Identifying Potential Security Flaws using Loophole Analysis and the SECREt

Curtis Busby-Earle
Department of Computing
The University of the West Indies
Mona, Jamaica
Email: curtis.busbyearle@uwimona.edu.jm

Ezra K. Mugisa
Department of Computing
the University of the West Indies
Mona, Jamaica
Email: ezra.mugisa@uwimona.edu.jm

***Abstract*—In contemporary software development there are a number of methods that attempt to ensure the security of a system. Many of these methods are however introduced in the latter stages of development or try to address the issues of securing a software system by envisioning possible threats to that system, knowledge that is usually both subjective and esoteric.**

In this paper we introduce the concept of path fixation and discuss how contradictory paths or loopholes, discovered during requirements engineering and using only a requirements specification document, can lead to potential security flaws in a proposed system.

The SECREt is a proof-of-concept prototype tool developed to demonstrate the effectiveness of loophole analysis. We discuss how the tool performs a loophole analysis and present the results of tests conducted on an actual specification document. We conclude that loophole analysis is an effective, objective method for the discovery of potential vulnerabilitites that exist in proposed systems and that the SECREt can be successfully incorporated into the requirements engineering process.

***Index Terms*—Security, Loopholes, Requirements Engineering, SECREt.**

I. Introduction

Cross-cutting concerns are those parts of a system whose functionality spans multiple system modules. Security is typically represented as one such cross-cutting concern. The security of systems and the information these systems are designed and built to manage is primary among the issues considered when developing these systems [1]–[3]. To address the issues related to the security of systems, many methods and approaches have been incorporated into software development processes. These include UMLSec [4], abuse cases [5], misuse cases [6] and the SQUARE method [7]. Many approaches and methods are however threat specific, reactive and subjective [8].

Anti-malware applications are dependent on the detection of what is known i.e. specific threats. In essence this is accomplished by the identification of a piece of malware's signature, actions, heuristics or a combination of these [9]. Intrusion detection systems, stateful inspection and packet analyzing firewalls investigate incoming and outgoing packets (e.g. TCP and IP) for violations against rules contained within their configuration files. Again these rules are based on specific threats. Examples include various forms of hijacking and port scanning, SYN flooding and amplification attacks. When infiltrated, knowledge specific to the attack is built to prevent

DOI: 10.5176_2010-2283_1.2.40

it in the future.

On the other hand, approaches that are used to incorporate security during the early stages of software development attempt to determine potential threats. The tasks associated with the identification of potential security threats are to select the assets to be protected, identify vulnerabilities in the context of potential threats and specify countermeasures [10]. The associated methods are however, highly subjective. For example, the guidelines for the incorporation of misuse cases during the engineering process involves the introduction of "threats that are reasonably likely" to a set of use cases. UMLsec models attackers by representing possible capabilities (e.g. the default attacker described as one with "modest capability"). The SQUARE method suggests the use of misuse cases, abuse cases and threat scenarios among others to elicit security requirements.

Interestingly, the source of many an attack can be traced to stages of development as early as the requirements engineering phase. The Common Weakness Enumeration listed the top twenty-five sources of software vulnerabilities [11]. The number one source was the improper validation of inputs, a function that can be designed into a system as early as the requirements engineering phase of development. Attacks such as cross-site scripting, SQL and command injection take advantage of improper input validation. To extend coetaneous approaches and methods to securing software systems we determined that a more proactive, objective approach is needed.

The remainder of the paper is as follows. In section two we introduce and discuss the concepts of path fixation and loopholes, in section three we describe the methods involved in the approach, section four presents the Secure Requirements Writer (SECREt) and demonstrates the effectiveness of the loophole analysis. We conclude and discuss our future work in section five.

II. Path Fixation and Loopholes

Functional fixation is the inability to see uses for something beyond the use commonly presented for it [12], in other words, it is the belief that something can only be used for its default or intended purpose. We transpose the notion of functional fixation to the analysis of requirements, by considering possibilities for using a system being designed in ways other than those that are intended. To investigate such possibilities we analyse paths.

We define paths to be the sequences of desired system or application functions and their pre- and post conditions, as they are defined in a software requirements specification (SRS) document. Analogously then, we define path fixation as the belief that the paths described in an SRS are the only ones that will exist in the proposed system. The discovery of paths that contradict such beliefs is the basis of our loophole analysis.

Loopholes are potential paths that will result in a user or system capability that contradicts a user or system constraint, as specified in the requirements document. We will demonstrate, using the SECREt, that loopholes can lead to potential security flaws in a system or application.

III. The Methods of the Approach

The fundamental aspect of our approach is that it is based on a security policy. A security policy is the specification of the allowed interactions of a system's users and objects [8]. For the purpose of the approach we must also select an appropriate definition of a vulnerability, one that is also based on a security policy. Piessens defines a vulnerability as any aspect of a computer system that allows for breaches in its security policy [13]. A loophole (see definition in section II above) is

therefore a vulnerability. We adopt Piessens' definition, as our concept of a loophole coincides with his notion of a vulnerability. The steps involved in our loophole analysis are summarised below:

1) Convert the SRS into a more concise format.
2) Develop a more complete specification document, with respect to security.
3) Discover potential vulnerabilities by analysing the amended specification document for loopholes.
4) Improve the security of the intended system by rectifying any discovered loopholes.

A. Derived Requirements

To convert an SRS into a more concise format we employ the use of derived requirements. A derived requirement (DR) is a policy-based expression: it describes the action a subject performs on an object as either a capability or a constraint [8]. For example, the requirement,

2.1.5 The accountant shall be able to read the expenses file

would be expressed in the derived requirement format as,

$^{2.1.5}[user\ _{accountant} : access\ _{read} : file\ _{expenses}\ ]_{cap}$

where the subject is the 'user$_{accountant}$', the action is 'access$_{read}$' and the object is 'file$_{expenses}$'. In the derived requirement format, accountant, read and expenses are called clarifiers. These clarifiers and the other variables subject, action and object are the elements of a derived requirement expression. By representing an SRS as a set of derived requirements we are able to reduce its size, on average, by over 90%, and still maintain the essence of the description of the intended system functionality.

B. Imposed Security Dependence

To appropriately analyse the specification document for loopholes, based on the description of the functions of a system contained in a specification document, the document itself must be suitably complete. In particular, it must be suitably completed with respect to its security requirements, as these are usually major cross-cutting concerns in the development of a system. The primary issue however, with incorporating aspects related to the security of a system during the requirements engineering process is that these aspects are typically included implicitly [14]. We address this by defining the imposed security dependency.

Elements of derived requirements such as actions and objects may have imposed security dependencies [15]. We define an imposed security dependence as follows:

Boilerplate placement value (BPV) α has an imposed security dependence (ISD) on BPV β when the use of α in a derived requirement dictates the use of β in at least one related derived requirement.

Boilerplates are templates that are used to express requirements in a consistent manner [16]. We incorporate boilerplates in developing derived requirement expressions. The placement values are the items of data used to complete the requirement template and are then extracted into a derived requirement. ISD relationships are neither reflexive nor symmetrical but can be transitive. To reduce possible ambiguity of meaning and due to the implicit nature of security requirements, placement values can be standardized in an ISD table that defines them, their attributes and their imposed security dependencies.

C. The Loophole Algorithm

Having converted a specification document into a more concise form using derived requirements, and developed a more complete set of requirements using the derived format and imposed security dependencies, we can now begin the analysis for loopholes.

Let D be the set of all derived requirements that are obtained from a particular requirements document, and R be a relation that maps an element of D to its successor element(s). A requirement's successors are those requirements (0..n) that describe the functions or processes that shall be available or reachable directly from it. Successors are obtained from the post conditions of each requirement in a specification document. As we wish to represent the possible transitions from function to function within the proposed system, the properties of R are listed below.

$$R : D \leftrightarrow D \tag{1}$$

$$\forall r : D \bullet r \mapsto r \notin R \tag{2}$$

$$\forall r, q : D \bullet r \mapsto q \in R \Rightarrow q \mapsto r \in R \tag{3}$$

$$\forall r, q, s : D \bullet r \mapsto q \in R \wedge q \mapsto s \in R \Rightarrow r \mapsto s \in R \tag{4}$$

Using these properties of R, a hierarchy of derived requirements and their successors is created, and this hierarchy is then used to construct the mappings of R. The hierarchy is depicted as a simple digraph. The requirement references (nodes) and directed edges are used to represent the relationships among requirements and their successors. By utilizing R we want to identify policy breaches. The policy is described by the allowed interactions of the intended system's users and objects i.e. the members of D. To identify breaches, we analyse the potential transitions (paths) by analysing the mappings contained in R. For this purpose R is incomplete as the intended transitions from process to process have been defined in the requirements document, but intrinsically these are not all that are possible. We therefore complete the "description" of the set of all possible paths by finding the transitive closure of R. The steps of the loophole analysis are as follows:

1) Represent the relation R as a binary matrix M.
2) Find the transitive closure of R, using the Floyd-Warshall algorithm. Call this new relation R*.
3) Represent R* as a binary matrix M′.
4) Perform the bit-wise XOR of corresponding elements of M and M′. This will identify maplets created as a result of step 2.
5) Where a 1 exists in M′ but not M, excluding any that exist along its diagonal (as we are not interested in reflexivity by statement 2), create a temporary derived requirement by combining the subject of the head derived requirement of the path (head endpoint) with the action and object of the tail derived requirement (tail endpoint) of the path. In a two dimensional matrix representation of M′ the head endpoint will correspond to a row identifier and the tail endpoint a column identifier.
6) Compare each temporary derived requirement with every derived requirement in D expressed as a constraint.
7) A loophole (i.e. a vulnerability) exists when there is a match.

IV. Loophole Analysis with the SECREt

The SECREt is a proof-of-concept prototype written in Javascript, HTML and SQLite. It was designed to be used by a requirements engineer to augment existing approaches to developing requirements for more secure software systems. It is to be incorporated at the end of the requirements engineering phase: having created a set of

requirements using traditional methods of elicitation and incorporating methods such as misuse cases and attack trees, the engineer would then use the SECREt to further "harden" the intended system by identifying omissions and inconsistencies among requirements, creating a more suitably complete set of requirements with respect to security, and searching for and addressing any loopholes. Its main features include the ability to:

- format derived requirements.
- identify security related deficiencies in requirements (first stage analysis or 1SA). These deficiencies are oversights, omissions, weak authentication (passwords and phrases) and potential disclosures.
- generate derived requirements based on imposed security dependencies using an ISD table (second stage analysis or 2SA). As discussed earlier, we are striving for completeness with respect to the security aspects of the requirements specification.
- identify policy breaches using the algorithm previously discussed (third stage analysis or 3SA).
- generate a skeletal requirements document using a derived requirement set, the Volere requirements shell [17] and boilerplates [16].

Sets of DRs can also be saved to or loaded from a file, printed and sorted. ISD tables are also managed using the SECREt. The 1SA is described in [8]. The 2SA is governed by the function *analyzeISD()*. This function identifies omitted security related requirements based on an ISD table stored in one of the tool's SQLite databases. It incorporates the use of four sub-functions that each loop through the set of derived requirements and check for omissions related to actions, objects, action and object clarifiers. The 2SA successfully completes when no new derived requirements have been added to the set. The function *findLoopHoles()* governs the 3SA by executing the following processes:

a) sorts the set D by object.
b) verifies that all successor requirement entries are valid. A successor is deemed invalid if a corresponding derived requirement does not exist. *findLoopHoles()* cannot continue until all successor requirement entries have been validated.
c) builds the binary, n x n matrix M (where n=$|D|$) using multi-dimensional arrays, and makes a copy of M. This copy is used to complete the transitive closure of M, M′.
d) performs the Floyd-Warshall algorithm to complete the transitive closure of R.

findLoopHoles() then compares the arrays used to represent M and M′. If there is no difference between the two, the function raises an alert that indicates no loopholes were discovered. If there is a difference then the bit-wise XOR is performed on the arrays, temporary derived requirements are built, and the function then loops through an array of derived requirements expressed as constraints. The function then compares the temporary derived requirements with these constraints and displays any matches that are found as potential flaws. These matches correspond to loopholes or vulnerabilitites that exist in the system's design.

Fig. 1 displays a screen capture of the results of a 3SA on an actual specification document [18]. In the results pane we see that the SECREt has identified flaws. It indicates that an ordinary user can execute a process that allows him/her to become the application's administrator and perform such tasks as creating user accounts. However, the specification document prohibits an ordinary user from carrying out such tasks as R.4.2.1.2, (see fig. 1) is expressed as a constraint. This is a violation of policy

and therefore a vulnerability. Our multi-stage loophole analysis, embodied in the SECREt, is summarised in Fig. 2.

V. Conclusion and Future Work

We have demonstrated that the SECREt can identify potential vulnerabilities in systems based on a system's requirements specification document. The identification is based on the discovery of loopholes. These are unknown paths that would exist if the system were to be developed in accordance with the specification document in the form prior to the loophole analysis. Further, the analysis did not include nor require the motives, resources and skills of an attacker or possible threats to the system to be postulated. Its foundation is based on the statement of policy, expressed as a derived requirement. We conclude that the loophole analysis is an objective approach to the identification of potential vulnerabilities in systems.

We are pursuing a number of enhancements to the SECREt. It is to be re-written in a stricter language and a less 'volatile' environment since keeping abreast with changes in browser functionality has been an unnecessary development requirement for the tool. Once rewritten we intend to improve the interface and features of the tool and make it freely available via the Internet. Feature enhancements include the ability to automatically transform a requirements document from formats such as the Volere requirements shell or the IEEE 830 standard, into that of the derived requirement. The tool will also provide the ability to step through loopholes to visualize the points along the paths that raise concerns. This will provide an engineer the ability to develop appropriate countermeasures with minimal effect on the intended system's features.

Perhaps the most important feature to be included will be the SECREt's security assurance rating of proposed systems. This rating will assert that the requirements engineers of a proposed system have satisfactorily considered and addressed particular classes of vulnerabilities during the design process. The rating, however, will only be applicable if the system is developed in strict accordance with the specification, after the SECREt's analyses and modifications have been performed and made to the specification.

The SECREt, although tested using small to medium sized documents, has not been tested with documents that detail large and very large systems. This is primarily due to the unavailability of such documents for our research due to the reluctance of organizations to disclose such information.

Finally, our research has raised a few questions that are primarily statistical in nature. Do loopholes exist in every specification document? What is the average number of loopholes that can be expected in documents that specify small, medium, large and very large systems? Is the number of loopholes that are discovered proportional to the size of the intended system? How can the process of requirements engineering be improved to reduce the number of loopholes that are introduced? The SECREt will be utilized to investigate the answers.

References

[1] K. Beznosov and B. Chess, "Security for the rest of us: An industry perspective on the secure-software challenge," *IEEE Software*, vol. 25, pp. 10–12, 2008.

[2] L. Liu, E. Yu, and J. Mylopoulos, "Security and privacy requirements analysis within a social setting," in *RE '03: Proceedings of the 11th IEEE international conference on requirements engineering*. Washington, DC, USA: IEEE Computer Society, 2003, p. 151.

[3] R. De Landtsheer and A. van Lamsweerde, "Reasoning about confidentiality at requirements engineering time," in *ESEC/FSE-13: Proceedings of the 10th european software engineering conference held jointly with 13th ACM SIGSOFT international*

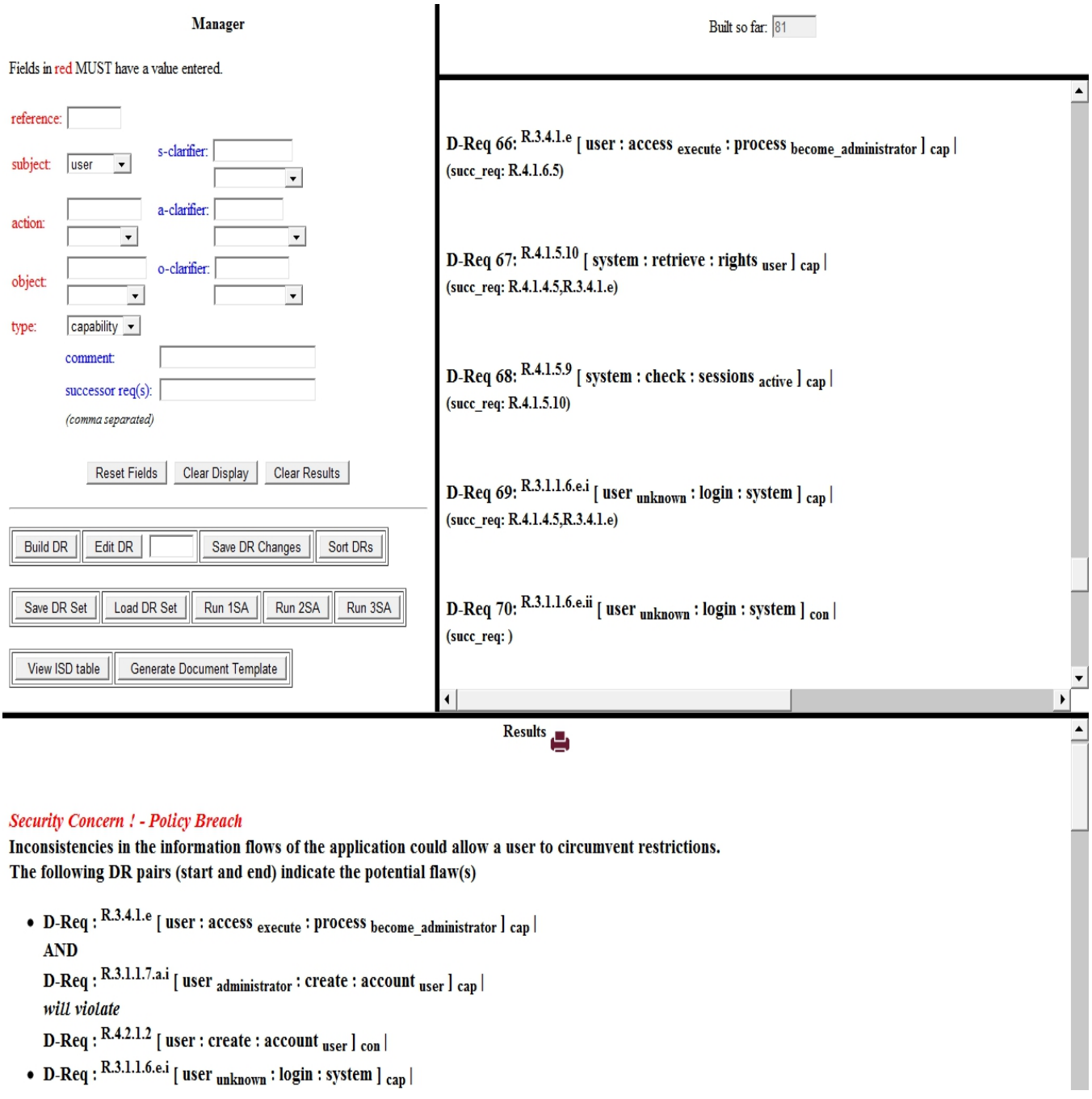


Fig. 1. Results of a 3SA performed by the SECREt

Fig. 2. Multi-stage SRS loophole analysis

symposium on foundations of software engineering. New York, NY, USA: ACM, 2005, pp. 41–49.

[4] J. Jurjens, "Umlsec: extending uml for secure systems development," in *Proc. of the 5th international conference on the unified modeling language*. London, UK: Springer-Verlag, 2002, pp. 412–425.

[5] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings of Computer security applications conference*. IEEE Computer Society, 1999, pp. 55–64.

[6] G. Sindre and A. Opdahl, "Eliciting security requirements by misuse cases," in *Proc. of technology of object oriented languages and systems*. IEEE, 2000, pp. 120–131.

[7] N. Mead, "Identifying security requirements using the security quality requirements engineering (square) method," in *Integrating security and software engineering: advances and future vision*, H. Mouratidis and P. Giorgini, Eds. IGI Global, 2007, ch. 3, pp. 44–69.

[8] C. Busby-Earle and E. K. Mugisa, "Towards writing secure software requirements," in *Proceedings of the IASTED international conference on software engineering (SE 2009)*. CA, USA: ACTA Press, 2009, pp. 101–105.

[9] W. Stallings and L. Brown, *Computer security:principles and practice*, 1st ed. Pearson Prentice Hall, 2008.

[10] B. H. C. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *FOSE '07: Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 285–303.

[11] "Common weakness enumeration," 2008, [Accessed January 2009]. [Online]. Available: http://cwe.mitre.org

[12] P. Zatko, "Psychological security," in *Beautiful security:leading security experts explain how they think*, 1st ed., A. Oram and J. Viega, Eds. O'Reilly Media, 2009, ch. 1, pp. 1–20.

[13] F. Piessens, "A taxonomy (with examples) of cases of software vulnerabilities in internet software," Katholieke Universiteit Leuven, Belgium, Tech. Rep. CW346, 2002.

[14] J. Routh, "Forcing firms to focus: is secure software in your future?" in *Beautiful security:leading security experts explain how they think*, 1st ed., A. Oram and J. Viega, Eds. O'Reilly Media, 2009, ch. 11, pp. 183–197.

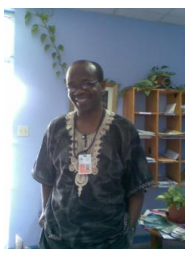
[15] C. Busby-Earle and E. K. Mugisa, "Metadata for boilerplate placement values for secure software development using derived requirements," in *Proceedings of the 13th IASTED international conference on software engineering and applications (SEA 2009)*. USA: ACTA Press, 2009, pp. 196–201.

[16] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*, 2nd ed. Springer, 2005.

[17] "Volere requirements specification template," 2010, [Accessed February 2010]. [Online]. Available: http://www.volere.co.uk/template.htm

[18] "A free 'real world' software requirements specification," 2010, [Accessed January 2010]. [Online]. Available: http://www.devdaily.com/uml/software-requirements-specification-example-use-case

Curtis Busby-Earle is a member of the Faculty of the Department of Computing, University of the West Indies (Mona). He has a BSc General in Computer Science (Major) and Mathemathics (Minor); an MSc in Computer Based Management Information Systems, and is a certified ethical hacker. His primary research interest is in the area of security requirements engineering. He also has interests in web development and computer networks.



Ezra K. Mugisa is a member of the Faculty of the Department of Computing, University of the West Indies (Mona). He received a PhD from Imperial College, London; an MSc from the University of Sheffield and a Dipl. Ing. from the University of Ljubljana - all in Computer Science. His research interests lie primarily in component-based software engineering with a growing interest in IT for development.