An Approach towards Secure Programming in Undergraduate Computing Curricula

Full Paper

SACLA 2019

© The authors/SACLA

Preprint

Sifiso Bangani[1] [0000-0001-9550-3185], Lynn Futcher[2] [0000-0003-0406-8718], Johan Van Niekerk[3,4][000-0003-1739-4563]

[1,2,3] Nelson Mandela University, Port Elizabeth, South Africa
{s214098389, Lynn.Futcher, Johan.VanNiekerk}@mandela.ac.za
[4] Noroff University College, Norway
johan.vanniekerk@noroff.no

Abstract. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures and loss of information. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. Many software developers are not competent in secure programming, resulting in risks that are caused by vulnerabilities in the application code of software applications. Although there are various techniques for writing secure code in the current body of knowledge, these techniques are rarely fundamental components of a computing curriculum, resulting in incompetent graduate software developers. This paper argues that secure programming education needs to be included across computing curricula. It proposes the incorporation of secure coding practices into undergraduate computing curricula through a step-by-step approach. This approach includes the identification of application risks and secure coding practices as they relate to each other and to fundamental programming concepts. It specifically aims to improve the security of software applications developed in the .Net environment.

Keywords: Computing Curricula, Software Security, Application Risks, Secure Coding Practices, Fundamental Programming Concepts.

1 Introduction

As the world advances in technology by creating new and exciting software applications, so does the need to protect these software applications as their vulnerabilities and associated risks also increase. Software applications have become integral to billions of people as they use them on a day-to-day basis for working with top-secret enterprise intellectual property, sharing personal information, making bank transactions and sharing pictures with family and friends [1].

Although software plays an important role on a day-to-day basis, it often has associated risks as a result of vulnerabilities in the application layer [2]. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures, and loss of information [3]. Failure to secure software solutions can have more serious effects than just a temporary interruption to a service. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. The human factor, which includes the programmer, has a major impact on the success and failure of efforts to secure and protect the business, services, and information [4]. According to [5], the main cause of software application failure is human error in application programming, which happens during the coding process.

Software developers are typically equipped with relevant programming knowledge and skills to develop innovative software [6]. However, software developers are rarely equipped with secure programming knowledge and skills from the undergraduate level [7]. According to [8], *"Students graduating from technical programs such as information technology often do not have the attributes to fill the needs of industry"*. Fundamental programming principles are often introduced to students without an understanding of their security implications, resulting in non-adherence to secure programming [7]. For example, arrays and loops are introduced and explained without the mention of buffer over flows that could occur due to lack of adherence to secure programming.

The purpose of this research paper is to argue that secure programming education needs to be included across computing curricula. Secure programming is an important part of information security education, as [9] argue that relevant topics of information security must be taught to some extent, in all of the modules of the main curriculum from the first year of study, through to the final year of study. The contribution of this paper is five-fold:

- Firstly, it identifies relevant application risks in the .Net environment.
- Secondly, it identifies secure coding practices to be taught to undergraduate computing students.
- Thirdly, it determines the basic programming concepts taught in the .Net environment in South African undergraduate computing curricula.
- Fourthly, it maps the basic programming concepts to relevant application risks
- Finally, it maps the relevant application risks to the identified secure coding practices.

These mappings help us understand how secure programming education can be incorporated into undergraduate computing curricula. By computing curricula, this research refers to university courses that teach programming with a focus on Computer Science and Information Technology.

Preprint

2 Related Literature

As much as new software technologies are needed and are being developed, the industry increasingly demands software developers that possess relevant security knowledge, skills and abilities [8]. Advancements in technology also increases the security risks associated with those technologies, creating a gap of outdated knowledge and skills for industry and academia [10]. According to [8], "*although [cybersecurity] jobs are and will be available, employers find it increasingly difficult to find qualified people to fill them. Students graduating from technical programs such as information technology often do not have the attributes to fill the needs of industry*". Software security is becoming every company's norm and concern as a result of the rising trend of software application vulnerabilities [1, 10, 11], which is the driving force behind the demand for software developers with security knowledge and skills. This security skill demand results in industry's need to hire developers experienced in secure programming. These developers must have the knowledge, skills, and abilities of secure programming that enables them to implement security-related solutions.

The security skills demand often forces companies to enroll their employees in secure programming certifications such as, IBM's Application Security Analyst Mastery Award, and Microsoft's Software Development Fundamentals course. These certifications are an attempt to make software developers competent in secure programming, as they often lack the required knowledge [12, 13]. However, the knowledge acquired through certifications is not sufficient to be productive in secure programming without the necessary skills, as there should be a balance between knowledge and skills [14]. Secure programming certifications and training focus on two primary factors, namely: awareness of a specific security threat, and having adequate training in the use of the security counter-measure to such a threat [15]. However, these certifications and training do not guarantee a change in human behaviour [4, 15], as human behaviour requires more than just awareness of a specific security threat. For software developers to be competent in secure programming, they must be trained on the requisite skills of secure programming at an undergraduate level.

Producing competent software developers should therefore begin in universities and colleges where students are being educated in understanding and applying learned concepts, and the ability to work in a team environment [10, 16]. Universities are responsible for providing a hands-on teaching approach for undergraduate students, which includes classroom lecturing, computer laboratory practical classes and experiments [2, 14]. The fundamentals of computing are introduced to learners at university level, where learners are educated and guided through computer laboratory practical classes. The learning outcomes from university curricula are used to show what the students will know, and be able to demonstrate after the completion of that course [10], and are key to the shift of focus in education from a paradigm concerned with providing instructions, to a paradigm of producing learning [17].

Various computing curricula guidelines such as the Association for Computing Machinery (ACM) state that, Information Assurance and Security (IAS) belong at an advanced level of a four year computing program, yet many students in three year computing courses graduate and leave university before completing the fourth year of study

Preprint

[9]. Furthermore, [9] argues that "*Therefore, for information security to become pervasive, relevant topics could be taught, to some extent, in all of the modules of the main curriculum from first year through to the final year*". Programming is fundamental to computing curricula. However, often not much attention is given to secure programming. Therefore, students can only apply what they have been taught. The behaviour of a student in a certain area such as secure programming can be improved by providing students with the requisite knowledge [18]. This can be done through software security education in computing at universities.

Software security is the idea implemented to protect software to ensure it functions correctly under malicious attacks [19]. Furthermore, [19] states that "*Software security is about building secure software: designing software to be secure, making sure that software is secure, and educating software developers, architects, and users about how to build secure things*". Software security is not simply implemented by installing an anti-virus software to a computer or electronic device, as hackers steal or get access to top secret enterprise information, or even damage the behaviour of software applications. Hackers can damage software through embedding malicious software or scripts in the code. Software applications without proper security built-in can be vulnerable to various computer attacks such as Cross-Site Scripting, SQL Injections, Session Hijacking, Cross-Site Request Forgery and Denial of Service attacks [2]. The only way to avoid such attacks is by practicing good secure programming techniques [5, 20].

Secure programming is the manner of writing code to minimise software security vulnerabilities, as many problems faced by users nowadays are caused by vulnerabilities resulting from flaws in application code. There are various techniques for writing secure code in the current body of knowledge [7, 21, 22]. Although these techniques exist, they are rarely fundamental components of a computing curriculum, but rather treated as secondary topics that are briefly discussed in programming courses [7]. To maintain security in software applications, students must have the necessary skills and knowledge. According to [23] "*the ability to write secure code should be as fundamental to a university computer science undergraduate as basic literacy*". This research proposes the explicit incorporation of secure programming practices into undergraduate computing curricula. The following section briefly describes computing education in the South African context.

Preprint

3 Computing Education in the South African Context

Institutions of higher learning in South Africa are divided into public and private universities [24]. For the purposes of this research, the focus is on public universities. South African public universities are divided into three categories namely: traditional universities, universities of technology and comprehensive universities. The South African public universities are overseen by the Department of Higher Education and Training, which is responsible for post-school education and training.

South African universities offer three to four year degree qualifications depending on the type of university [25]. Comprehensive universities and universities of technology offer three year diploma qualifications, where a student can graduate and leave university with a diploma to join the workplace [9]. For a student in such universities to obtain a degree, the student would be required to advance their diploma qualification by completing their fourth year of study. The fourth year of study is considered to be an advanced level, where students can be introduced to advanced topics [10]. In the case of programming qualifications, the fourth year of study would typically include an introduction to security and secure programming basics [9, 10]. Traditional universities with three year degree qualification teach fundamental programming basics in the undergraduate level. The fourth year of study in traditional universities is also considered as an advanced level, where students are introduced to advanced computing topics.

South African universities offer semester courses and year courses. Semester courses are usually carried out over a period of six months, and year courses are carried out throughout the year [24]. Both semester and year courses in various universities offer fundamentals of programming. However, secure coding practices are rarely explicitly taught to undergraduate students, but are rather treated as secondary topics that are briefly discussed in these programming courses [7]. Examples of such courses include: Programming Fundamentals, Computing Fundamentals, Development Software, Applications Development, Mobile Computing, Technical Programming and Web Systems.

The focus in this research is on applications developed in the .Net environment, since most South African universities teach programming in the .Net environment, with Microsoft promoting free product usage by university students. However, the approach for incorporating secure coding practices into undergraduate computing curricula can be used in other development environments and frameworks that are not .Net based.

4 Research Approach

A preliminary investigation and content analysis were conducted, to determine whether South African universities incorporate secure programming in their undergraduate computing curricula, in an effort to ensure that students will be competent in the secure programming of software applications. The preliminary investigation was conducted on South African universities through a thematic content analysis, by reviewing the Prospectus and Learner and Lecturer Guides of various universities. Where relevant themes and topics relating to secure programming were examined. The purpose of the investigation was to determine whether secure programming is being included in the teaching of programming concepts, as writing secure code is fundamental to an undergraduate computing student [23]. A content analysis is typically conducted to make replicable and valid inferences from texts and examining data [26, p. 18]. Therefore, in the context of this research, the content analysis was used to examine various universities curricula documents and Learner and Lecturer Guides, in an effort to understand the state of secure programming in the undergraduate level.

Fig. 1 represents the research process followed by this study which led to the step-by-step approach for incorporating secure programming into undergraduate computing curricula.

Preprint

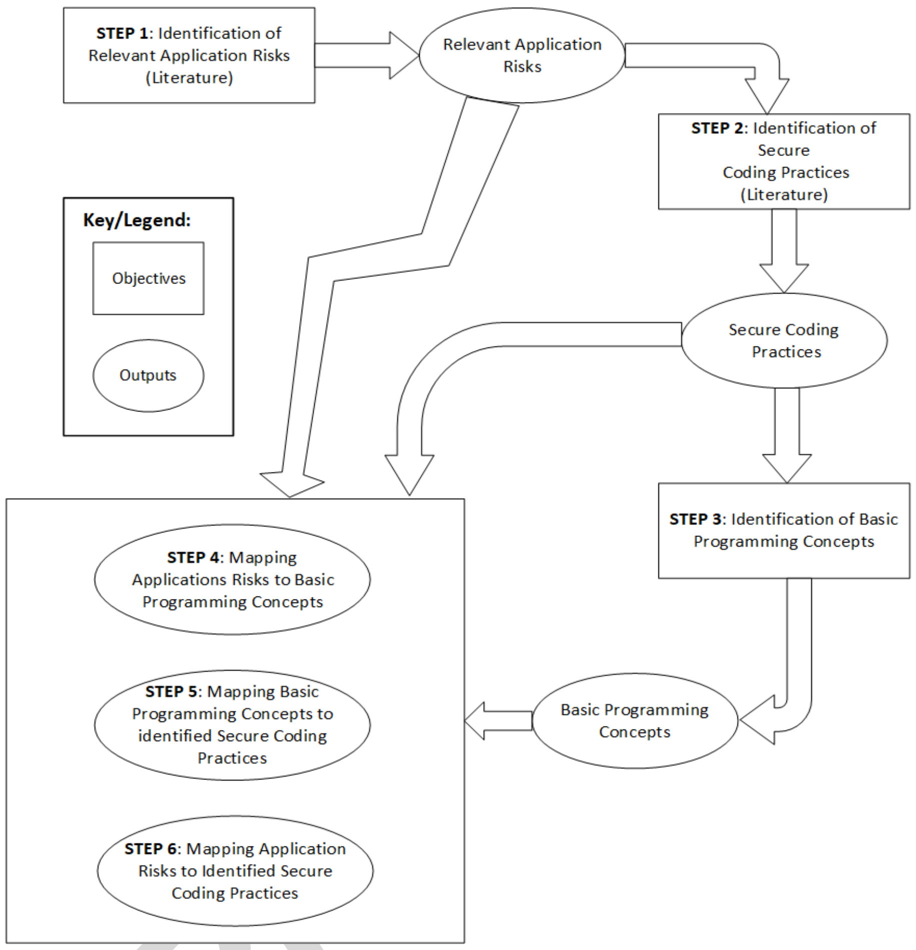

STEP 1: Identification of Relevant Application Risks (Literature)
Relevant Application Risks
STEP 2: Identification of Secure Coding Practices (Literature)
Key/Legend:
Objectives
Outputs
Secure Coding Practices
STEP 3: Identification of Basic Programming Concepts
STEP 4: Mapping Applications Risks to Basic Programming Concepts
STEP 5: Mapping Basic Programming Concepts to identified Secure Coding Practices
Basic Programming Concepts
STEP 6: Mapping Application Risks to Identified Secure Coding Practices

Fig. 1. Research Process.

This research proposes a step-by-step approach for incorporating secure coding practices into programming modules:

- **STEP 1:** Identification of Relevant Application Risks which involves the identification of relevant application risks in the .Net environment and are important for teaching secure programming.
- **STEP 2:** Identification of Secure Coding Practices requires the identification of the secure coding practices that should be taught to computing students as requisite knowledge for secure programming.
- **STEP 3:** Identification of Basic Programming Concepts determines the basic programming concepts typically taught to undergraduate students in the .Net environment.
- **STEP 4:** Mapping Application Risks to Programming Concepts in order to demonstrate the need for teaching application risks along with programming concepts.

- **STEP 5:** Mapping Basic Programming Concepts to Identified Secure Coding Practices in order to highlight the need for, and relevance of integrating secure coding practices to programming concepts taught.
- **STEP 6:** Mapping Application Risks to Identified Secure Coding Practices in order to show the relationship between application risks and secure coding practices to highlight the importance of secure programming.

Each step is described in detail in the following sub-sections.

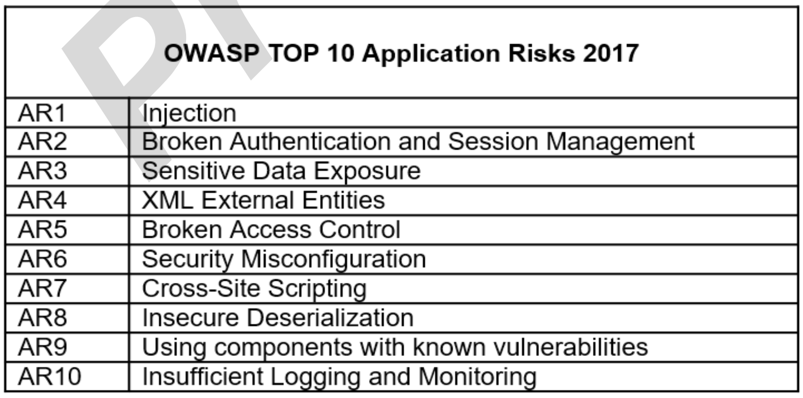
4.1 STEP 1: Identification of Relevant Application Risks (ARs)

An initial literature review was conducted to identify application risks that can affect software applications developed in the .Net environment. The Open Web Application Security Project (OWASP) was used as a source of software application security guidance. OWASP is an international not-for-profit group that is dedicated to helping organisations develop, purchase, and maintain software applications [27]. OWASP is known for providing free documentation for application risks and provides a Top 10 Application Risks document for awareness in web application security [8]. This document represents a broad consensus about the most critical security risks to web applications [21]. Although the OWASP list of Top 10 Application Risks is mostly relevant to web applications, it can also be used for other software applications during application development, testing and maintenance. The examples provided in this paper relate to web applications as they are often deemed the most vulnerable software applications.

Table 1 shows the OWASP Top 10 Application Risks ordered according to their severity, with an encoding identifier being **AR** for Application Risk, followed by its position number in the list.

Preprint

Table 1. OWASP Top 10 Application Risks 2017.



OWASP TOP 10 Application Risks 2017	
AR1	Injection
AR2	Broken Authentication and Session Management
AR3	Sensitive Data Exposure
AR4	XML External Entities
AR5	Broken Access Control
AR6	Security Misconfiguration
AR7	Cross-Site Scripting
AR8	Insecure Deserialization
AR9	Using components with known vulnerabilities
AR10	Insufficient Logging and Monitoring

OWASP's list of Top 10 Application Risks can be used in the development of other software solutions that are not .Net based, to guide and test for well-known vulnerabilities, as these application risks can affect most applications regardless of the development environment. In the identification of these application risks, the SANS Top 25

Programming Errors list [28] was used to compare the current application risks to well-known programming errors, to determine the extent to which the errors could cause the risks listed in OWASP's list of Top 10 Application Risks. Some errors in the SANS Top 25 Errors list are no longer critical, as there have been changes in the security of development platforms and frameworks. This also resulted in the change in OWASP's list of the Top 10 Application Risks, causing cross-site scripting dropping from number 2 in the 2013 list to number 7 in 2017 [21, 28].

4.2 STEP 2: Identification of Secure Coding Practices (SPs)

To identify the secure coding practices, a literature review was conducted where principles, techniques, and practices of secure programming from existing best practices were reviewed. The literature review of fundamental secure coding practices was conducted to understand what software developers need to be competent in with regards to secure programming [16].

The Secure Coding Practices Checklist recommended by OWASP was used in the identification of secure coding practices. The OWASP Secure Coding Practices Checklist can be used to mitigate most common software application vulnerabilities [29]. This checklist addresses the application risks listed in Table 1 and is used later in this paper to map with application risks and basic programming concepts. Table 2 shows OWASP's Secure Coding Practices Checklist, with an encoding identifier being **SP**, followed by its position number in the list.

Preprint

Table 2. OWASP Secure Coding Practices Checklist.

OWASP Secure Coding Practices Checklist	
SP1	Input Validation
SP2	Output Encoding
SP3	Authentication and Password Management
SP4	Session Management
SP5	Access Control
SP6	Cryptographic Practices
SP7	Error Handling and Logging
SP8	Communication Security
SP9	System Configuration
SP10	Database Security
SP11	File Management
SP12	Memory Management
SP13	General Coding Practices

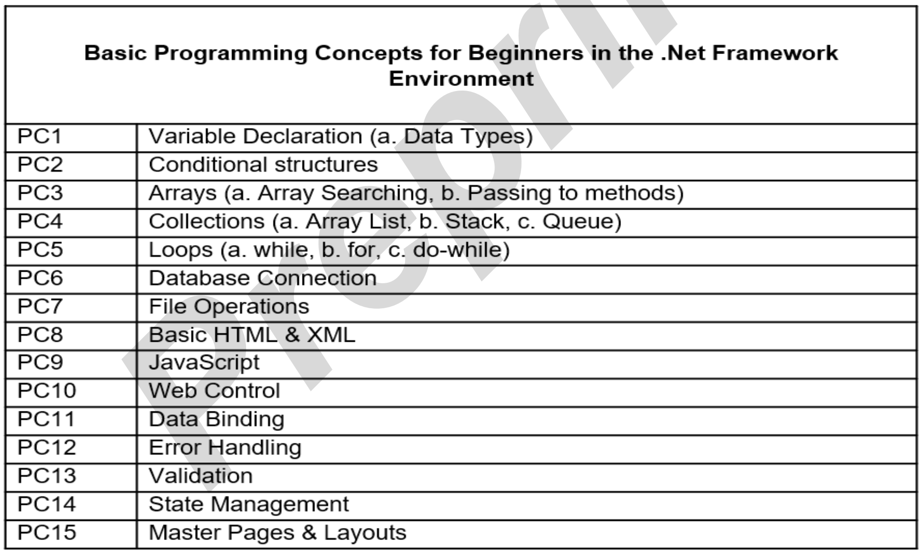
In the ongoing investigation of secure coding practices to be taught to undergraduate students, the concept map by the University of California Davis Secure Programming Clinic [30], was reviewed for the identification and classification of programming practices. The identification and classification of the secure coding practices was verified by the OWASP Secure Coding Practices Checklist [29]. The verification was done to test for the validity of the guidelines and principles in the current secure programming clinic.

4.3 STEP 3: Identification of Basic Programming Concepts (PCs)

Having identified the secure coding practices, the globally published curricula guidelines for undergraduate computing programs were reviewed, to determine the extent to which secure programming should be addressed in Computer Science (CS) and Information Technology (IT) qualifications. The focus was on the ACM curricula documents, as the ACM tailors curricula recommendations to the rapidly changing landscape of computer technology. Although the ACM curricula guidelines mention security as being part of computing curricula, the guideline documents for CS and IT do not have adequate guidance on how secure programming can be taught to enable a graduate software developer to be competent in secure programming. The key to educating and training software developers is typically in the Prospectus and Learner and Lecturer Guides pertaining to each university [10, 14].

Preprint

Table 3. Basic Programming Concepts for Beginners in the .Net Environment.



Basic Programming Concepts for Beginners in the .Net Framework Environment	
PC1	Variable Declaration (a. Data Types)
PC2	Conditional structures
PC3	Arrays (a. Array Searching, b. Passing to methods)
PC4	Collections (a. Array List, b. Stack, c. Queue)
PC5	Loops (a. while, b. for, c. do-while)
PC6	Database Connection
PC7	File Operations
PC8	Basic HTML & XML
PC9	JavaScript
PC10	Web Control
PC11	Data Binding
PC12	Error Handling
PC13	Validation
PC14	State Management
PC15	Master Pages & Layouts

To understand the state of programming in the undergraduate level, a thematic content analysis was conducted on undergraduate computing curricula in South Africa. The content analysis was conducted to determine basic programming concepts taught in the .Net environment, across different public universities in South Africa. The Prospectus and Learner and Lecturer Guides that are available on the universities websites were used in understanding the state of programming in the undergraduate level.

Table 3 provides a list of basic programming concepts that are typically taught across South African universities. The list does not provide the order in which the concepts are taught, but it rather outlines the fundamentals of programming that are for beginners developing in the .Net environment. Each item in the list is given an encoding identifier **PC** for programming concept, followed by its position number in the table.

4.4 STEP 4: Mapping Application Risks (ARs) to Basic Programming Concepts (PCs)

After consolidating findings about what programming concepts should be included when teaching secure programming in South African universities, the researcher created mapping links of how application risks can be taught when teaching programming concepts. Mapping links of how application risks must be taught along with programming concepts were created by the researcher. The purpose of the mapping links is to demonstrate the need for, and relevance of teaching application risks along with programming concepts. The mapping links in the content analysis results were given an impact value (**I**). **I** is based on how many times a Programming Concept (**PC**) was linked to an Application Risk (**AR**) horizontally according to the Programming Concept (**PC**), and a measure of how many times an Application Risk (**AR**) was linked to a Programming Concept (**PC**) vertically according to the Application Risk (**AR**). **I** can also be seen as a way of prioritising important links. Table 4 shows the mapping links between the identified Programming Concepts and the OWASP Top 10 Application Risks.

Table 4. Mapping of Basic Programming Concepts to OWASP Top 10 Application Risks.

Programming Concept	OWASP Top 10 Application Risks 2017										*(I)*
	AR1	AR2	AR3	AR4	AR5	AR6	AR7	AR8	AR9	AR10	
PC1: Variable Declaration		x									*1*
PC2: Conditional Structures *	x	x			x			x			*4*
PC3: Arrays	x										*1*
PC4: Collections	x		x								*2*
PC5: Loops	x										*1*
PC6: Database Connection *	x	x	x	x	x						*5*
PC7: File Operations	x			x		x					*3*
PC8: Basic HTML & XML *	x			x		x	x		x		*5*
PC9: JavaScript		x					x		x		*3*
PC10: Web Controls *	x					x	x		x		*4*
PC11: Data Binding	x										*1*
PC12: Error Handling *	x	x	x		x	x				x	*6*
PC13: Validation *	x	x	x		x		x	x			*6*
PC14: State Management		x									*1*
PC15: Master Pages & Layouts		x			x						*2*
Impact *(I)*	*11*	*8*	*4*	*3*	*5*	*4*	*4*	*2*	*3*	*1*	*(I)*

The mapping of programming concepts to application risks shows the relationship that the programming concept has directly to the application risk. A programming concept can have a number of application risks associated with it, and an application risk can occur due to poorly written programming concepts. A programming concept that links with many application risks receives a high impact value (**I**), where an impact value of

Preprint

4 and above would mean that the link needs special attention. Therefore, lecturers of programming courses could then prioritise the time taken for each link according to the impact value, allowing them to spend more time teaching links with a high impact value. For the purpose of this paper, the programming concepts with a high impact value will be used to demonstrate the importance of considering application risks when teaching programming.

The programming concept Error Handling (**PC12**) links to many application risks and thus, it received a high impact value (**I)** of 6. Error handling is the last defense in a software application when written code statements do not execute as expected [5, 29]. To educate students on how to program securely, the associated application risks must be taught to students after the introduction of the programming concept. The introduction of these application risks should begin from the first year of study, and be taught in parallel with programming concepts as they occur in the syllabus. In an attempt to ensure that students adhere to secure programming and avoid these application risks, students must implement a means that recovers from errors e.g. Try-catch [7].

Similarly, the programming concept Validation (**PC13**) received a high impact value (**I**) of 6 which shows its importance to software applications. Applications without proper validation of data can be vulnerable to various applications risks [21]. Students must be encouraged to always use input validation to avoid the application risks such as Injection (**AR1)** associated with Validation (**PC13)** in **Table 4** [13]. Encouraging students to do Validation (**PC13**) would require lecturers to examine the students' adherence through setting laboratory practicals that require input validation. Students would be assessed and their work graded by reviewing the code they develop.

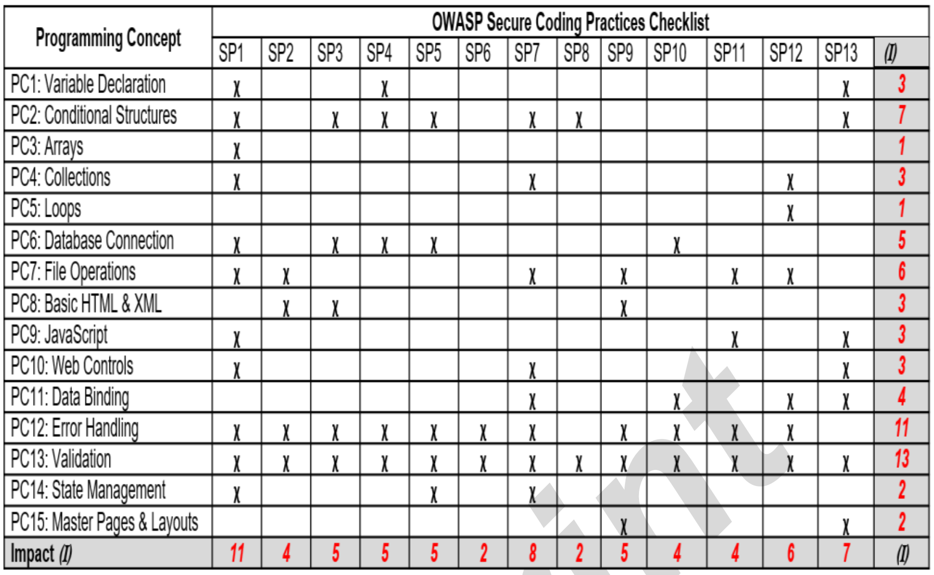
In Table 4, Injection (**AR1)** is the first in the list of OWASP Top 10 Application Risks, which shows how critical this risk is to software applications [21]. This application risk should be introduced and taught in parallel with the associated programming concepts to avoid this risk from occurring. To avoid Injections (**AR1),** students must be taught how they relate to each of the associated programming concepts (i.e. PC2, PC6, PC8, PC10, PC12 and PC13).

Preprint

4.5 STEP 5: Mapping Basic Programming Concepts (PCs) to Secure Coding Practices (SPs)

After understanding the application risks that must be taught to undergraduate computing students, the researcher created mapping links of how secure coding practices can be taught when teaching basic programming concepts. The purpose of the mapping links is to demonstrate the need for, and relevance of integrating secure coding practices to basic programming concepts taught to computing students. The mapping links identified in the content analysis results were given an impact value (**I**). **I** is based on how many times a Programming Concept (**PC**) was linked to a Secure Coding Practice (**SP**) horizontally according to the Programming Concept (**PC**), and a measure of how many times a Secure Coding Practice (**SP**) was linked to a Programming Concept (**PC**) vertically according to the Secure Coding Practice (**SP**). **I** can also be seen as a way of prioritising important links that need special attention. Table 5 shows the mapping links between Programming Concepts and the identified Secure Coding Practices.

Table 5. Mapping of Basic Programming Concepts to Secure Coding Practices.



Programming Concept	OWASP Secure Coding Practices Checklist													
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	*(I)*
PC1: Variable Declaration	X			X									X	3
PC2: Conditional Structures	X		X	X	X		X	X					X	7
PC3: Arrays	X													1
PC4: Collections	X						X					X		3
PC5: Loops												X		1
PC6: Database Connection	X		X	X	X					X				5
PC7: File Operations	X	X					X		X		X	X		6
PC8: Basic HTML & XML		X	X						X					3
PC9: JavaScript	X										X		X	3
PC10: Web Controls	X						X						X	3
PC11: Data Binding							X			X		X	X	4
PC12: Error Handling	X	X	X	X	X	X	X		X	X	X	X		11
PC13: Validation	X	X	X	X	X	X	X	X	X	X	X	X	X	13
PC14: State Management	X				X		X							2
PC15: Master Pages & Layouts									X				X	2
Impact *(I)*	11	4	5	5	5	2	8	2	5	4	4	6	7	*(I)*

The mapping link between programming concepts and secure coding practices shows a direct relationship between basic programming concepts, and secure coding practices. A programming concept can have a number of secure coding practices that can be associated with it, and a secure coding practice can be applied to a number of programming concepts. A programming concept that links with many secure coding practices receives a high impact value (I), where an impact value of 4 and above would mean that the link needs special attention.

Table 5 shows that the programming concepts Error Handling (**PC12)** and Validation (**PC13)** in this mapping prove to be the most important, as they received the highest impact values (**I)** of 11 and 13 respectively. Similarly, in Table 4 **PC12** and **PC13** achieved high impact values of 6. Most application failures are as a result of lack of Error Handling (**PC12**) and poor Validation (**PC13**). For Input Validation (**SP1)** to work effectively, it is mostly used with Conditional Structures (**PC2)** to avoid errors that might occur due to a lack of Error Handling (**PC12**) and Validation **(PC13)**. When Input Validation (**SP1)** is not properly implemented, an application can be vulnerable to many application risks as shown in Table 4. When educators teach these programming concepts, they should therefore pay specific attention to the impact caused by the association, and try to keep a balance between the programming concept and its associated secure coding practices.

Preprint

4.6 STEP 6: Mapping Application Risks (ARs) to Identified Secure Coding Practices (SPs)

After understanding the secure coding practices that must be taught to undergraduate computing students, the researcher created mapping links that show the relationship between application risks and secure coding practices. The purpose of the relationship links on the mapping is to demonstrate the need for, and relevance of incorporating application risks and secure coding practices in teaching secure programming to computing students. The mapping links in the content analysis results were given an impact value (**I**). **I** is based on how many times an Application Risk (**AR**) was linked to a Secure Coding Practice (**SP**) horizontally according to the Application Risk (**AR**), and a measure of how many times a Secure Coding Practice (**SP**) was linked to an Application Risk (**AR**) vertically according to the Secure Coding Practice (**SP**). **I** can also be seen as a way of prioritising important links that need special attention. Table 6 shows the mapping links between the OWASP list of Top 10 Application Risks and the related Secure Coding Practices Checklist.

Table 6. Mapping Identified Application Risks to Secure Coding Practices.

Application Risk	Secure Coding Practices													
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	*(I)*
AR1: Injection	X	X					X			X	X	X	X	6
AR2: Broken Authentication	X		X	X	X	X	X		X				X	8
AR3: Sensitive Data Exposure					X		X		X	X	X			5
AR4: XML External Entities		X	X	X					X					4
AR5: Broken Access Control	X		X	X	X	X	X							6
AR6: Security Misconfiguration		X		X			X	X	X	X	X	X	X	9
AR7: Cross-Site Scripting	X	X					X		X		X			5
AR8: Insecure Deserialization	X	X					X		X					4
AR9: Using components with known vulnerabilities		X											X	2
AR10: Insufficient Logging and Monitoring							X							1
Impact *(I)*	5	6	3	4	4	2	7	1	6	3	4	2	4	*(I)*

Preprint

The mapping links between application risks and secure coding practices shows a direct relationship between application risks and secure coding practices. An application risk can have a number of secure coding practices that address it, and a secure coding practice can be applied to mitigate a number of application risks. An application risk that links with many secure coding practices receives a high impact value (**I**), where (**I**) of 4 and above would mean that the link needs special attention. Therefore, educators must not teach application risks and secure coding practices in isolation, as the secure coding practices in Table 2 are used to prevent or mitigate the application risks in Table 1.Broken Authentication (**AR2)** in the mapping shown in Table 6, links with many secure coding practices, and thus it receives a high impact value (**I**) of 8. Software applications without a properly structured authentication mechanism can be vulnerable to privilege escalation [5, 29]. The application risk Broken Authentication (**AR2)** and secure coding practice Authentication and Password Management (**SP3)**, provide an example that can be used to teach students not to hard-code passwords, nor leave plaintext passwords in the config files, as that can enable attackers to bypass access controls [5]. Error Handling and Logging (**SP7**) has received a high impact value (**I**) of 7, which shows the importance that error handling and logging has to avoiding application risks such as

Sensitive Data Exposure (**AR3)**. When error handling and logging is properly used in an application, default errors that show critical information such as server details are avoided by showing a custom error created by the programmer [5, 27]. To avoid Security Misconfiguration (**AR6**), students must be taught to avoid insecure default configurations, and verbose error messages containing sensitive information. For ASP.Net applications, students can avoid Security Misconfiguration (**AR6**) by being taught to properly configure the ***.config*** file in the solution. A typical example of configuring the ***.config*** file would be to enable ***customErrors***, so that default error messages will not be displayed.

5 Discussion and Conclusion

For graduate software developers to be competent in secure software development, they should be equipped with relevant and necessary secure programming knowledge in the undergraduate level. Literature review shows that secure coding practices and techniques do exist in the current body of knowledge [7, 21, 22]. However, these secure coding practices and techniques are rarely used as fundamental components of computing curricula, but are rather treated as secondary topics which are briefly discussed in programming courses [7].

Universities are responsible for educating undergraduate computing students, where fundamentals of computing are introduced to students and guided through practical classes in the computer laboratories [14]. Although many universities teach programming, often very little attention is given to secure programming, resulting in incompetent undergraduate software developers. The university Prospectus and Learner and Lecturer Guides are key to teaching undergraduate students, as these documents show what the student will know and be able to apply after completion of the course.

Computing curricula reports such as the various ACM curricula guidelines recommend the teaching of secure programming in undergraduate computing courses. However, these guidelines do not provide adequate guidance on how secure programming can be integrated into the curriculum to enable a graduate software developer to be competent in secure programming.

The step-by-step approach proposed by this paper can be used in various levels of preparing a computing curriculum. The approach can be used in setting up the Prospectus and Learner and Lecturer Guides, to ensure that relevant application risks and secure coding practices are considered in secure programming education, and the actual teaching of the secure coding practices and application risks to students. The steps proposed by this paper go hand-in-hand and cannot be addressed in isolation, as isolating these steps may lead to vulnerabilities that can affect the application.

In addition, the mappings presented in this paper show the relationship between the programming concepts taught to undergraduate students, to the identified application risks and secure coding practices. The mappings serve as a guide for how the application risks can be addressed by considering secure coding practices relating to basic programming concepts. Secure coding practices must be explicitly incorporated in the undergraduate computing curricula, to ensure that students will be competent in secure software development.

Preprint

This paper proposes that secure coding practices be integrated throughout the undergraduate computing curriculum, from the first year of study, throughout to the final year of study. This approach would not only impact the competence of graduate software developers, but it would positively influence the security of software applications developed by these graduate software developers.

The main limitation of this paper is that the approach and mappings suggested in this paper have not yet been formally validated. This will form part of future research as well as the actual implementation of this approach at various universities across South Africa.

Acknowledgements

The financial assistance of the National Research Foundation (NRF), Nelson Mandela University Research Capacity Development (RCD) and BankSETA towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors, and are not necessarily to be attributed to the funders.

Preprint

References

1. Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., Del Cuvillo, J.: Using innovative instructions to create trustworthy software solutions. Proc. 2nd Int. Work. Hardw. Archit. Support Secur. Priv. - HASP ’13, pp. 1–1. ACM, New York, NY, USA (2013).
2. Uskov, A. V.: Hands-on teaching of software and web applications security. Proc. 3rd Interdiscip. Eng. Des. Educ. Conf. IEDEC 2013, pp. 71–78. IEEE, Santa Clara, CA, USA (2013).
3. Mumtaz, H., Alshayeb, M., Mahmood, S., Niazi, M.: An empirical study to improve software security through the application of code refactoring. Inf. Softw. Technol., pp. 112–125 (2018).
4. Metalidou, E., Marinagi, C., Trivellas, P., Eberhagen, N., Skourlas, C., Giannakopoulos, G.: The Human Factor of Information Security: Unintentional Damage Perspective. Procedia - Soc. Behav. Sci., pp. 424–428 (2014).
5. Veracode: State of Software Security 2017. https://www.veracode.com/state-software-secutity-2017, last accessed 2019/02/20.
6. Rajlich, V.: Teaching developer skills in the first software engineering course. Proc. - Int. Conf. Softw. Eng., pp. 1109–1116. IEEE, San Francisco, CA, USA (2013).
7. Whitney, M., Lipford, H. R., Chu, B., Thomas, T.: Embedding Secure Coding Instruction Into the IDE: Complementing Early and Intermediate CS Courses With ESIDE. J. Educ. Comput. Res., 56(3), pp. 415–438 (2018).
8. Burley, D., Bishop, M., Buck, S., Ekstrom, J., Futcher, L., Gibson, D.: Joint Task Force on Cybersecurity Education. 1(1) November (2017).
9. Mabece, T., Futcher, L., Thomson, K.-L.: Towards using pervasive information security education to influence information security behaviour in undergraduate computing graduates. CONF-IRM 2016 Proc. Int., p. 14 (2016).
10. Lunt, B. M., Ekstrom, J. J., Lawson, E.: Curriculum guidelines for undergraduate degree programs in Information Technology. pp. 1–139 (2008).
11. Ramachandran, M.: Software security require ments management as an emerging cloud

computing service. Int. J. Inf. Manage., 36(4), pp. 580–590 (2016).

12. Perrone, L. F., Aburdene, M., Meng, X.: Approaches to undergraduate instruction in computer security. 2005 ASEE Annu. Conf. Expo. Chang. Landsc. Eng. Technol. Educ. a Glob. World, pp. 651–663 (2005).
13. Cotler, J., College, S., Mathews, L., College, S., Hunsinger, S.: Information Systems Applied Research 2015 AITP Education Special Interest Group (EDSIG) Board of Directors. J. Inf. Syst. Appl. Res., 8(1), pp. 1–65 (2015).
14. The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery, Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology (2017).
15. Aytes, K., Conolly, T.: A research model for investigating human behaviour related to computer security. Proc. 9th Am. Conf. Inf. Syst., pp. 1–6. AMCIS, Tampa, FL, USA (2003).
16. Buoncristiani, M., Buoncristiani, P.: How People Learn (2014).
17. Barr, R. B., Tagg, J.: From Teaching to Learning - A New Paradigm For Undergraduate Education. Chang. Mag. High. Learn., 27(6), pp. 12–26 (2012).
18. Van Niekerk, J. F., Von Solms, R.: Information security culture: A management perspective. Comput. Secur., 29(4), pp. 476–486 (2010).
19. Mcgraw, G.: Software security. IEEE Secur. Priv. Mag., 2(2), pp. 80–83 (2004).
20. Aziz, N. A., Shamsuddin, S. N. Z., Hassan, N. A.: Inculcating Secure Coding for beginners. 2016 Int. Conf. Informatics Comput. ICIC 2016, Icic, pp. 164–168. IEEE, Mataram, Indonesia (2017).
21. OWASP: OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks. Owasp, p. 24 (2017).
22. Singhal, A., Winograd, T., Scarfone, K.: Guide to Secure Web Services. NIST Spec. Publ. 800-95, 95(1), pp 1-128 (2007).
23. Bishop, M., Frincke, D. A.: Teaching secure programming. IEEE Secur. Priv., 3(5), pp. 54–56 (2005).
24. Department of Education: Department of Education Higher Education Act , 1997 : Regulations for the Registration of Department of Education (1997).
25. Department of Education: Creating Comprehensive Universities In South Africa : A Concept Document. Department of Education, pp. 1–40 (2004).
26. Krippendorff, K.: Content Analysis An Introduction to its Methodology, 31(6) (1985).
27. OWASP: OWASP Secure Coding Practices Checklist, https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_Checklist, last accessed 2019/05/24.
28. Christey, S. and Martin, B.: CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors. SANS Institute. p. 41 (2011).
29. OWASP: OWASP Secure Coding Practices Quick Reference Guide, pp. 1–17 (2010).
30. Bishop, M. et al.: Secure Programming Clinic: Concept Map, http://spc.cs.ucdavis.edu/index.php/conceptmap, last accessed 2019/02/20.
31. An, Z. and Liu, H.: Realization of buffer overflow. Proc. - 2010 Int. Forum Inf. Technol. Appl. IFITA 2010, 1(1) pp. 347–349 (2010)

Preprint