THE ROLE OF EXPERTISE ON CODE REVIEW FOR SECURITY: AN EYE TRACKING STUDY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UTKU KAPLAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE IN THE DEPARTMENT OF
CYBER SECURITY

FEBRUARY 2019

Approval of the thesis:

THE ROLE OF EXPERTISE ON CODE REVIEW FOR SECURITY: AN EYE TRACKING STUDY

Submitted by UTKU KAPLAN in partial fulfillment of the requirements for the degree of **Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Assoc. Prof. Dr. Cengiz Acartürk
Head of Department, **Cyber Security** _____

Assoc. Prof. Dr. Cengiz Acartürk
Supervisor, **Cognitive Science Dept., METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Aysu Betin Can
Information Systems Dept., METU _____

Assoc. Prof. Dr. Cengiz Acartürk
Cognitive Science Dept., METU _____

Asst. Prof. Dr. Murat Perit Çakır
Cognitive Science Dept., METU _____

Assoc. Prof. Dr. Banu Günel Kılıç
Information Systems Dept, METU _____

Asst. Prof. Dr. Özkan Kılıç
Dept. of Computer Engineering, YBU _____

Date: **12.02.2019**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: UTKU KAPLAN

Signature : _____

ABSTRACT

THE ROLE OF EXPERTISE ON CODE REVIEW FOR SECURITY: AN EYE TRACKING STUDY

Kaplan, Utku

MSc. Department of Cyber Security

Supervisor: Asst. Prof. Dr. Cengiz Acartürk

February 2019, 82 Pages

To improve the quality of the software and find security vulnerabilities, code review is usually performed during software development activities. The experience of software developers reviewing the code may affect the quality of the code review. This study investigates whether differences between novices and experts in the detection of vulnerabilities in the code can be identified by eye tracking. Participants' eye movements were recorded by an eye tracker while they investigated program codes for security review. The experiment was carried out with 20 programmer participants. The results showed that eye tracking can be used to identify the differences between the code review of novices and experts.

Keywords: software security vulnerabilities, eye tracking, source code review

ÖZ

GÜVENLİ PROGRAMLAMADA UZMANLIĞIN KOD GÖZDEN GEÇİRME ÜZERİNDEKİ ROLÜ: GÖZ TAKİBİ ÇALIŞMASI

KAPLAN, UTKU

Yüksek Lisans, Siber Güvenlik Bölümü

Tez Yöneticisi: Doç. Dr. Cengiz ACARTÜRK

Şubat 2019, 82 Sayfa

Yazılımın kalitesini artırmak ve güvenlik açıklarını bulmak için yazılım geliştirme aktivitileri sırasında genellikle kod gözden geçirme yapılır. Kodu gözden geçiren yazılımcıların tecrübeleri kod gözden geçirmenin kalitesini etkileyebilir. Bu çalışma, koddaki güvenlik açıklarının tespiti konusunda uzmanlar ve acemiler arasındaki farkların göz takibi ile tanımlanıp tanımlanamayacağını araştırmaktadır. Güvenlik açıklıkları gözden geçirmesi için program kodlarının incelenmesi sırasında katılımcıların göz hareketleri göz takip cihazı ile kaydedilmiştir. Deney toplam yirmi programlama katılımcısı ile gerçekleştirilmiştir. Sonuçlar, acemilerin ve uzmanların kod incelemesi arasındaki farkları belirlemek için göz takibinin kullanılabileceğini göstermiştir.

Anahtar Sözcükler: yazılım güvenlik zafiyetleri, göz izleme tekniği, kod gözden geçirme

To my dear family

ACKNOWLEDGMENTS

Firstly, I would like to thank my supervisor Asst. Prof. Dr. Cengiz ACARTÜRK for his guidance and support during my study.

I would like to thank to Sibel Sel for their support during my thesis.

I would like to thank to my mother ŞAZİYE KAPLAN, my father ZAFER KAPLAN and my sister ÇAĞLA KAPLAN for their support during every moment of my life.

I would also like to thank to my friends UMUT DOĞAN and DORUK UZ for their support, review and comments for my thesis.

Finally, I am very grateful to participants of my study for their contribution to science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

METU	Middle East Technical University
px	pixel
ms	milliseconds
AOI	Area of Interest

CHAPTER 1

INTRODUCTION

Various code review techniques are available to detect code errors and security vulnerabilities. These techniques have been used to improve the quality of program codes by detecting the errors in the code. Code review techniques are usually classified into three main categories: formal code review, lightweight code review and pair programming (Kolawa & Huizinga, 2007, p.260). Formal code review includes a detailed review with multiple participants and multiple stages. On the other hand, lightweight code review needs less effort than formal code reviews, as the name suggests (Kolawa & Huizinga, 2007, p.260). Despite the availability of the existing code review techniques, there is no consensus on which code review technique is more effective and successful in detecting security errors in code.

A likely source of security errors in coding is programming skills and experience of coders. In particular, the experience of the coders and their performance may have a major impact on incorrectly designed software programs. Uwano, Nakamura, Monden & Matsumoto (2006) reviewed differences in individual performance and experience by investigating the eye movements of coders as a complementary means of employing the code review techniques. They presented code fragments with logical errors to five participants, and they asked the participants to detect the errors. As a result, they reported specific patterns in eye movement data, in particular, *scanpath* (i.e., the initial examination topology from the beginning to the end). They also found that the participants, who inspected the program code for a longer time during the initial examination, found the

errors in a shorter time. The focus of Uwano, et al. (2006) was logical errors in the program code. In our study, we focus on security vulnerabilities. For this, in the present study, we used predefined program codes with vulnerabilities and we asked the participants to detect them. We report an experimental study that was carried out with 20 participants. We employed eye tracking to record the eye movements of the participants during review of code fragments.

1.1. The Purpose of the Study

Eye tracking has been used in various fields of research to reveal insights into human information processing. These fields involve marketing and education among many others (Horsley, Eliot, Knight, & Reilly, 2014). For example, in marketing, eye-tracking technologies have been used to understand consumer responses to advertising communications, product preferences, billboards, product labeling, TV commercials, and supermarket shelving. The use of eye tracking can also provide valuable information about the learning process of students. Since eye tracking can reveal information about visual attention, it is well suited to examine differences in attention processes of students.

On the other hand, the use of eye tracking in software development has found limited applications to date. The focus of the present study is to investigate whether differences between novices and experts in the detection of vulnerabilities in the code can be identified by eye tracking. The participants' software experience was taken as an independent variable, whereas scan time, defect reporting time, accuracy, fixation count, average fixation duration, total fixation duration and gaze shift were defined as the dependent variables. Some eye movement measurements (dependent variables) may be appropriate for the purpose of this study, while others may not be appropriate. We expect that the result of the current research may contribute methodologically to the field of computer science and engineering education, as well as the companies working on code reviews.

1.2. Research Questions

The research questions were formulated based on participants' potential physiological response (eye movements in this study) and performance as a function of the relationship between their software experience and the efficiency to detect security vulnerabilities in software codes. Both accuracy and response times (to detect security vulnerabilities) were considered. In particular, we address the following research questions in this study:

Q1: Can eye tracking be used to identify differences in a code review of novices and experts?

1.3. The Outline of the Thesis

The rest of the thesis is arranged as follows. In Chapter 2, a literature review was presented on two topics related to the research questions: eye tracking technology and security vulnerabilities in coding. In Chapter 3, the design of the experiment is reported. Results and analyzes are reported in Chapter 4. Finally, Chapter 5 discussed the results and the results were evaluated in terms of research questions. The limitations of this study are also mentioned. In addition, opinions were presented for further studies.

CHAPTER 2

LITERATURE REVIEW

The following sections provide information on eye tracking technology, the use of eye tracking technology in coding and security vulnerabilities in program codes.

2.1. Eye tracking technology

Eye tracking devices are used to measure the gaze location of a participant by collecting eye movement data (Holmqvist & Andersson, 2017). It is usually accepted as an indicator of visual attention. Eye movements have an important role in the detection of stimuli processed by the brain. The stimuli may be any object needed to perform a given task. Eye movement data are analyzed according to regions of stimuli. These regions are called Areas of Interest (AOI). This region may or may not be able to attract the attention of the participant while performing a task. For example, when invetigating a program code, participants focus on the areas interested in, whereas they do not focus on the other sections.

Recent desktop eye tracking technologies have been simplified and integrated into computer monitors or as standalone devices that are no longer physically connected to the participant. Figure 1 shows a participant using desktop eye tracker that is built into the monitor. Eye tracking devices mounted on laptops are used. Figure 2 shows a participant using a mobile eye tracker mounted on a laptop. Desktop eye tracker devices can also be placed next to the screen, which is useful not to distract the participant attention.

The data of eye movement may reveal information about the participant's attentional processes. In addition to its capabilities, most eye-tracking devices allow voice recording, which can provide additional data as a complementary feature. Tobii X2-60 eye-tracking device was used in the present study. That device is a video-based tool to track eye movements with two cameras. The eye-tracker can sample 60 frames per second.

Figure 1. A participant using a desktop eye tracker that is built into the monitor (Bergstrom & Schall, 2014)

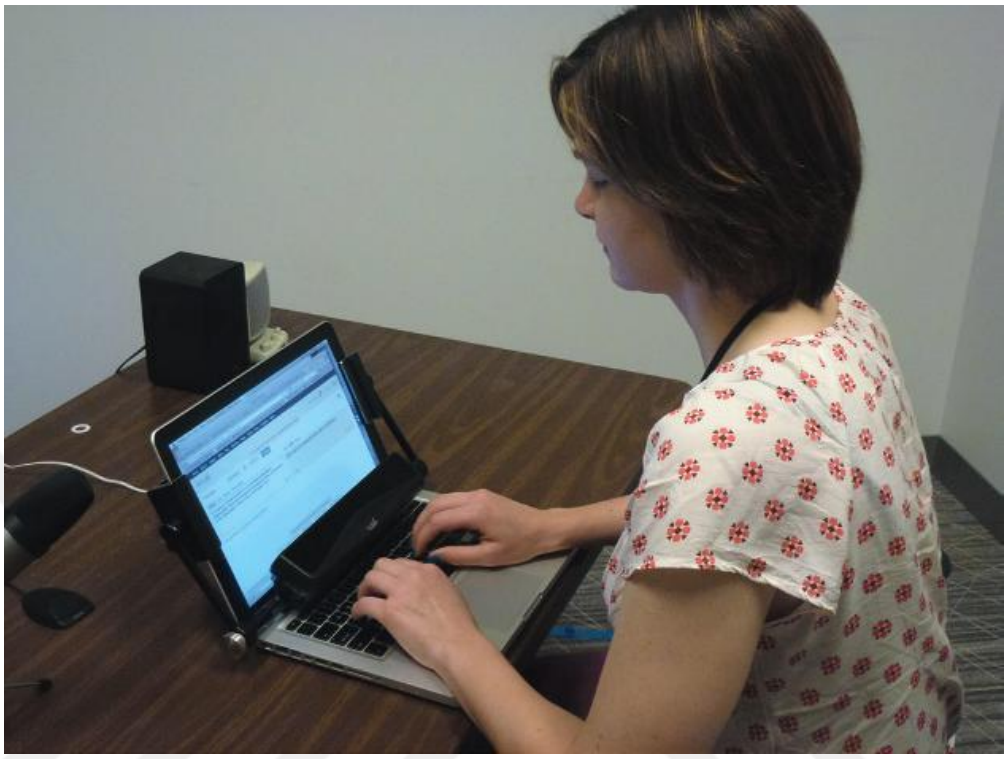

Figure 2. A participant using a mobile eye tracker mounted on a laptop. (Bergstrom & Schall, 2014)

Eye tracking is a method that can provide robust information compared to classical methods, such as recording the person's responses and the time it takes to get that response (Rayner, 1998). For instance, subjects may be asked to find mistakes after understanding and reviewing a program code in a laboratory setting. Eye tracking has several advantages over classical experimental methods, such as verbal protocols. For instance, by using the classical methods, the results are usually obtained after the end of an experimental task. It is difficult to observe how a participant investigates the program code and how and when he/she finds the correct answer. The participants may also forget to tell what they have experienced at the end of a long experiment. Eye tracking may help to overcome some of these difficulties by introducing real-time recording of gaze data.

Eye movement data are basically classified based on oculomotor events such as fixation, saccade, and scanpath (Holmqvist & Andersson, 2017). These eye movement events (oculomotor events) are used as eye movement measures. By using eye movement measures, statistical analysis can be performed, and these measures allow us to understand what our data means in relation to our experimental design. In the following paragraphs, some of those eye movement measures are introduced.

Fixation means looking at a certain location for a certain period of time (Duchowski, 2017, p.44). The following are the three major measures related to fixation (Holmqvist & Andersson, 2017). The first measure is the *fixation position* measure, which holds the information that where a participant looks. Position measure is usually recorded in Cartesian (x, y)-coordinates by eye tracking devices. The second measure is the *fixation duration*, which addresses the question of how long the participant's gaze stays on a specific fixation position. The third is the *number of fixations*. It addresses the question of how many fixations are in the entire test or an area of interest during a test. By using multiple fixation points, it is possible to find out how much visual effort participants spend in a specific experimental setting, such as examining program codes. Fixation measures are important since it has been reported that the acquisition of information takes place during the fixations.

Saccade, another important eye movement event, defines the sudden action from one fixation to another (Gilchrist, 2011, p. 85). It occurs very quickly; at about 40-50 ms. Rayner (1998) reported that the processing of information is minimal during a saccade. Saccade also has measures such as direction, amplitude, velocity, and duration. The saccade direction is the direction of any saccadic movement. The saccade amplitude refers to the length of the path taken from the starting point to the end of a saccadic movement. The saccade duration is defined as the time when taking the path between fixations. It is easy to calculate the start point of a saccade, but the calculation of endpoint may be complicated due to reasons such as blinking. The last measure is the saccade velocity,

which can be found by dividing the path to the duration. Since the velocity levels changes during the movement, the average speed is calculated.

Another eye movement measure is *scanpath*, which is the set of fixations in a sequence (Bylinskii, Borkin, Kim, Pfister, Oliva, 2017). Scanpath describes the task that participants perform. Figure 3 shows two stimuli with four areas of interest. The red and blue lines represent the *scanpath* in the stimuli. *Scanpath* also has measures such as direction, length, and duration. The scanpath direction is a measure of the general direction of fixations and saccades while investigating a stimulus. The scanpath length can be defined as the sum of all saccade length in a scanpath. Scanpath duration defines the time from the start point to the end of the scanpath. The main challenge in this measurement is to calculate the start and end points of the scanpath correctly.

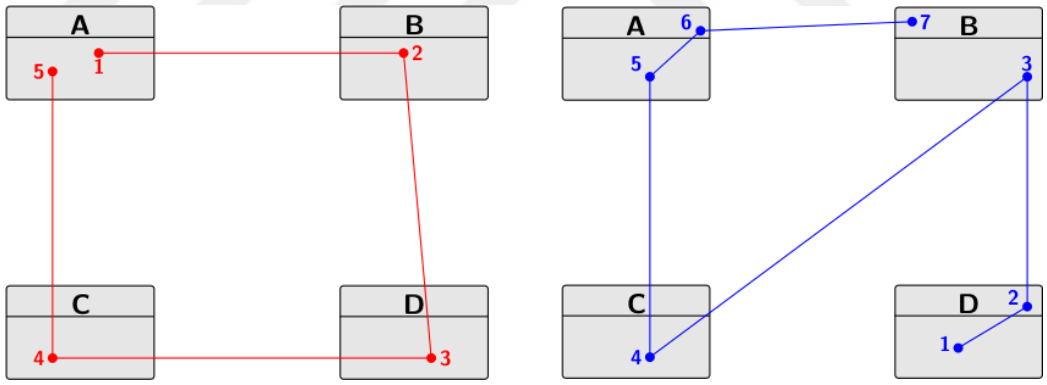

Figure 3. Two different scan-paths (red and blue lines) on the same stimulus by Sharafi et al. (2015)

In the present study, we focus on fixation measures by leaving the saccade and scanpath measures to future work. Another known event is a gaze shift. This event identifies movement from one area of interest to another (Holmqvist & Andersson, 2017). We can define this movement as, for instance, the number of eye movements between text and graphics in a textbook reading. Another major domain of analysis in eye tracking research is a visualization of gaze data. Eye trackers usually come with software packages that can instantly produce *visualizations* (Bergstrom & Schall, 2014, p. 15) although that is not a

must. The output from these software packages help to indicate where the participants focus, the length of gaze and its pattern their eyes follow. Otherwise, third-party software may be used for visualization. Some of the most commonly used visualization techniques are heat map, gaze opacity and gaze plot.

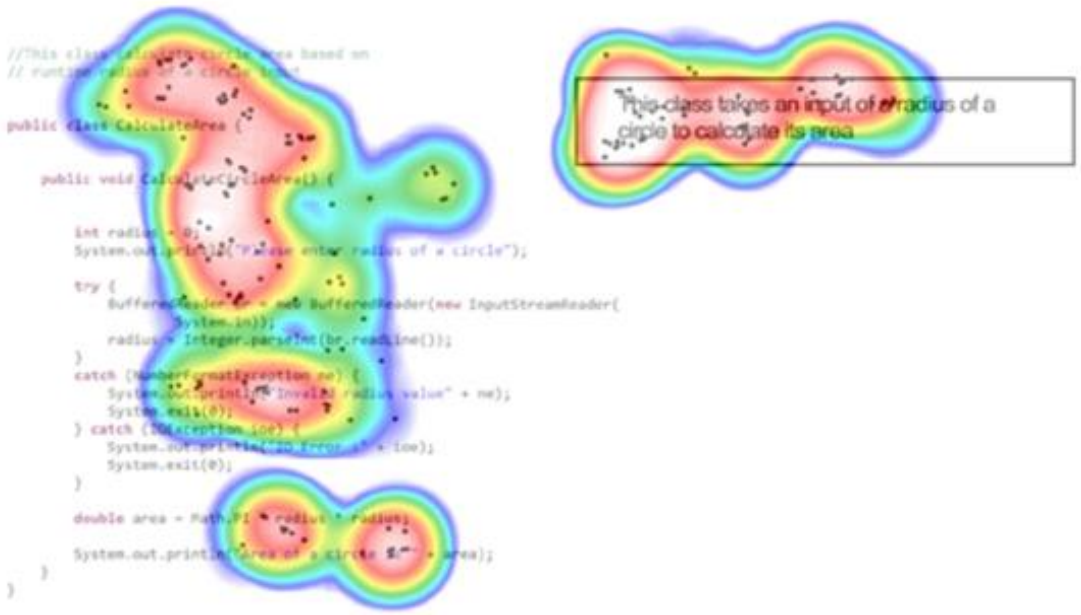

Figure 4. Heat-map by Ali et al. (2012)

The heat map is a collection of different colors representing the density of fixations. The green, blue and red colors describe the intensity of fixations. It shows areas where the participants gazed at by using a heat map on stimuli. A non-colored area in the heat map indicates that the participants did not gaze at the area. Heat map is presented for the task of comprehending the program code in Figure 4. Heat maps show that there are many fixations in comments, method names, and class names according to the density of fixations.

Similar to heat maps, the gaze-opacity technique shows the areas where the participants generally look at or do not look at (Bergstrom & Schall, 2014, p. 61). This technique shows which areas participants overlook and which areas they look at more carefully. The map visualizations are usually easier to understand than numbers, ratios, and graphics. In Figure 5, the maps show that the participants concentrated on the identification number and password fields instead of the instructions on the login screen.

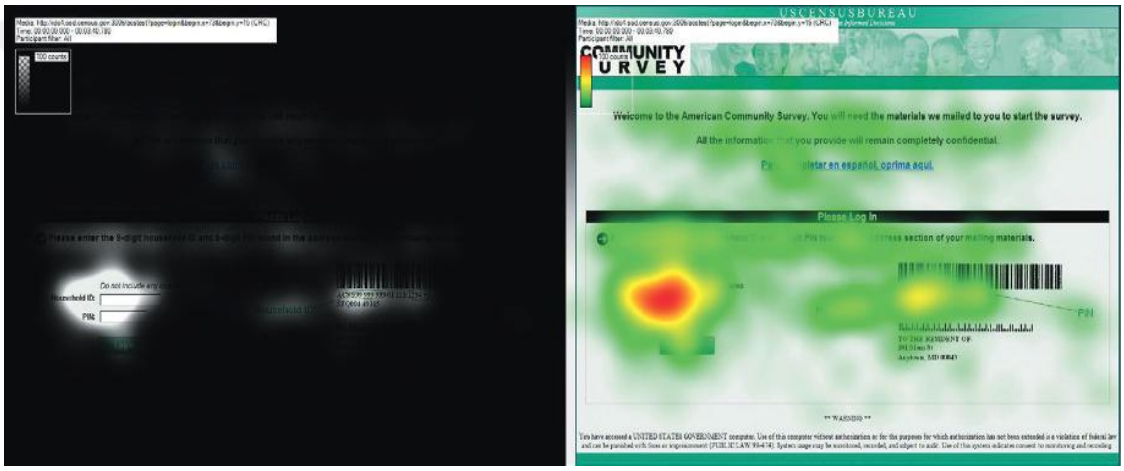

Figure 5. Gaze opacity map (left) and heat map (right) showing that the majority of fixations were on the log-in section and the example Household ID, not on the instruction text by Bergstrom & Andrew (2014)

Another visualization technique is a gaze plot that shows the gaze location of participants (Bergstrom & Schall, 2014, p. 64). A gaze plot visualizes the path that the participants follow (Figure 6). By using the gaze plot technique, the points that the subjects visually inspect can be shown as ordered and concatenated components. In most eye-tracking applications, fixations are represented by circles and saccades are represented by lines that link dots. Fixations are usually numbered to indicate the fixation sequence. The radius of the points shown in the gaze plot varies as a function of how long people look at these points. A point with a larger radius may mean that the participant looks at this point for a longer time.

Figure 6. Example of a gaze plot diagram representing fixations from one individual participant by Bergstrom & Andrew (2014)

2.2. Use of eye-tracking technology in coding assessment

A review of the literature shows that researchers have employed the eye-tracking methodology to analyze how people investigate and comprehend program codes. Using the eye tracking techniques researchers can collect more data than using standard methods (Busjahn et al., 2014, p. 2). For example, by using classical techniques, the participants may be asked to report their experience during the experiment. However, this is challenging because participants may have to interrupt their ongoing work to report what they experience. A similar situation applies to think aloud protocols: the attention of the participants may be disrupted, which may, in turn, result in unintended mistakes during

their ongoing experimental tasks. The participants also usually forget to think aloud during the experiment because they are not accustomed to doing it during daily work. Therefore, the researcher has to track the experiment and warn the participant in cases as such. In this situation, the participants' attention may distract, and as a result, the task may fail unintendedly. Eye tracking helps to overcome some of these challenges that classical experiment methodologies face by reducing introducing less disruptive data recording techniques.

In 2017, Peachock and Sharif investigated the differences between how programmers read normal text and source code. They also observed how the experience of developers has an impact on reading patterns. In the experiment, thirty-three students were asked to perform ten assignments consisting of three normal texts and seven programs. While the type of stimulus is defined as an independent variable, linearity measures such as vertical later text, horizontal later text, vertical next text, regression rate, and line regression rate are defined as dependent variables. As a result, they reported that both experienced and inexperienced software developers read the normal language text more linearly compared to the source code.

Hofmeister, Bauer, Siegmund, and Apel (2017) investigated whether novices use a different strategy from the experts while reading the source code. They wanted to find an answer to whether these differences are due to participants' strategies or the characteristics of the code. To understand the effects caused by the characteristics of the code, linearity, which is a static code measure, is proposed. Linearity was described as a relation between the number of functions and the total jump length. They intended to observe the participants' programming strategies using the eye-tracking device during their programming course. To this end, they prepared code snippets that differ in linearity. After that, differences in the examination of these codes by novices and experts were investigated. The results are pending.

Hofmeister, Bauer, Siegmund (2017) studied the relationship between indentation and code comprehension by using the eye-tracking. In an experiment, they used the Java programming language, which is one of the most used programming languages. They aimed at finding the most appropriate level of indentation for code comprehension. Novices and experts were expected to differ in comprehending the codes. The authors expected that the indentations in the program code would have an impact on saccades and their length. The small indentations would make it difficult to understand the code structure and the large indentations would cause more jumping between lines. The indentation defined as an independent variable was inserted into the code snippets at different levels, and the relationship between the indentation level and comprehension was investigated. The results are pending.

Begel (2015) conducted a study about code review by using Tobii EyeX eye tracker. Begel investigated eye movement measures in order to see if the eye tracking technique would provide useful information about how software engineers perform code reviews. The measures included coverage, reading speed and structural *scanpath*. The results show that the time spent per word, the number of words read per minute (reading speed), and the reading order of the methods (structural scanpath) can be used to determine the quality of feedback from reviewer. Also, it is necessary to understand the whole program while trying to find the bugs in the software. However, the reviewer's programming experience influences the level of comprehension. Thus, the reviewer's knowledge and experience can affect the quality of finding bugs in the software.

Tvarozek, Konopka, Navkat and Bielikova (2015) investigated the students' approach to different programming tasks. They report that comprehending source code requires more effort than comprehending normal text because the structure of the program code is more complicated than the normal text. The purpose of the study was to investigate how students learn programming and how to improve learning process accordingly. The experiment was conducted using a Tobii X2-60 device with undergraduate students in

programming courses. They divided the programming tasks into four sub-types: static code reading, programming tasks, a combination of development with static code reading and code reading for debugging. As a result, they proposed three research questions according to task types to investigate code comprehension strategies for different tasks.

There are also studies where the researchers investigated participants' eye movements in the domain of software programming. For instance, in Crosby and Stelovsky's 1990 study (as cited in Sharafi et al., 2015, p. 20), inexperienced participants investigated comments in the program code for a longer time than experienced participants did. In Bednarik et al. study 2006 (as cited in Sharafi et al., 2015, p. 20), inexperienced participants first looked at the visual graphics instead of reading the code. In Binkley's 2013 study (as cited in Sharafi et al., 2015, p. 20) inexperienced participants paid more attention to the camel case style in the program code than experienced participants did.[1] Besides, experienced participants were not affected by identifier style differences. In Turner's 2014 study (as cited in Sharafi et al., 2015, p. 20) investigated how language selection would affect participants' code reading. Turner's work shows that programming languages influence inexperienced participants more than experienced participants when investigating program codes.

From the perspective of code review methods, various results were reported in the literature. In Bednarik and Tukiainen's 2006 study (as cited in Sharafi et al., 2015, p. 20) code review techniques are valid at the start of the code review process, but is less significant towards the end of the code review process. In Duru et al 2013 study (as cited in Sharafi et al., 2015, p. 20) reported that visualization techniques increased participants' performance and helped participants to develop various strategies during the review of the code. In Sharafi et al. 2012 study (as cited in Sharafi et al., 2015, p. 20) male and female participants used distinct methods to answer the questions.

[1] Camel case style tells how to write compound words or phrases, that is, every word or abbreviation in the middle of the word must begin with an uppercase letter with no spaces.

Moreover, researchers investigated the differences between how participants read the regular text and the program code. Busjahn's 2011 study (as cited in Sharafi et al., 2015, p. 20) demonstrates that participants have longer fixation times and regression rates when reviewing the code than during normal text reading. In Binkley's 2013 study (as cited in Sharafi et al., 2015, p. 20), on the other hand, reading and comprehending the program code is essentially distinct from reading the normal text.

Besides, researchers have investigated the effects of identifier styles. In Binkley's 2013 study (as cited in Sharafi et al., 2015, p. 20) the experience of the participants has an impact on reading the identifiers in the program code. In Sharafi's 2012 study (as cited in Sharafi et al., 2015, p. 20), on the other hand, demonstrates that camel case and underscore identifier styles do not differ in the understanding of the program code when considering accuracy, time and effort.[2] In another study, Sharafi (2012) analyzed the differences between women and men in examining the camel case and underscore identifier styles. He found that there were no differences in considering time, accuracy and effort measures.

In previous studies, source code reviews were conducted using eye tracking to understand how participants investigated the code. Some of these studies focused on how software developers investigate logical or syntactically incorrect codes, while some focus on how software developers examine comments in the code. Instead of investigating logic or syntax errors, as in previous studies, the security vulnerabilities in the code were investigated in the present study. Code reviews were performed with few participants and a small number of stimuli in most previous studies. In the present study, ten different stimuli were used and twenty participants participated in the experiment to obtain results that are more general, as reported in Chapter 3. In addition, in previous studies, the participants were categorized as experts and novices according to their software experience. Similarly, in the present study, it was determined whether the participants were experienced in programming by using questionnaires.

[2] Underscore identifier style is a practice that use to create visual spacing within a sequence of characters, where a whitespace character is not permitted.

2.3. Security vulnerabilities in coding

The software is used with many devices such as mobile devices and household appliances in daily life. The fact that the software has such a wide range of use reveals its importance for human life. Accordingly, the software should be developed in accurate ways and also in a way that does not allow security vulnerabilities. McGraw (2006) proposed that "Software security is the idea of engineering software so that it continues to function correctly under malicious attack." (p. 24) In other words, software security can be defined as designing, building and testing software securely.

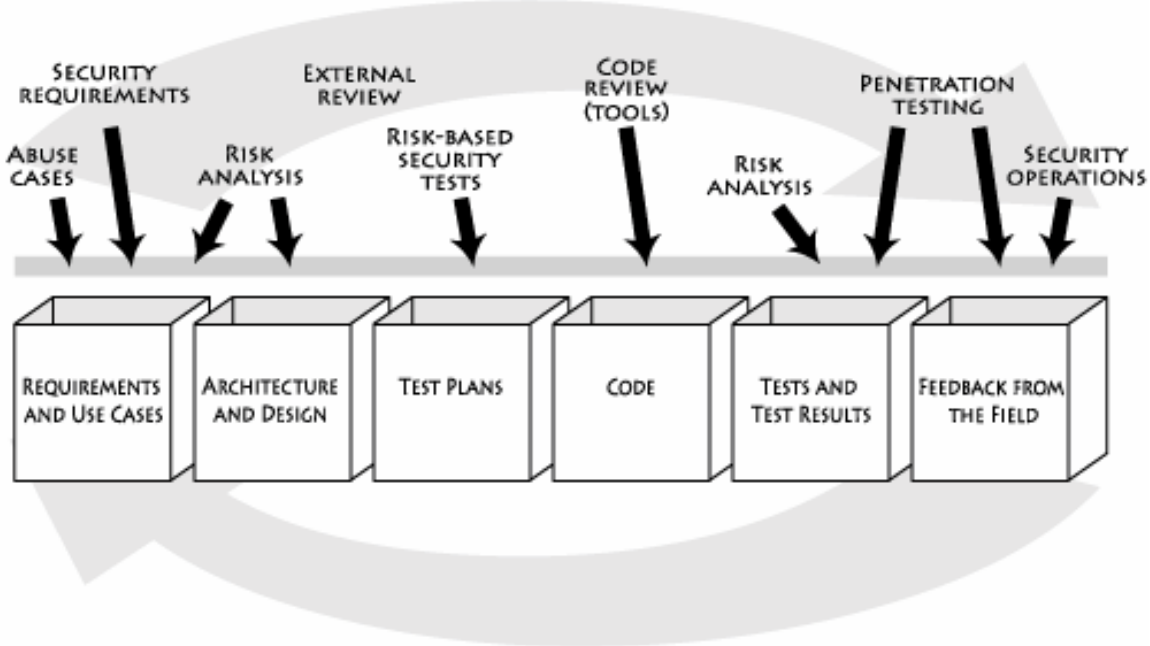

Figure 7. Software security best practices for various software structures by Gary McGraw (2006)

There are three pillars of software security, which are applied to risk management, software security touchpoints, and knowledge (Gary McGraw, 2006, p. 46). Since security risks in software can show up throughout the software lifecycle, risk management must be addressed throughout this process. Also, developers need to apply touchpoints (best

practices) to solve the problems during software development. Touchpoints (best practices) are specified in Figure 7. Moreover, software security knowledge can be applied through the software development lifecycle using touchpoints.

Software security vulnerability is one of the subcategories of software security knowledge. A security vulnerability is a programming error or weakness that attackers can use to compromise the integrity of the code. A weakness in the program code is all an attacker needs to make a security attack. These attacks usually target confidentiality, integrity, or availability of information that a program's creators and users possess. Attackers often use certain tools or methods to find vulnerabilities and attack the program.
The most common security flaws in the code are presented below (CWE and SANS Institute, 2011).

1) SQL Injection

> CWE and SANS Institute (2011) defined that "Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands. SQL injection has become a common issue with database-driven websites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes". For example, the following code executes a SQL query that searches for items according to the currently-authenticated username.

Example Language: C#

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
```

Figure 8. Example of SQL Injection by CWE & SANS Institute (2011)

If an attacker enters the below string for “itemName”:

```
name' OR 'a'='a
```

Figure 9. Example of statement condition by CWE & SANS Institute (2011)

the query result is always true. Actually, the query becomes logically equivalent to the query specified below:

```
SELECT * FROM items;
```

Figure 10. Example of SQL query by CWE & SANS Institute (2011)

The query in figure 10 allows an attacker to bypass the requirement and see the items that all users have.

2) Unlimited File Upload

CWE and SANS Institute (2011) defined that “The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.” In Figure 11, there is no check on the type of file being uploaded. Assuming that images are available at the root of the web document, an attacker could upload a malicious file.

Example Language: **PHP**

```
// Define the target location where the picture being

// uploaded is going to be saved.
$target = "pictures/" . basename($_FILES['uploadedfile']['name']);

// Move the uploaded file to the new location.
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target))
{
  echo "The picture has been successfully uploaded.";
}
else
{
  echo "There was an error uploading the picture, please try again.";
}
```

Figure 11. Example of SQL query by CWE & SANS Institute (2011)

3) Cross-site Scripting

CWE and SANS Institute (2011) defined that "The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users." For instance, the code in Figure 12 displays a welcome message on a web page.

Example Language: **PHP**

```
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

Figure 12. Example of code including cross-site scripting

The attacker can modify the URL of the page and put malicious code in $username parameter.

```
http://trustedSite.example.com/welcome.php?username=<Script Language="Javascript">alert("You've been attacked!");</Script>
```

Figure 13. Example of the script by CWE & SANS Institute (2011)

4) Missing Authentication for an Important Method

CWE and SANS Institute (2011) state that "The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources." In Figure 14, the method is used to create a new bank account. However, there is no authentication to check that the user has the authority to produce new bank accounts.

Example Language: **Java**

```
public BankAccount createBankAccount(String accountNumber, String accountType,
String accountName, String accountSSN, double balance) {
  BankAccount account = new BankAccount();
  account.setAccountNumber(accountNumber);
  account.setAccountType(accountType);
  account.setAccountOwnerName(accountName);
  account.setAccountOwnerSSN(accountSSN);
  account.setBalance(balance);

  return account;
}
```

Figure 14. Example of critical function by CWE & SANS Institute (2011)

5) Hard-coded Credentials

CWE and SANS Institute (2011) defined that "The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its inbound authentication, outbound communication to external components, or encryption of internal data."

Figure 15 shows that a piece of code uses a hard-coded username and password to connect to a database. Anyone who has access to code can use a password to connect to the database.

```
Example Language: Java

...
DriverManager.getConnection(url, "scott", "tiger");
...
```

Figure 15. Example of database connection code with hard-coded credentials by CWE & SANS Institute (2011)

6) Misuse of Hash and Salt

CWE and SANS Institute (2011) defined that "The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use salt as part of the input." In figure 16, if given password matches a stored password, the user can log in. This code does not use salt for hashing function; thus attacker is able to reverse the hash and find the original password.

```
Example Language: Java

String plainText = new String(plainTextIn);
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
  login_user();
}
```

Figure 16. Example of a method that checks a given password by CWE & SANS Institute (2011)

7) Unrestricted Authentication Attempts

CWE and SANS Institute (2011) defined that "The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks." In figure 17, the method performs authentication when the application is invoked. Nevertheless, the software does not attempt to restrict excessive authentication attempts.

Example Language: **Java**

```
String username = request.getParameter("username");
String password = request.getParameter("password");

int authResult = authenticateUser(username, password);
```

Figure 17. Example of authentication code by CWE & SANS Institute (2011)

8) Encryption of Valuable Data

CWE and SANS Institute (2011) defined that "The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys." In figure 18, function stores the user credentials in a cookie on the user's machine. If an attacker compromises a user's machine, the user's information can be exposed.

Example Language: **PHP**

```
function persistLogin($username, $password){
  $data = array("username" => $username, "password"=> $password);
  setcookie ("userdata", $data);
}
```

Figure 18. Example of missing encryption of valuable data by CWE & SANS Institute (2011)

9) Trusting Unsafe Data Entiries

CWE and SANS Institute (2011) defined that "Developers may assume that inputs such as cookies, environment variables, and hidden form fields cannot be modified. However, an attacker could change these inputs using customized clients or other attacks. This change might not be detected. When security decisions such as authentication and authorization are made based on the values of these inputs, attackers can bypass the security of the software. Without sufficient encryption, integrity checking, or another mechanism, any input that originates from an outsider cannot be trusted."

Example Language: **Java**

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
  Cookie c = cookies[i];
  if (c.getName().equals("role")) {
    userRole = c.getValue();
  }
}
```

Figure 19. Example of reliance on untrusted inputs in a security decision by CWE & SANS Institute (2011)

10) Improper Authorization

CWE and SANS Institute (2011) defined that "Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource. When access control checks are incorrectly applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution."

Example Language: **PHP**

```
$role = $_COOKIES['role'];
if (!$role) {
  $role = getRole('user');
  if ($role) {
    // save the cookie to send out in future responses
    setcookie("role", $role, time()+60*60*2);
  }
  else{
    ShowLoginScreen();
    die("\n");
  }
}
if ($role == 'Reader') {
  DisplayMedicalHistory($_POST['patient_ID']);
}
else{
  die("You are not Authorized to view this record\n");
}
```

Figure 20. Example of improper authorization by CWE & SANS Institute (2011)

Figure 20 shows a data to authenticated users by confirming the user's authorization using a cookie.

Detection of the security vulnerabilities mentioned in the literature review may be important for program users or those who own the program. These vulnerabilities were experimentally investigated to support our hypothesis that differences in code review of novices and experts can be identified by eye tracking methodology. The eye tracking method can be employed to provide information about the code review process. In the next section, the methodology of this review will be presented.

CHAPTER 3

METHODOLOGY

The research design of the study, variables and hypotheses, brief description about participants, stimuli used in experiment and participants' task description are reported in this chapter.

3.1. Research Design

The focus of this study is to determine whether the differences in code review of novices and experts can be identified by eye tracking methodology. First of all, the literature review was conducted for the eye-tracking technology and software security vulnerabilities. Since there are few studies related to the effect of software experience on finding vulnerabilities in source code, the research was decided to focus on investigating this topic by using eye-tracking technology principally. After the literature review, the hypotheses of the research were constructed. After that, the stimuli of the study were prepared based on the most dangerous software errors. The development process of the stimuli was explained in the following sections. After the preparation of stimuli, the data collection process started by applying the stimuli to software developers. After data collection, all data were recorded and made ready for statistical analysis. The answers to the demographic data and the information test were examined and compared between the experts and the novices. In the following sections, the results and comparisons of the stimuli will be reported.

3.2. Hypotheses

A hypothesis have been formulated from the research question defined in the study. We expected that the differences in code review of novices and experts can be identified by eye tracking methodology (H1). The hypothesis is presented below:

> H1: Eye tracking can be used to identify differences in a code review of novices and experts.

3.3. The procedure

The data collection procedure of the present study consisted of three sessions, namely demographics, information test, and stimuli session. In the first session, that was demographics, the participants were asked about their software experience, education status, age, gender and occupation. The second session consisted of an information test consisting of questions designed to test participants' basic domain knowledge of software and cybersecurity. In this test, which consisted of 20 questions, there were five answer options for each question, and only one of these options was true. The correct response rate of the participants to this information test and the correct response rate of the participants to the stimuli were two necessary and complementary criteria to determine the participant's software experience.

The third session presented the stimuli shown to the participant in an eye tracking setting. The stimuli consisted of a total of 10 code snippets and four different clusters. The sequence of the code fragments that made up the clusters was different to provide variance within the stimuli. The task in the experiment was to investigate the code fragments that contained security vulnerabilities. Since the task of the participants in this study is visual search, all of the predefined source codes used in the study are vulnerable. In this way, participants were able to perform self-reading tasks.

There were no typographical or logical errors in program codes. The security vulnerabilities in the program codes were selected from among the top twenty-five security vulnerabilities identified by the SANS Institute. The CWE / SANS Top 25 list includes the most common and critical software errors. These errors can lead to serious security vulnerabilities in source code. These software errors could be detected by hackers thoroughly and could be used to put the system at risk. Moreover, these errors may allow attackers to capture the software, steal data, or prevent software from functioning properly, as presented in the previous chapter. The relevant characteristics were presented in Table 1.

Table 1. Program parts information.

Program Language	Vulnerability Description	Total Line Number	Vulnerability Line Numbers.
C#	SQL Injection	9	5-6
C#	Unlimited File Upload	10	6
JSP	Cross-site Scripting	10	4
Java	Missing Authentication for an Important Method	11	3-5
Java	Hard-coded Credentials	9	3
Python	Misuse of Hash and Salt	9	6
C	Unrestiricted Authentication Attempts	14	6-7
PHP	Reliance on Untrusted Inputs in a Security Decision	11	3-5
PHP	Use of a Broken or Risky Cryptographic Algorithm	11	3-4
Python	Execution with Unnecessary Privileges	12	6-8

After the instrument was prepared in the Tobii Studio (the manufacturer software), version 3.4.5, which worked with the eye tracking device, a pilot study was conducted with two participants to inspect whether the code fragments could be understood as intended by the participants. Based on the interpretations of the pilot participants and the data obtained from the eye tracking device, the code fragments were rearranged. The code fragments have been made ready for experimentation after correcting the logic and syntax errors. The code fragments investigated by the participants in the present study were given in Appendix A.

3.4. Data Collection

At the beginning of the experiment, the participants were presented with the instructions. They were also presented a sample question to make them more comfortable and adapted to the experiment. On the left side of the page layout, there was a piece of source code that contained security vulnerabilities, and on the right side of the page layout, there were three security vulnerability options. One of these options was the answer to security weakness in the source code. Figure 21 shows one of the stimuli. The source code in this figure contained a security vulnerability that may allow the use of SQL injection technique. The bottom of the options was the answer to the security vulnerability in the source code.

The eye tracker was calibrated by using the device before the experiment started. While the code reviewer was allowed to ask questions until the instructions and sample question parts, no questions were allowed until the the experiment is over. During the experiment, the participants were asked to say the correct answer with a loud voice while receiving their answer. If they were sure which option was right, they were asked to choose the option that was the most likely.

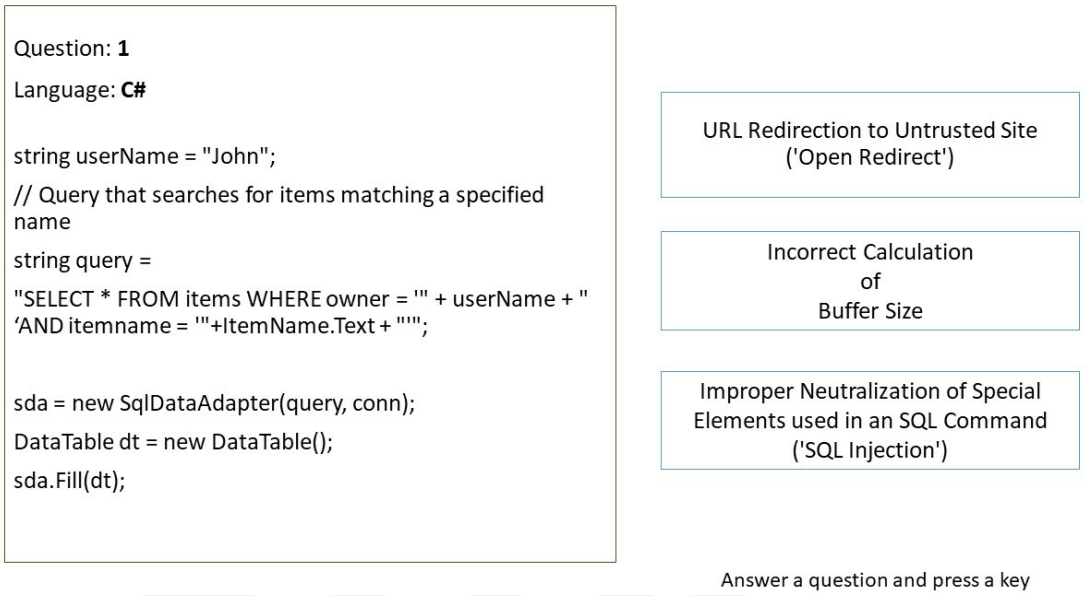

Figure 21. Sample stimulus that contains a security vulnerability, which allows SQL injection technique.

Tobii X2-60 eye-tracking device was used in the present study. No external hardware was used; therefore, the test environment was very similar to the real working environment. Conducting the experiment as close as possible to the actual working environment was important for the accuracy of the result obtained from the experiment. The cameras are on a portable device that connected to the processor box via USB. Since the eye-tracking device was portable, the tests were performed at different locations. Quiet and tranquil places were preferred when conducting experiments to ensure that the participants were not distracted. As the necessary information was given at the beginning of the experiment, participants were not allowed to ask questions during the experiment.

3.5. Participants

Twenty participants with previous programming experience had voluntarily participated in our study. The information test was applied to all participants. In addition, participants filled demographic information forms. The gender of all participants is male and the mean

age of the participants was 31 (SD = 4.14). Participants do not have any vision problem. They used ad-hoc (lightweight) code review technique to investigate program codes. Participants were not informed about the possible outcomes of the experiment.

CHAPTER 4

RESULTS

In this chapter, the results of the statistical analyses were reported. Firstly, the results of the information test were presented and use to categorize participants as novices and experts. Secondly, the records in which the participants investigated the stimuli were examined. Tobii Studio eye tracking software showed participants' eye movements as interconnected points in figure 22. These points were depicted as circles.

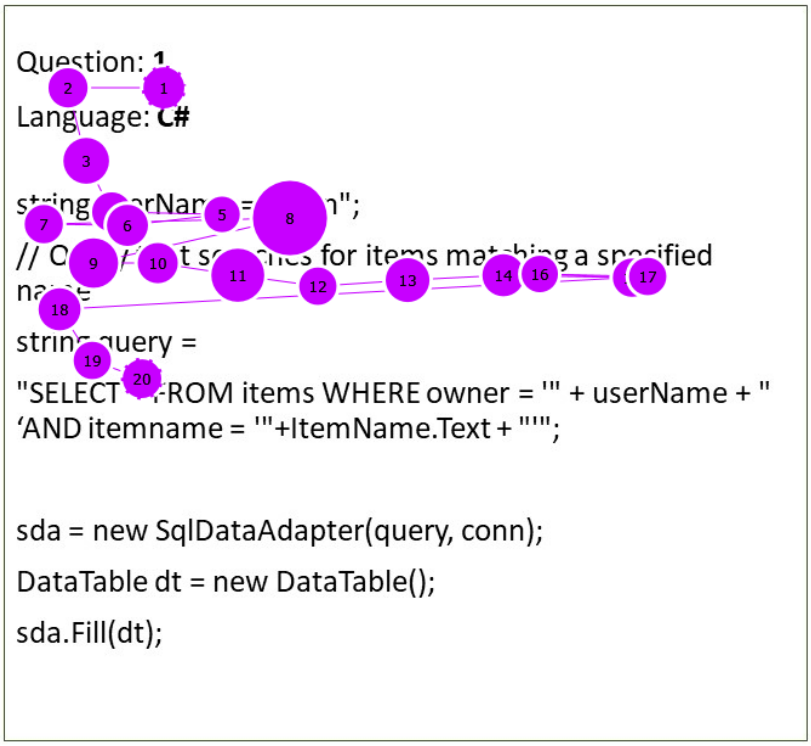

Figure 22. Participant investigates the program

Variables defined in the previous section; scan time, defect reporting time, fixation count and total fixation duration were calculated and analyzed by using Tobii Studio. For novices and experts, these variables were compared with an independent samples t-test. Gaze samples and weighted gaze samples measurements were used to assess the amount of valid gaze samples. Since the sample percentages were high, data cleansing was not performed in the record set.

Multiple-choice information test was conducted to measure participants' expertise in the programming. The information test consisted of 20 questions which are compiled from multiple sources designed to test participants' basic domain knowledge of software and cybersecurity. In this test, there were five answer options for each question, and only one of these options was true. The goal of the test was to divide the participants into two groups as experts and novices. According to the results, the independent sample t-test was made to compare the knowledge level of novices and experts. The results suggest that there was a significant difference between novices (M = 12.90, SD = 0.99) and experts (M =18.10, SD = 1.19) in terms of their accuracy ($t(18)=3.198$, $p=0.005$). These results are presented in Table 2.

Table 2. T-test result for information test accuracy between novices and experts (number of correct answers out of 20)

Mean		T-test		
Novices	Experts	Sig.	df	t
12.90 (0.99)	18.10 (1.19)	0.000	18	3.198

The participants were instructed to report the answer verbally if they found a vulnerability in the stimulus. Accordingly, their answers were used for the calculation of defect reporting time[3]. Defect reporting time was compared with an independent samples t-test.

[3] The time elapsed between the time the participant started to review the code and the time the participant found the vulnerability in the code.

Levene's test showed that the variances for the defect reporting time of the novices and the experts were equal, p = 0.359. The t-test results demonstrated that there was a significant difference between novices (M = 63.70, SD = 15.81) and experts (M = 47.20, SD = 11.32) in terms of their defect reporting time (t(18)=2.680, p=0,015), as shown in Table 3.

Table 3. T-test result for defect reporting time between novices and experts (in seconds)

Mean		T-test		
Novices	Experts	Sig.	df	t
63.70 (18.81)	47.20 (11.32)	0.015	18	2.680

Scan time was calculated by examining the participants' records. The scan time was measured as the time elapsed between the time the participant started to review the code and the time focusing on a smaller subset of lines. Scan time of participants was compared with the independent sample t-test. Levene's test showed that the variances for the scan time of the novices and the experts were equal, p = 0.861. The t-test results demonstrated that there was a significant difference between the mean scan time of the novices (M = 18.10, SD = 5.79) and experts (M = 12.40, SD = 5.41) in terms of their scan time (t(18)=2.248, p=0.037), as shown in Table 4.

Table 4. T-test result for scan time between novices and experts (in seconds)

Mean		T-test		
Novices	Experts	Sig.	df	t
18.10 (5.79)	12.40 (5.41)	0.037	18	2.248

At the end of each trial, the participants verbally reported the answer they thought to be true. Answers were collected by reviewing the records afterward. According to these answers, another independent sample t-test was conducted to compare the accuracy of

finding the security vulnerabilities of the novices and the experts. Levene's test showed that the variances for the accuracy of finding the security vulnerabilities of the novices and the experts were equal, p = 0.184. The t-test results showed that there was a significant difference in accuracy between novices (M = 6.30, SD = 1.88) and experts (M = 8.50, SD = 1.08) in terms of their answers (t(18)=3.198, p=0.005). These results are presented in Table 5.

Table 5. T-test result for accuracy between novices and experts (number of correct answers out of 10)

Mean		T-test		
Novices	Experts	Sig.	df	t
6.30 (1.88)	8.50 (1.08)	0.005	18	3.198

We also had four eye movement measurements. The visit count, fixation count, fixation duration and total fixation duration measurements described in the literature review chapter may give us information about the participants' expertise. Firstly, fixation count measurements were compared with an independent samples t-test. Levene's test showed that the variances for the fixation count of the novices and the experts were equal, p = 0.843. The t-test results demonstrated that there was a significant difference between the fixation count of the novices (M = 160, SD = 37.84) and experts (M = 115, SD = 38.11) in terms of their fixation counts (t(18)=2.651, p=0.016), as shown in Table 6.

Table 6. T-test result for fixation count between novices and experts (number of fixation)

Mean		T-test		
Novices	Experts	Sig.	df	t
160 (37.84)	115 (38.11)	0.016	18	2.651

Secondly, total fixation duration measurements were analyzed using an independent samples t-test. Levene's test showed that the variances for the total fixation duration of the novices and the experts were equal, p = 0.978. The t-test results demonstrated that there was a significant difference between the total fixation duration of the novices (M = 33, SD = 8.94) and experts (M = 24, SD = 9.13) in terms of their total fixation duration (t(18)=2.308, p=0.033), as shown in Table 7.

Table 7. T-test result for total fixation duration between novices and experts (in seconds)

Mean		T-test		
Novices	Experts	Sig.	df	t
33 (8.94)	24 (9.13)	0.033	18	2.308

Thirdly, fixation duration measurements were analyzed using an independent samples t-test. Levene's test showed that the variances for the fixation duration of the novices and the experts were equal, p = 0.743. The t-test results demonstrated that there was no significant difference between the fixation duration of the novices (M = 205.70, SD = 22.73) and experts (M = 219.20, SD = 30.32) in terms of their fixation duration (t(18)=1.126, p=0.275), as shown in Table 8.

Table 8. T-test result for fixation duration between novices and experts (in seconds)

Mean		T-test		
Novices	Experts	Sig.	df	t
205.70 (22.73)	219.20 (30.32)	0.275	18	1.126

Finally, visit count measurements were analyzed using an independent samples t-test. Levene's test showed that the variances for the visit count of the novices and the experts were equal, p = 0.138. The t-test results demonstrated that there was a significant difference between the visit count of the novices (M = 6.55.70, SD = 2.25) and experts (M = 4.79, SD = 1.09) in terms of their visit count (t(18)=2.224, p=0.039), as shown in Table 9.

Table 9. T-test result for visit count between novices and experts (number of visits)

Mean		T-test		
Novices	Experts	Sig.	df	t
6.55 (2.25)	4.79 (1.09)	0.039	18	2.224

The total number of correct answers given by the novices and experts to the eye tracking tests is shown in Table 10 below. These results showed the differences between novices and experts in finding the security vulnerabilities in the source code. A chi-square test of independence was performed to examine the relation between expertise and accuracy scores. The relation between these variables was significant, X^2 (1, N = 200) = 11.60, p <.05.

Table 10. Total Number of Correct Answers in Eye Tracking Tests

	Total Number of Correct Answers	
Code Fragments	Novices	Experts
1 - Sql Injection	6	10
2 - Upload Of File	8	8
3 - Cross Site Scripting	4	8
4 - Missing Auth Critical Function	8	6
5 - Hard Coded Credential	4	8
6 - One Way Hash Without Salt	5	8
7- Improper Restr Excess Auth Attpt	8	9
8 - Missing Encryption Data	7	9
9 - Untrust Inputs Security Decision	6	9
10 - Incorrect Authorization	8	10

As mentioned in the literature review section, the heat map was a visualization method that uses different colors to show how many fixations participants made or how long they were fixated. For example, the red color was used to indicate a higher number of fixations or duration. We used the heat map technique to investigate the differences in the code review of novices and experts. The heat map in figure 22 shows the result of code analysis of the novices. Figure 23 also shows the result of the code analysis of the experts. As can be seen, there was a relatively intense red color in the heat map of the novices. Moreover, colors on the novices' heat map cover a larger area. This result supports the hypothesis that eye tracking can be used to identify differences in a code review of novices and experts.

Figure 22. Heat map of the second question for novices.

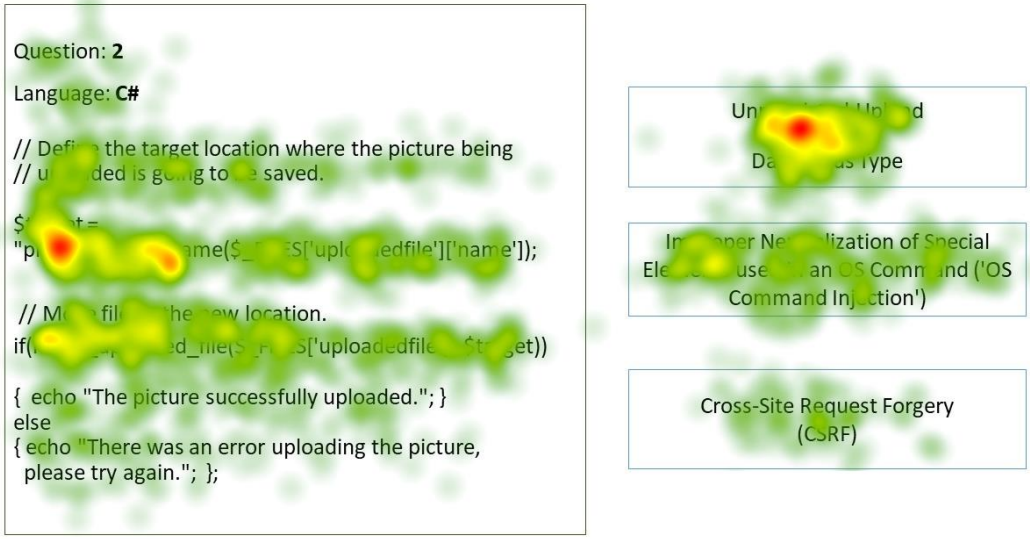

Figure 23. Heat map of the second question for experts

4.1. Summary of the Results

In summary, the results showed that scan time, defect reporting time and accuracy rates of novices and experts were statistically different from each other. The scan time and defect reporting time of the experts were higher than the scan time and defect reporting time of the novices. Also, the accuracy of detecting the security vulnerabilities of the experts was higher than the accuracy of detecting the security vulnerabilities of the novices. Moreover, eye movement measures can be used to investigate the effect of the programming experience on finding security vulnerabilities in the code. Eye movement measurement results showed that the average fixation counts of novices, the average of total fixation duration of novices and the average visit counts of novices are higher than the average of experts. To support our hypotheses, a heat map which is a visualization technique was also used. The colors of the novices' heat map results were relatively more intense and covered a larger area. It indicated that eye tracking can be used to identify differences in a code review of novices and experts

CHAPTER 5

DISCUSSION & CONCLUSIONS

Many organizations are dependent on software to carry out their daily activities. Even if the latest security technologies are used in software systems, most of these systems can still face a large number of security breaches. The main reason for it is software developers making mistakes, mostly due to low skill, personal differences and lack of security knowledge. (Islam & Dong, 2008). Secure programming depends on personal knowledge and experience of software developers involved in software development stages. Therefore, eye tracking systems that can help us better understand individual differences and human factors can be used in software programming.

The focus of this study is to determine whether the differences in code review of novices and experts can be identified by eye tracking methodology. To this end, the effect of the developer's experience on the detection of security vulnerabilities was investigated by using the eye tracking methodology. In this section, the results were evaluated and discussed within the scope of previous studies. Then, the outcomes of this research were interpreted according to the results of the experiment. The results of the eye tracking experiment and information test were evaluated by considering the accuracy results. Finally, the limitations and shortcomings of this study were presented, and recommendations for future works are given.

5.1. Discussion

The results of accuracy analysis comparing the novices and experts were presented in Chapter 4. Accuracy, which was calculated in the present study, was the number of correct

answers in the Information Test and the eye tracking experiments. Both the Information Test and the experiment results showed that the accuracy scores of the experts were higher than the accuracy scores of the novices. A code review task using eye tracking methodology was conducted in order to evaluate the performance of the participants comparatively. The result supports the hypothesis that eye tracking can be used to identify differences in a code review of novices and experts.

The results of scan time analysis comparing the novices and experts were presented in Chapter 4. Scan time was measured as the time elapsed between the time the participant started to review the code and the time focusing on a smaller subset of lines. The scan time of the experts was shorter than the scan time of the novices. The result supports the hypothesis that eye tracking can be used to identify differences in a code review of novices and experts.

The results of defect reporting time analysis comparing the novices and experts were presented in Chapter 4. Defect reporting time was measured as the time elapsed between the time the participant started to review the code and the time the participant found the vulnerability in the code. The defect reporting time of the experts was shorter than the defect reporting time of the novices. The result supports the hypothesis that eye tracking can be used to identify differences in a code review of novices and experts.

The fixation count, total fixation duration and gaze shift eye movement measurements' analyzes were also reported. Fixation count measures the number of gaze fixations on an area of interest and total fixation duration measures the sum of the duration for all fixations within an area of interest (Liversedge, Gilchrist, & Everling, 2011). Gaze shift measures the movement from one area of interest to another (Holmqvist & Andersson, 2017). In the present study, the area of interest was assumed to be the part of the display where the question stimuli were presented to the participants. According to findings, the fixation count, total fixation duration and gaze shifts of the experts were lower than the fixation

count, total fixation duration and gaze shift of the novices. The result supports the hypothesis that eye tracking can be used to identify differences in a code review of novices and experts.

Some eye movement measurements, such as fixation duration, could be used for the purpose of this study, but some eye movement measurements, such as the fixation count, could not be used to achieve the expected results. In addition to the dependent variables and eye movement measurements, the differences between the code review of the novices and the experts can be determined by investigating the source code analysis records of the participants. While novices tended to review the source code from beginning to end, experts tended to focus on a smaller subset of lines that could contain the vulnerability. In addition, after reviewing the source code and options, novices tend to review the source code again and again. However, after examining the source code, experts usually tend to choose one of the options without having to return to the code again.

5.2. Contribution to Research Literature

The experiment was conducted with 20 participants with different programming experience. The participants are university graduates, and almost all of them are employed in ICT companies. Most of the previous studies, however, were carried out with five or fewer people with 3 or 4 years of programming experience. In addition, an average of five program code fragments written in the same programming language was used in most of the previous studies. In this study, ten different program code fragments written in Python, Javascript, C #, Java and C programming languages were used. Thus, the effect of different programming languages on finding the vulnerabilities in the code was investigated.

5.3 Limitations and Future Research

One of the limitations of this study was that participants could investigate predefined code fragments of the programs that include security vulnerabilities, which limits the

generalizability of the findings. Instead of code fragments known to contain vulnerabilities, a more general result can be obtained in future studies using a larger code fragment of software in use. The size of the program codes was also limited since the area displayed to the user was limited to the screen size of the computer and the program of eye tracker used for the experiment did not allow the participant to scroll down the page. Furthermore, the fact that all of the participants were male was a limitation for this study.

In order to find bugs and vulnerabilities in the program code, many developers use programs that perform code analysis. Thanks to these programs, code analysis can be done much faster than software developers can do. Code analysis programs can save time, but not all bugs and security vulnerabilities may be detected. With the findings obtained from the studies such as present study, it is possible to improve these code analysis programs by adding human factors to the algorithms of code analysis programs.

In the eye tracking analysis, the area of interest was determined as the whole piece of code. In future studies, the area of interest may be narrowed down as a sub-section leading to the vulnerability. For this sub-section, differences between novices and experts can be investigated. The review time of these sub-sections can be examined comparatively between novices and experts.

REFERENCES

Begel, A. (2015). Eye Movements During Code Review. In *Proceedings of the Third International Workshop on Eye Movements in Programming*, Joensuu, Finland, pp. 3-4.

Bergstrom, J. R., & Schall, A. J. (2014). *Eye Tracking in User Experience Design.* Lund, Sweden: Lund Eye- Tracking Research Institute.

Busjahn, T., Schulte, C., Sharif, B., Simon, Begel, A., Hansen, M., Bednarik, R., Orlov, P., Ihantola, P., Shchekotova, G. & Antropova, M. (2014). Eye Tracking in Computing Education. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, United Kingdom, pp. 3-10.

Bylinskii, Z., Borkin, M.A., Kim, N.W., Pfister, H., Oliva, A. (2017). Eye Fixation Metrics for Large Scale Evaluation and Comparison of Information Visualizations. In: Burch M., Chuang L., Fisher B., Schmidt A., Weiskopf D (Eds.), *Eye Tracking and Visualization. ETVIS 2015* (pp. 85). Mathematics and Visualization. Springer, Cham.

Common Weakness Enumeration. (2018, March 29). Retrieved Jan 16, 2018, from https://cwe.mitre.org/data/slices/2000.html

Duchowski, A.T. (2017). *Eye Tracking Methodology.* Cham, Switzerland: Springer International Publishing AG.

ECDL. (2013, December). ECDL Module 1 Sample Test. Retrieved Feb 07, 2018, from https://www.slideshare.net/stacio/modul-1sampletest

EnSurePass. (n. d.). CompTIA Security+ JK0-018 Exam. Retrieved Feb 05, 2018, from http://www.dumps4shared.com/wp-content/uploads/2014/07/Latest-CompTIA-EnsurePass-JK0-018-Dumps-PDF.pdf. (2002, February).

Hofmeister, J., Bauer. J., Siegmund, J., Apel, s. (2017). Comparing Novice and Expert Eye Movements during Program Comprehension. In *Proceedings of the Fourth International Workshop on Eye Movements in Programming*, Magdeburg, Germany, pp. 17-18.

Holmqvist, K., & Andersson, R. (2017). *Eye tracking: A comprehensive guide to methods, paradigms and measures*. Waltham, USA: Elsevier Inc.

Horsley, M., & Eliot, M., Knight, B., Reilly, R. (2014). *Current Trends in Eye Tracking Research*. Cham, Switzerland: Springer.

Huizinga, D., & Kolawa, A. (2007). *Automated defect prevention: best practices in software management.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Islam, S. & Dong, W. (2008). Human Factors in Software Security Risk Management. In *Proceedings of the first international workshop on Leadership and management in software architecture*, Leipzig, Germany, pp. 13-16.

Liversedge, S.P., Gilchrist, I.D., Everling, S. (2011). *The Oxford Handbook of Eye Movements*. Great Clarendon Street, Oxford: Oxford University Press.

OWASP, (2011, August). OWASP Top 10 Threats and Mitigations Exam – Single Select. Retrieved Jan 10, 2018, from https://www.owasp.org/index.php/OWASP_Top_10_Threats_and_Mitigations_Exam_-_Single_Select

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological bulletin, 124 3*, pp. 372-422.

Peachock, P. & Sharif, B. (2017). Investing Eye Movements in Natural Language and C++ Source Code – A Replication Experiment. In *Proceedings of the Fourth International Workshop on Eye Movements in Programming*, Yougstown, Ohio, pp. 4-5.

ProProfs. (2012, October). Basic Computer Questions. Retrieved Feb 16, 2018, from https://www.proprofs.com/quiz-school/story.php?title=basic-computer-questions

ProProfs. (2017, August). IT Security Quiz. Retrieved Feb 16, 2018, from https://www.proprofs.com/quiz-school/story.php?title=it-security-quiz

Ann, K. (n. d.). Internet Ethics. Retrived Jan 16, 2018, from https://quizlet.com/1086399/internet-ethics-flash-cards/

SANS Institute. (2011). CWE/SANS Top 25 Most Dangerous Software Errors. *Insecure Interaction Between Components.* Retrieved from https://www.sans.org/top25-software-errors/

SANS Institute. (2011). CWE/SANS Top 25 Most Dangerous Software Errors. *Risky Resource Management.* Retrieved from https://www.sans.org/top25-software-errors/

SANS Institute. (2011). CWE/SANS Top 25 Most Dangerous Software Errors. *Porous Defenses.* Retrieved from https://www.sans.org/top25-software-errors/

Sharafi, Z., Soh, Z., Gueheneuc, Y. & Antoniol, G. (2012). Women and Men – Different but Equal: On the Impact of Identifier Style on Source Code Reading. In *Proceedings of 20th IEEE Intl. Conf. on Program Comprehension*, Germany, pp. 27-36.

Sharafi, Z., Zephyrin, S. & Gueheneuc, Y. (2015). A Systematic Literature Review on the Usage of Eye-tracking in Software Engineering. In *Information and Software technology*, 67, 79-107.

Sharif, B., Falcone, M. & Maletic, J. I. (2012). An Eye-tracking Study on the Role of Scan Time in Finding Source Code Defects. In *Proceedings of the Symposium on Eye Tracking Research and Applicaitons*, California, pp. 381-384.

TechTarget. (n. d.). Quiz #1: Help Desk Basics. Retrieved Feb 05, 2018, from https://searchmicroservices.techtarget.com/news/763878/Quiz-1-Help-Desk-Basics

TechTarget. (n. d.). Quiz #20: Cryptography. Retrieved Feb 05, 2018, from https://searchmicroservices.techtarget.com/quiz/Quiz-20-Cryptography ECDL. (2002, February).

TechTarget. (n. d.). Quiz #20: Cryptography. Retrieved Feb 05, 2018, from https://searchsecurity.techtarget.com/definition/Quiz-Cryptography. (200, April).

Tvarozek, J., Konopka. M., Navrat, P., Bielikova, M. (2015). Studying Various Source Code Comprehension Strategies in Programming Education. In *Proceedings of the Third International Workshop on Eye Movements in Programming*, Bratislava, Slovakia, pp. 25-26.

Turner, R., Falcone, M., Sharif, B. & Lazar, A. (2014). An eye-tracking study assessing the comprehension of c++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applicaitons*, Florida, pp. 231-234.

Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In *Proc.of Symp. On Eye tracking research&applications*, San Diego, pp.133-140

Uwano, H., Monden, A., & Matsumoto, K. (2008). DRESREM 2: An Analysis System for Multi-document Software Review Using Reviewers' Eye Movements. In *Proc. of Intl. Conf. on Software Engineering Advances*, Malta, pp. 177 – 183

APPENDICES

APPENDIX A

“Question: **1**

Language: **C#**

```
string userName = "John";

// Query that searches for items matching a specified name

string query =

"SELECT * FROM items WHERE owner = '" + userName + " ‘AND itemname
= '"+ItemName.Text + "'";

sda = new SqlDataAdapter(query, conn);

DataTable dt = new DataTable();

sda.Fill(dt);”
```

“Question: **2**

Language: **C#**

```
// Define the target location where the picture is going
//to be saved.
$target= "pictures/".basename($_FILES['uploadedfile']['name']);

 // Move file to the new location.

if(move_uploaded_file($_FILES['uploadedfile'], $target))

{  echo "The picture successfully uploaded."; }
else
{ echo "There was an error uploading the picture,
  please try again.";  };”
```

“Question: **3**

Language: **JSP**

```
//Query a database for an employee with a given ID

<%Statement stmt = conn.createStatement();
  ResultSet result=

  stmt.executeQuery("select * from Emp where id="+eid);

  if (result != null) {

     result.next();
     String name = result.getString("name");
%>


  // Prints the corresponding employee's name
  Employee Name: <%= name %>”
```

“Question: **4**

Language: **Java**

```
// Create a BankAccount object for a bank management application

public BankAccount createBankAccount(String
        accountNumber, String accountName,        double balance) {

    BankAccount account = new BankAccount();

    account.setAccountNumber(accountNumber);

    account.setAccountOwnerName(accountName);

    account.setBalance(balance);

    return account;

}”
```

“Question: **5**

Language: **Java**

```
// Attempt to verify a password
public boolean VerifyAdmin(String password) {
 if (password.equals("68af404b5130b6f226c63e6b"))
{
  System.out.println("Entering Diagnostic Mode...");
  return true;
}
  System.out.println("Incorrect Password!");
  return false;”
```

“Question: **6**

Language: **Python**

```
// New user provides a new username and password to create an account
def storePassword( "John", "123456"):
    hasher = hashlib.new('md5')
    hasher.update("123456")
    hashedPassword = hasher.digest()
    // UpdateUserLogin returns True on success
    return
    updateUserLogin("John", hashedPassword)”
```

“Question: **7**

Language*:* **C**

```
int validateUser(char *host, int port){

 int socket = openSocketConnection(host, port);
 int isValidUser = 0;
 char username[USERNAME_SIZE];
 char password[PASSWORD_SIZE];
 while (isValidUser == 0) {

   if (getNextMsg(socket,username,USERNAME_SIZE) > 0){

    if(getNextMsg(socket, password,PASSWORD_SIZE) > 0) {
         isValidUser = AuthenticateUser (username,
           password);
     }

   }

 } return(SUCCESS);
}”
```

“Question: **8**

Language: **PHP**

```
// It performs authentication by checking if a cookie has been set

$auth = $_COOKIES['authenticated'];

if (! $auth) {

   if (AuthenticateUser($_POST['user'],  $_POST['pswrd']){

       setcookie("authenticated", "1", time());}

   else {

      ShowLoginScreen();

      die("\n");

}  DisplayMedicalHistory($_POST['patient_ID']);”
```

“Question: **9**

Language: **PHP**

```
function encryptPassword($password)

{

   $iv_size = mcrypt_get_iv_size(MCRYPT_DES,
         MCRYPT_MODE_ECB);

   $iv = mcrypt_create_iv($iv_size,  MCRYPT_RAND);
   $key = "This is a password encryption key";


   $encryptedPassword =       mcrypt_encrypt(MCRYPT_DES, $key,
                  $password, MCRYPT_MODE_ECB, $iv);

   return $encryptedPassword;
}”
```

“Question: **10**

Language: **Python**

```
  def makeNewUserDir(username):

     if invalidUsername(username):

        print(‘Cannot contain invalid characters')
        return False
     try:

        raisePrivileges()
        os.mkdir('/home/' + username)
        lowerPrivileges()
     except OSError:

        print('Unable to create new user directory')
        return False
     return True”
```

APPENDIX B

Information Test

1. This is the smallest unit of data in a computer.

   a) bit
   b) exabyte
   c) bogomip
   d) byte
   e) nit

2. This is the standard interactive and programming language for getting information from a database.

   a) SQL
   b) SSL
   c) DCOM
   d) PL/1
   e) DASD

3. Which of the following is not a high-level programming language?

   a) x86
   b) Python
   c) C#
   d) Ruby
   e) C

4. Of the following if statements, which one correctly executes three instructions if the condition is true?

   a) If (x < 0) a = b * 2; y = x; z = a – y;
   b) { if (x < 0) a = b * 2; y = x; z = a – y; }
   c) If{ (x < 0) a = b * 2; y = x; z = a – y ; }
   d) If (x < 0) { a = b * 2; y = x; z = a – y; }

5. The variable "i" defined in many programs is most commonly used for:

   a) Allocating Arrays
   b) Initializing Variables
   c) Preventing Collisions
   d) Loop Operations

6. Cryptography supports all of the core principles of information security except

   a) Availability
   b) Confidentiality
   c) Integrity
   d) Authenticity

7. Which one of the following describes how confidential information should be sent using an unsecured network?

   a) In an unsigned email.
   b) In an encrypted format.
   c) In a compressed format.
   d) In an attachment

8. Which of the following ensures a user cannot deny having sent a message?

   a) Availability
   b) Integrity
   c) Non-repudiation
   d) Confidentiality

9. An example of a complex password is:

   a) password
   b) winston
   c) MdCi3yo!
   d) 12345678

10. Encryption is best described as:

    a) A method hackers use to access information
    b) A particular brand of laptops produced by Sony
    c) The translation of data into a secret code
    d) A punishment that employers can put on their employees who violate a policy

11. What type of protocol would you expect to see in a URL for a secure document?

a) https b) ftps c) hits d) hashes

12. Fill in the blank of the following statement: "_____ encryption is a method of encryption involving one key for both encryption and decryption."

a) Symmetric
b) Asymmetric
c) Public key
d) SSL

13. Which of the following web application security weaknesses can be mitigated by preventing the use of HTML tags?

a) LDAP injection
b) SQL injection
c) Error and exception handling
d) Cross-site scripting

14. In password protection, this is a random string of data used to modify a password hash.

a) Sheepdip
b) Salt
c) Bypass
d) Dongle

15. Which of the following script is an example of a SQL injection attack?

a)
```
var Shipcity;
ShipCity = Request.form ("ShipCity");
var SQL = "select * from OrdersTable where ShipCity = '" + ShipCity + "'";
```

b)
```
var Shipcity;
ShipCity = Request.form ("ShipCity");
```

c)
```
var Shipcity;
var SQL = "select * from OrdersTable where ShipCity = '" + ShipCity + "'";
```

16. Today, many Internet businesses and users take advantage of cryptography based on this approach.

a) Public key infrastructure
b) Output feedback
c) Encrypting File System
d) Single sign-on

17. The two methods of encrypting data are

a) Substitution
b) Block and stream
c) Symmetric and asymmetric
d) DES and AES

18. What is a hash in Computer terms?

a) Potatoes
b) An encrypted value
c) A decryption key
d) None of them

19. Which of the following is the best way to prevent malicious input exploiting your application?

a) Input validation using an allow List
b) Using encryption
c) Using table indirection
d) Using GET/POST parameters

20. You should set a secure flag in a cookie to ensure that:

a) The cookie is a persistent cookie.
b) The cookie is not available to client script.
c) The cookie is sent over an encrypted channel.
d) The cookie is deleted when the user closes the browser.

APPENDIX C

Demografik Bilgi Formu

1. Doğum Yılı : _____

2. Cinsiyet :
   a) Erkek
   b) Kadın

3. Eğitim düzeyiniz :
   a) Lise Mezunu
   b) Lisans Öğrencisi
   c) Yüksek Lisans Öğrencisi
   d) Doktora Öğrencisi
   e) Doktora Mezunu

4. Mesleğiniz :
   a) Öğrenci
   b) Akademisyen
   c) Bilgisayar Mühendisi
   d) Makine Mühendisi
   e) Elektronik Mühendisi
   f) Siber Güvenlik Uzmanı
   g) Diğer

5. Meslekte çalıştığınız süre nedir :
   a) 1 yıldan az
   b) 1-3 yıl arası
   c) 3-6 yıl
   d) 6-9 yıl
   e) 9 yıl ve üzeri

TEZ İZİN FORMU / THESIS PERMISSION FORM

ENSTİTÜ / INSTITUTE

Fen Bilimleri Enstitüsü / Graduate School of Natural Sciences ☐

Sosyal Bilimler Enstitüsü / Graduate School of Social Sciences ☐

Uygulamalı Matematik Enstitüsü / Graduate School of Mathematics ☐

Enformatik Enstitüsü / Graduate School of Informatics ☐

Deniz Bilimleri Enstitüsü / Graduate School of Marine Sciences ☐

YAZARIN / AUTHOR

Soyadı / Surname : .....
Adı / Name : .....
Bölümü/Department : .....

TEZİN ADI / TITLE OF THE THESIS (**İngilizce** / English) :

.....

.....

TEZİN TÜRÜ **Yüksek Lisans** / Master ☐ **Doktora** / PhD ☐

1. **Tezin tamamı dünya çapında erişime açılacaktır.** / Release the entire work immediately for access worldwide. ☐
2. **Tez iki yıl süreyle erişime kapalı olacaktır.** / Secure the entire work for patent and/or proprietary purposes for a period of **two year. *** ☐
3. **Tez altı ay süreyle erişime kapalı olacaktır.** / Secure the entire work for period of **six months**. ***** ☐

***** *Enstitü Yönetim Kurulu Kararının basılı kopyası tezle birlikte kütüphaneye teslim edilecektir. A copy of the Decision of the Institute Administrative Committee will be delivered to the library together with the printed thesis.*

Yazarın imzası / Signature ..... **Tarih** / Date .....