UNIVERSITY OF PUERTO RICO
RECINTO UNIVERSITARIO DE MAYAGÜEZ
MAYAGÜEZ, PUERTO RICO
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Project Phase One: TITANIUM-CODE: Programming Language for Secure Coding

INEL 4036 - 096 : Programming Languages
Prof. Wilson Rivera

Kensy Bernardeau
Luis Diaz
Noel Valentín
Dong Wang (Robbie)

1. Introduction

Secure coding standards provide rules and recommendations to guide the development of secure software systems [1]. These standards ensure that the system will be secure in the developing phase; it also provides a common set of criteria that can be used to measure and evaluate software development efforts and software development process [2]. This proposal will describe motivations, reasons, language features, and the general structure of the programming language that will be developed. By using this language, we will try to design a functional system to implement security checkpoints that would detect code that could lead to a security flaw when the program is being complied.

The motivation for the development of this language is to apply the secure coding standards towards the programmers. Coding standards that most programmers use are incomplete and not security centric. With this language, we can have it running security checks during compilation time such that when it detects any security flaw, the source code ceases to compile and an error is thrown.

Titanium-Code, the programming language under development, looks to be a tool that regulates and facilitates the prevention of these security flaws. Secure Coding should be something that every programmer should be aware of, thus there should be general standards set to ensure a safe code. For this purpose, this programming language strives to do just that.

2. Language Features

This language will serve as a checkpoint before compilation: it will "read" the code, and will prevent the code from being compiled if it detects any code fragment that could indicate an eventual security flaw in the system. It should detect vulnerabilities, such as buffer overflow, or a format String Exploit, among others. It should then cancel compilation and warn the user that there is a vulnerability in a specific fragment their code, and would indicate how to best deal with the vulnerability. Therefor, the resulting code can have a secure structure, and the user (the programmer) can better understand and begin to code more carefully, avoiding the security risks that could arise.

Such features to be proposed include:

- Compiled
- Functional Paradigm
- Security Check during Compilation
- Core Security Functions

3. Example of a program

An implementable security vulnerability check is to validate if a given variable is within the acceptable bounds of the type to avoid a buffer overflow. This issue could be caused when a variable with an unacceptably large value overwrites adjacent memory locations. The code can check the value of a variable and prevent the user from going forward with it if its value exceeds the bounds established.

The language should allow programmers to choose the targeting vulnerabilities the code looks for from a range of options. Since the main objective of program is to read code and look for security risks, the user should also give the code to be checked as input for the program. The language of the user's input is also important; while Python is priority, it could be extended to various other programming languages with different syntax, which would cause different strategies in detecting security flaws.

As it was mentioned above, some simple vulnerability checks like the buffer overflow occur if a variable surpasses a specific bound or maximum value. For situations and security flaws like this, the user can also specify the range of values that is acceptable for a certain variable. The program would then warn the user and prevent compilation, then informing the programmer that there is a risk in the code that was written (in a specific case, Buffer Overflow).

The syntax of Titanium-Code, while it is not final, could follow this general structure:

```
//Create a CodeMonitor that has appropriate
functions

CodeMon = new CodeMonitor()

//Specify Language of the input code and include the
input file that will be evaluated. It could also be
a .txt file or a String

CodeMon.AddCode(file = "/Downloads/Pong99.py")
CodeMon.Lang(language = "Python")

//Add in the security risks that the program should
watch out for, and set any parameters that could be
needed. For example, the lowbound and highbound.

CodeMon.BufferOverflow(lowbound = 0, highbound =
300)
CodeMon.SensitiveInfo(password = "123456")
CodeMon.CodePointerExploit()
```

We can write a code that calls a command that does one or more of the following security coding techniques:

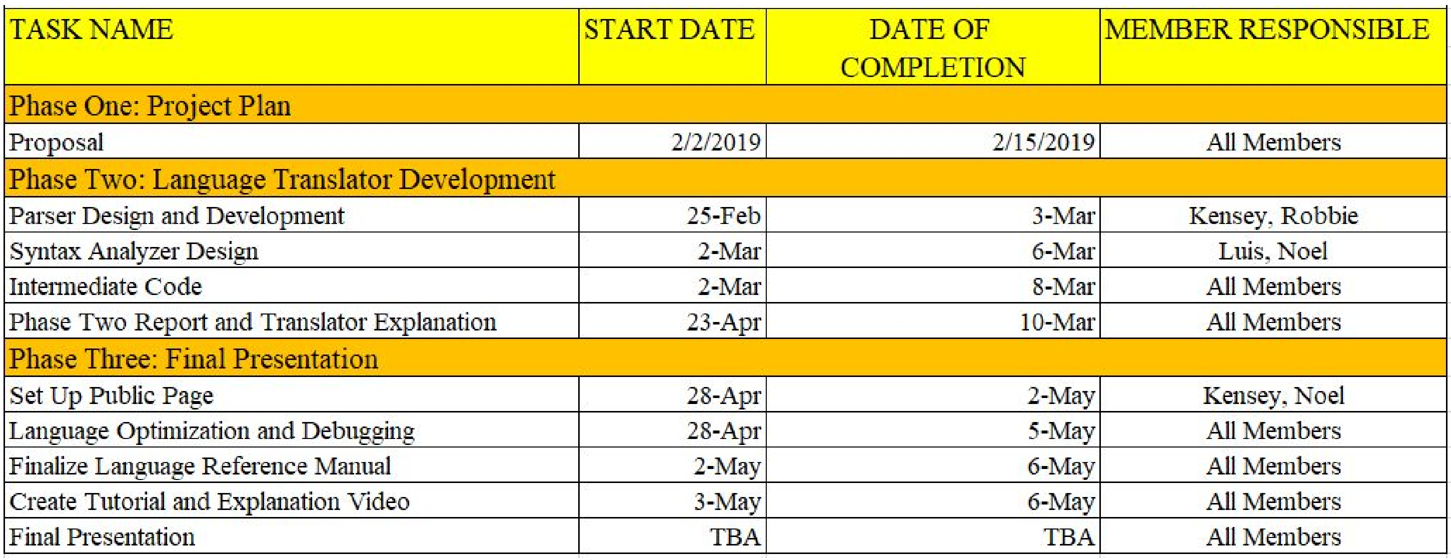
1. **Validate input from all external data sources including user controlled files, network interfaces, and command line arguments.**
2. **Have a layered defence with multiple strategies. If one does not prevent a vulnerability, the next layer of defence should.**

3. Implementation requirements and tools

Titanium-Code will use Python as its base, given that it is a relatively simple language to create our language as a proof of concept; because the main objective of the program is to create tools for detection of suspicious tendencies in programming. Therefore, a simpler program would be more advantageous.
-Parser: ANTLR
-Language: Python

4. Project Plan and Timeline

The program must then be finalized and presented with a final report, public page and instructional video by the end of the semester. However, several interpreter software components must be finalized at an earlier date. The timeline is as follows:



TASK NAME	START DATE	DATE OF COMPLETION	MEMBER RESPONSIBLE
Phase One: Project Plan			
Proposal	2/2/2019	2/15/2019	All Members
Phase Two: Language Translator Development			
Parser Design and Development	25-Feb	3-Mar	Kensey, Robbie
Syntax Analyzer Design	2-Mar	6-Mar	Luis, Noel
Intermediate Code	2-Mar	8-Mar	All Members
Phase Two Report and Translator Explanation	23-Apr	10-Mar	All Members
Phase Three: Final Presentation			
Set Up Public Page	28-Apr	2-May	Kensey, Noel
Language Optimization and Debugging	28-Apr	5-May	All Members
Finalize Language Reference Manual	2-May	6-May	All Members
Create Tutorial and Explanation Video	3-May	6-May	All Members
Final Presentation	TBA	TBA	All Members

References:

[1] F. Sheldon, A. Krings, S. Yoo, and A. M. Editors, “Third Annual Cyber Security and Information Infrastructure Research Workshop COMPREHENSIVE STRATEGIES THAT MEET THE CHALLENGES OF THE 21ST CENTURY,” 2007.
[2] R. W. Yeung, “Secure Network Coding,” p. 7803, 2002.