References

[1] Qurat Ul Ain1,Wasi Haider Butt1, Muhamad Waseem Anwar1,Farooque Azam1,AndBilalMaqbool1RecentAdvancem ents inCode Clone Detection Techniques and Tools

[2] Schaeffer Duncan1, Andrew Walker1, Caleb DeHaan1, Stephanie Alvord1,Tomas Cerny1, and Pavel Tisnovsky2 Pyclone: A Python Code Clone Test Bank Generator

[3] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng and Barbara Ryder CCLearner: A Deep LearningBased Clone Detection Approach

[4] Vara Arammongkolvichai, Rainer Koschke, Chaiyong Ragkhitwetsagul, Morakot Choetkiertikul, Thanwadee Sunetnanta Improving Clone Detection Precision using Machine Learning Techniques

[5] Hannes Thaller, Lukas Linsbauer, Alexander Egyed Towards Semantic Clone Detection via Probabilistic Software Modeling

[6] HAIBO ZHANG 1 AND KOUICHI SAKURAI 2 " A Survey of Software Clone Detection From Security Perspective"

[7] T. Wang, M. Harman, Y. Jia, and J. Krinke, "Searching for better configurations: A rigorous approach to clone evaluation," in Proceedings of the 2013 9th Joint Meeting onFoundations of Software Engineering

[8] C. Ragkhitwetsagul, M. Paixao, M. Adham, S. Busari, J. Krinke, and J. H. Drake, Searching for Configurations in Clone Evaluation – A Replication Study. Cham: Springer International Publishing, 2016

[9] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in Proceedings of the International Conference on Software Maintenance, 1998.

[10] [10] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, DECKARD: scalable and accurate tree-based detection ofcode clones, 2007.

[11] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big code,"

[12] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, Combining deep learning with information retrieval to localize buggy files for bug reports , 2015

[13] S. Wang, T. Liu, and L. Tan, Automatically learning semantic features for defect prediction, in Proceedings of the International Conference on Software Engineering, 2016.

[14] X. Gu, H. Zhang, D. Zhang, and S. Kim, Deep API learning, in Proceedings of the ACM International Symposium on Foundations of Software Engineering, 2016

[15] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering ,2016

[16] G. E. Hinton, S. Osindero, and Y.W.Teh, "A fast learning algorithm for deep belief networks," Neural Computing, 2006.

[17] A. Gupta, J. Habibi, M. S. Kirkpatrick, and E. Bertino, "Marlin: Mitigating Code Reuse Attacks Using Code Randomization," in IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 3, pp. 326-337, 1 May-June 2015, DOI: 10.1109/TDSC.2014.2345384.

[18] J. Hiser, A. Nguyen-Tuong, M. Co,M. Hall, and J. W. Davidson, "ILR: Where'd my gadgets go?" in Proc. IEEE Symp.Security Privacy, 2012, pp. 571–585.

[19] R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin, "Binary stirring: Self- randomizing instruction addresses of legacy x86 binary code," in Proc. ACM Conf. Comput. Commun. Security, 2012, pp.157–168.

[20] P. Chen, H. Xiao, X. Shen, X. Yin B. Mao, and L. Xie, "DROP: Detecting return-oriented programming malicious code," in Proc. 5th Int. Conf. Inf. Syst. Security, 2009, pp. 163–177.

[21] L. V. Davi, A. Dmitrienko, S. Nurnberger, and A.-R. Sadeghi, "Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm," in Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Security, 2013, pp.299–310.

[22] V. Pappas, M. Polychronakis, and A. D. Keromytis, "Smashing the gadgets: Hindering return-oriented programming using in-place code randomization," in Proc. IEEE Symp Security Privacy, 2012, pp. 601–615.

[23] L. Davi, A.-R. Sadeghi, and M. Winandy, "ROPdefender: A detection tool to defend against return-oriented programming attacks," in Proc. 6th ACM Symp. Inf. Comput. Commun. Security, 2011, pp. 40–51.

[24] T. Bletsch, X. Jiang, and V. Freeh, "Mitigating code-reuse attacks with control- flow locking," in Proc. 27th Annu. Comput. Security Appl. Conf., New York, NY, USA.