STORAGE ALLOCATION

Storage Allocation ~~in Compiler Design~~

In Computer Science, Storage Allocation means a process of assigning memory addresses to different data structures, such as variables, arrays, and objects. Assigning memory locations in a computer's memory is one of the important tasks that is performed by compilers and operating systems.

Choosing the right strategies for storage allocation is very important as a strategy can affect the performance of the software of the computer.

There are several storage allocation strategies in compiler design are following:

1. Static Allocation
2. Dynamic Allocation
3. Hybrid Allocation

All of these allocations are done automatically by the compiler. We will discuss each of these strategies in detail with examples in the below sections.

Static Allocation

In Static Allocation, All variables that are created will be **assigned to memory locations at compile time**, and the addresses of these variables will remain the same *throughout* the program's execution. As the size of variables does not vary much so *this allocation is* easy to understand and efficient, but this allocation is *not flexible and scalable.*

C, C++, and Java support static allocation through the use of the *"static" keyword.*

Advantages:

- Static Allocation is simple and easy to understand.
- As the memory is being allocated at compile time, so there will be no additional *time needed in run time.*
- Debugging can be easier for developers as the memory is allocated at compile time.

Disadvantages:

- Variables that are dynamic can not be handled using static allocation.
- No Flexibility and Scalability.

Here's an example of static allocation:

```
int a = 10;
static int b = 1;
const int x = 9;
```

// In all the above three examples, the memory will be assigned to each of them at the program startup, and the addresses of these variables will remain the same in the memory throughout the lifetime of the program.

In programming, There are several types of variables, such as local variables, global

Password-Based Authentication and The Experiences of End Users

Assumpta Ezugwu[1], Elochukwu Ukwandu[2,*], Celestine Ugwu[1], Modesta Ezema[1], Comfort Olebara[3], Juliana Ndunagu[4], Lizzy Ofusori[5], Uchenna Ome[1]

[1]Department of Computer Science, Faculty of Physical Science, University of Nigeria, Nsukka, Enugu State, Nigeria
[2]Department of Applied Computing and Engineering, Cardiff School of Technologies, Cardiff Metropolitan University, Wales, United Kingdom
[3]Department of Computer Science, Faculty of Physical Science, Imo State University, Owerri, Imo State, Nigeria
[4]Department of Computer Science, Faculty of Sciences, National Open University of Nigeria, Abuja, Nigeria
[5]School of Management, Information Technology and Governance, University of KwaZulu-Natal, WestVille Campus, Durban, South Africa

Author's email addresses: ({assumpta.ezugwu, celestine.ugwu, modesta.ezema, uchenna.ome}@unn.edu.ng), eaukwandu@cardiffmet.ac.uk, chy_prime@yahoo.com, jndunagu@noun.edu.ng, lizzyofusori@yahoo.co.uk
[*]**Correspondent Author:** Elochukwu Ukwandu (eaukwandu@cardiffmet.ac.uk)

Abstract

Passwords are used majorly for end-user authentication in information and communication technology (ICT) systems due to its perceived ease of use. The use for end-user authentication extends through mobile, computers and network-based products and services. But with the attendant issues relating to password hacks, leakages, and theft largely due to weak, reuse and poor password habits of end-users, the call for passwordless authentication as alternative intensifies. All the same, there are missing knowledge of whether these password-based experiences are associated with societal economic status, educational qualification of citizens, their age and gender, technological advancements, and depth of penetration. In line with the above, understanding the experience of end-users in developing economy to ascertain their password-based experience has become of interest to the researchers. This paper aims at measuring the experience of staff and students in University communities within southeastern Nigeria on password-based authentication systems. These communities have population whose age brackets are majorly within the ages of 16 and 60 years; have people with requisite educational qualifications ranging from Diploma to Doctorate degrees and constitutes good number of ICT tools consumers. The survey had 291 respondents, and collected data about age, educational qualifications, and gender from these respondents. It also collected information about their password experience in social media network, online shopping, electronic health care services, and internet banking. Our analysis using SPSS and report by means of descriptive statistics, frequency distribution, and Chi-Square tests showed that account compromise in the geographical area is not common with the respondents reporting good experience with passwords usage. Furthermore, this experience is not in any way related to their age (under 60), and educational qualification. Our experiment did not measure the entropy of end-users' passwords, their password hygiene culture and so cannot relate this experience with the strengths of their passwords nor that of their password hygiene culture. The outcome and recommendations of this research will help inform policy and research direction towards password hygiene culture, management, and the potentials or otherwise of passwordless authentication systems in developing economies.

Keywords: Password-based authentication, cyber-hygiene culture, End-User Experience, Cyber-attack, educational qualification, age, gender.

variables, and parameter variables, and the compiler allocates them in memory based on their types. We will discuss all of these in the below subsections.

Global Allocation

It is a type of allocation that comes under static allocation, where the **memory is allocated to the global variables**. Global Variables are the variables that are declared outside of any function and can be accessed in the program from any block of the code. A block of memory is allocated to the global variables in the static memory.

Here's an example of global allocation in C++:

```
#include<iostream>
using namespace std;

int x = 5;
void func() {
   print(x);
}

int main() {
   print(x);
   return 0;
}

// Here 'x' is a global variable that can be accessed from any block of code, and the memory is
allocated for 'x' in static memory.
```

Local Allocation

It is a type of allocation that comes under static allocation, where the **memory is *allocated to the local variables*.** Global Variables are the variables that are declared in *a function*, and their *lifetime* and scope are limited to that function means they can only *be accessed within* the function. A block of memory is allocated to the local variables in *either static memory* or stack memory.

Here's an example of local allocation in C++:

```
void func() {
   int x = 5;
   print(x);
}

// Here 'x' is a local variable that can only be accessed within the function 'func', and the
memory is allocated for 'x' in static memory or stack memory.
```

Parameter Allocation

It is a type of allocation that comes under static allocation, where the **memory is allocated to the parameters**. Parameters are the variables that are passed to the functions and can be accessed in the function it passed. A block of memory is allocated to the global variables in the stack memory.

Here's an example of parameter allocation in C++:

```
void func(int x) {
   print(x);
}
```

1. Introduction

Good number of literatures exists on passwords such as that of Renaud, Otondo, & Warkentin, (2019) on effect of endowment on password strength; AlSabah, Oligeri, & Riley, (2018) on culture and password, and Furnell (2022) on assessing website password practices. Majority of these have shown that passwords, especially text-based are the most popular used authentication method for end-users of information system in computers and network-based products and services (Shay *et al.*, 2010). However, due to unhygienic practices such as choosing weak passwords and insecure management of passwords, they are highly vulnerable to exploitation by both internal and external threat actors. In the other hand, the attraction to the use of passwords as argued by Sharma *et al.*, (2010) lies mostly in its simplicity, practicality, ease of use and low cost rather than the security. But are passwords simple to use, of low cost, easy to use and insecure? Authors of this paper wants to submit that the simplicity of passwords use could depend on so many factors such as age, health condition of the user, number of user's application that requires password protections as well as use of password utilities like password managers. Older generations, people living with dementia, end users with multiple online accounts will have challenges managing multiple passwords. Having multiple passwords can be complicated, prone to have issues related to remembering multiple passwords, temptation of reuse of single login credential on multiple accounts and so on.

The security of assets secured using passwords largely depends on the ability of the user to cultivate good password habits such as regular change of passwords, non-reuse of login credentials on multiple systems, and use of strong passwords derived by using multiple characters with special symbols and numbers as well as the length of it. All the same, cultivating some of these habits have been made easier using password managers. Although several studies have explored consumers experiences with password-based authentication system in developed economies (Butler and Butler, 2015; Bilgihan *et al.*, 2016; Morrison *et al.*, 2021) and the call and rollout of passwordless authentication use cases as alternative intensifies (Jakkal, 2021). For instance, in 2021, Microsoft posits that the development and roll-out of passwordless authentication system is the future of user access management in computer-based systems (Jakkal, 2021). This involves any method that helps to identify a user without the use of password. However, there is a paucity of research focusing on measuring password experience of end-users in developing economies such as Nigeria and relating same with the existing knowledge. This represents a gap in the literature and gives an opportunity for this study to address.

Recent studies by Ugwu *et al.*, (2022) in Nigeria suggest that age and educational qualification have no effect on personal cyber hygiene culture of information and communication technologies (*ICTs*) users. Further studies by Ugwu *et al.*, (2023) in Nigeria on the relationship between cyber hygiene culture of Internet users with gender, employment status and academic discipline did not establish any significant relationship between these dependent and independent variables. These studies by Ugwu *et al.*; (2022) and Ugwu *et al.*; (2023) having been done in Nigeria provided more basis for further study – a study that will establish the likelihood or otherwise of a relationship between cyber hygiene culture and password use experience in developing economies. In line with these, this study focuses on finding:

1. What are the end-user experiences in password-based authentication in Southeastern Nigeria?
2. Does age have impact on end-user experience in password-based authentication of this populace?
3. Does gender have impact on end-user experience in password-based authentication of this populace?
4. Does educational level have impact on end-user experience in password-based authentication of this populace?

```
// Here 'x' is a parameter that is passed to the function 'func' and can only be accessed within the
'func', and the memory is allocated for 'x' in stack memory.
```

Dynamic Allocation

Dynamic Allocation is a type of Storage Strategies in Compiler Design where the **memory is allocated to the data structures at run-time**, not at compile-time. A data structure can be allocated in memory depending on the program's need and can be allocated and deallocated at the program's execution.

In Dynamic Allocation, we have further two allocation strategies, are Stack-Based Allocation and Heap-Based Allocation, explained below:

Stack-Based Allocation

In Stack-Based Allocation, A fixed-sized stack data structure is allocated to a memory location during the run-time execution. A variable is assigned a memory location on the stack when they are declared and can be freed when the variable's scope goes out. C and C++ both support the stack-based allocation through the variables declared in the functions, or Standard Library functions can be used, such as *"alloca()" and* "alloca_array()".

Advantages:

- The process of assigning the memory locations is fast.
- Memory Deallocation is also fast in stack-based allocation.
- Function calls can be efficient using stack-based allocation.

Disadvantages:

- The amount of memory is limited for stack-based allocation.
- As the size of the stack is limited, issues like stack overflow can happen.

Here's an example of stack-based allocation:

```
void fun(int a, int b) {
        int c = a + b;
        print(c);
}

/*
        Whenever the function gets called, a memory location will be assigned to the variable '
c' on the stack.
*/
```

Heap-Based Allocation

In Heap-Based Allocation, Variables that are created and memory will be **dynamically allocated in memory (heap) at run-time execution**. The variables that use heap-based allocation are generally known as dynamic variables that can be changed anytime during the run-time.

Storage Allocation in Compiler Design

In Computer Science, Storage Allocation means a process of assigning memory addresses to different data structures, such as variables, arrays, and objects. Assigning memory locations in a computer's memory is one of the important tasks that is performed by compilers and operating systems.

Choosing the right strategies for storage allocation is very important as a strategy can affect the performance of the software of the computer.

There are several storage allocation strategies in compiler design are following:

1. Static Allocation
2. Dynamic Allocation
3. Hybrid Allocation

All of these allocations are done automatically by the compiler. We will discuss each of these strategies in detail with examples in the below sections.

Static Allocation

In Static Allocation, All variables that are created will be **assigned to memory locations at compile time**, *and the addresses of these variables will remain the same throughout the program's execution. As the size of variables does not vary much so this allocation is easy to understand and efficient, but this allocation is not flexible and scalable.*

C, C++, and Java support static allocation through the use of the "static" keyword.

Advantages:

- *Static Allocation is simple and easy to understand.*
- *As the memory is being allocated at compile time, so there will be no additional* time needed in run time.
- Debugging can be easier for developers as the memory is allocated at compile time.

Disadvantages:

- Variables that are dynamic can not be handled using static allocation.
- No Flexibility and Scalability.

Here's an example of static allocation:

```
int a = 10;
static int b = [illegible]
const int x = 9[illegible]

// In all the above three examples, the memory will be assigned at
startup, and the addresses of these variables will remain the same throughout the
lifetime of the program. //
```

In programming, there are several types of variables, such as local variables, global

C and C++ both support heap-based allocation, and dynamic data structures can be created using functions like. “malloc()”, “calloc()”, “realloc()”, or “new” keywords.
Java also supports heap-based allocation through the use of new keywords.

Advantages:

- Dynamic Data Structures can be handled by heap-based allocation.
- Heap-Based Allocation gives more flexibility to a data structure.
- It can also handle large amounts of memory as compared to stack-based allocation.

Disadvantages:

- Heap-Based Allocation is slower than Stack-Based in terms of speed.
- Risk of memory leaks.

Here’s an example of heap-based allocation:

```
int* arr1 = (int* malloc(5 * sizeof(int)));
int* arr1 = new int[5];

 /*
 In both these examples, the memory will be assigned to each of them at the program startup. and
 the memory locations will be assigned to both variables ‘arr1’ and ‘arr2’ in the memory pool of t
  he heap.
  */
```

Hybrid Allocation

This is a type of allocation strategy that is **used to achieve multiple memory allocation**. A compiler can use static memory allocation for global variables and heap memory allocation for dynamic data structures.

Advantages:

- The benefits of stack-based allocation and heap-based allocation can be taken.
- Hybrid Allocation is more efficient than any other allocation alone.
- The performance of the program gets better than static allocation and dynamic allocation.

Disadvantages:

- As Hybrid allocation is a sort of combination of allocations, It is more complex.
- Hybrid allocation can be difficult for developers to debug the code.

Here’s an example of hybrid allocation:

```
int x;
void func() {
   int a = 2;
```

(52.6%) have not. Also, those who did not notice if such compromise occurred and those who cannot remember its occurrence are 25 (8.6%) and 20(6.9%) respectively.

To further ascertain the respondents' password-based authentication experiences, the respondents were presented with the question: **How has your experience been using Password authentication?** Table 3 below gives the self-report frequency:

Table 3: Frequency Distribution of Respondents End-User Experience (Rating)

		Frequency	Percent	Cumulative Percent
Valid	Excellent	126	43.3	43.3
	Good	139	47.8	91.1
	Average	23	7.9	99.0
	Poor	3	1.0	100.0
	Total	**291**	**100.0**	

Only 3 (1%) of the 291 respondents reported having a poor experience with Password authentication *while 126 (43.3%) reported* excellent Password authentication experience, 139 (47.8%) and 23 (7.9%) *reported good and average experiences respectively.* This agrees with the report from Table 2, where *majority of the respondents reported not having* had their accounts compromised, and either not being *aware of such compromise or cannot remember* its occurrence.

Research Question 2

Does age have impact on end-user experience in Password-based authentication?

To answer this question, a hypothesis was raised and tested.

Null Hypothesis(**H0**): Age of Internet users have no impact on their Password-based authentication experiences.

Hypothesis Testing

Cross tabulation of the independent variable (age) and dependent variable (Password-based authentication experience) showed observed and expected counts recorded on each category's option item and assumption check following Chi-Square tests is used to select that appropriate values required to ascertain the association strength between the variables and if the independent impacts on the dependent.

Table 4: Age * End-user experience in Password-based authentication (Has someone ever logged into your account without your permission?)

Age		Has someone logged into your account without your permission?				Total
		Yes	Not At All	Don't Know	Can't Remember	
<=20	Count Expected Count % Within Age	31 27.5 36.0%	40 45.2 46.5%	9 7.4 10.5%	6 5.9 7.0%	86 86.0 100.0%

```
    print(a);
}

int *arr = (int*) malloc(5 * sizeof(int));

/*
'x' is a global variable that will be allocated in memory statically.
'a' is a local variable with value 2 in a function that will be allocated by stack-based allocation.
'arr' is an array of 5, which will be allocated by heap-based allocation.
*/
```

Memory Management in Storage Allocation Strategies

Managing Memory while allocating and deallocating the data structures is a very important task for a computer. If the memory is managed efficiently, Issues like. memory leaks, fragmentation, and overallocation can be created.

There are two aspects of compilers that help in memory management, Garbage Collection and Fragmentation Management, and both of these are explained below.

Garbage Collection

In Garbage Collection, Compilers automatically handle the process of allocation and deallocation using the techniques such as reference counting or mark-and-sweep algorithms. In programming, garbage collection is done automatically on the variables when they are unused, unreferenced, or the variables with no values.

High-Level Programming Languages support garbage collection, such as Python, C#, and Java.

Advantages:

- Garbage Collection improves the efficiency of the program.
- The computer gets more memory to use.
- Dynamic Data Structures or variables are allowed.

Disadvantages:

- While executing the program, delays or pauses can happen.
- Garbage Collection is slower than stack-based allocation.

Here's an example of Garbage Collection in Java:

```
List<int>* list = new List<int>();
list->Add(5);
delete list;

/*
'list' is created using the 'new' keyword, which is empty initially, but we added one element 5 to
this 'list', and 'list' is deleted using the 'delete' keyword,          which is an example of garbag
e collection, where a memory will be deallocated automatically when 'list' is no longer needed.
*/
```

Demographic Data	Frequency	Percent	Cumulative Percent
Gender			
Male	157	54.0	54.0
Female	134	46.0	100.0
Age			
<=20	86	29.6	29.6
21-30	148	50.9	80.4
31-40	28	9.6	90.0
41-50	21	7.2	97.3
51-60	8	2.7	100.0
Educational Qualification			
School Certificate	187	64.3	64.3
OND (ND)	16	5.5	69.8
HND/B.Sc.	42	14.4	84.2
Ph.D.	46	15.8	100.0
Employment Status			
Students/Not Employed	243	83.5	83.5
Contract/Temporary Staff	3	1.0	84.5
Permanent Staff	39	13.4	97.9
Other	6	2.1	100.0

Chi-Square tests were used to gain insight into the association level that exists between the variables.

Assumptions: *for a 2 X 2 table, not more than 10% of cells have expected counts of less than 5, while for tables greater than 2 X 2, not more than 20% of cells should have expected counts of less than 5. If* the assumption is violated, the readings are taken from the Likelihood Ratio row *instead of the Pearson* Chi-Square row.

Research Question 1
What are the end-user experiences in password-based authentication?

To answer this question, descriptive analysis of respondents' self-reports on variables in this domain was carried out. Tables 2 and 3 below show the results of the analysis.

Table 2: Frequency Distribution of Respondents (Experienced unauthorized access to account?)

		Frequency	Percent	Cumulative Percent
Valid	Yes	93	32.0	32.0
	Not At All	153	52.6	84.5
	Don't Know	25	8.6	93.1
	Can't Remember	20	6.9	100.0
	Total	**291**	**100.0**	

From Table 2 above, 93 respondents, giving a distribution of 32% of total respondents, have experienced unauthorised access to their accounts, while a higher percentage of the respondents 153

Fragmentation Management

Fragmentation Management is one of the aspects that affect the performance of the program in terms of speed and efficiency. In fragmentation, the memory divides into smaller parts, and these sections can not be used. Issues like these can create when allocation and deallocation are done at run-time.

But If the data structures are allocated statically, problems like fragmentation will not arise.

We have some techniques by which we can reduce the fragmentation.

First Technique is to **move the allocated memory blocks** that will free up the space in memory. This free space can be used for the allocation and deallocation of the next data structures. Even though moving allocated memory blocks will help us to reduce the fragmentation, this approach is complex and time-consuming, which will ultimately affect the performance.

The second Technique can use the **memory pooling approach**, where a pre-allocated block of memory will be provided, and this block of memory can be used later for allocating new data structures. This approach will help us to reduce fragmentation and will not affect the performance of the program.

Trade-Offs Between Storage Allocation Strategies

There are several trade-offs to consider while choosing a strategy in compiler design. **Static Allocation** provides fast access to memory and efficiency, but the memory can be wasted. At the same time, **Dynamic Allocation** provides flexibility to the data structures but is slower than static allocation and can be less efficient.

Stack-Based Allocation is fast and efficient, but the size of the memory (stack) can be limited, whereas **Heap-Based Allocation** provides flexibility to the data structures but can be slower than stack-based allocation and less efficient.

Hybrid Allocation is an allocation that takes the benefits of multiple allocations, but it's more complex and difficult to debug.

Challenges in Storage Allocation Strategies

Several challenges in storage allocation strategies can affect the performance of the program. There are two significant challenges are discussed below:

1. **Overhead:** It is a limitation that can arise where processing time and memory usage required to manage allocations and deallocations are extra. Depending on the allocation strategies, they have different levels of overhead. The stack-based allocation has a low overhead. The heap-based allocation has a higher overhead. Overhead mostly occurs during the execution when the memory is to be allocated at run-time.

2. **Memory Usage:** This is also one of the challenges encountered when the allocation and deallocation are done with the wrong amount of memory blocks to the data structures. Depending on the allocation strategies, different strategies have different limits on the amount of memory. As the size of the stack is limited, the stack-based allocation has a limit very limited.

If the right storage allocation strategies are chosen, the issues such as memory leaks, fragmentation, and overallocation can be reduced, and the performance of the program can be increased.

Sekaran & Bougies, (2010), using sample relative to the entire population was considered as being able to produce reliable and better result and remove unnecessary stress. A systematic sampling approach was adopted to provide for a heterogenous population characteristics (Leedy & Ormrod, 2005) and to make generalisation to the population (Bryman & Bello, 2003). Systematic sampling technique was considered suitable by previous studies such as (Albuelu & Ogbouma, 2013; Asgharnezhad, Akbarlou & Karlcaj, 20 13; Aliyu, 2013). At the end of the survey, a sample size of 291 was used for the study after data cleaning, which represented the actual number as suggested by Krejcie & Morgan (1970).

2.3 Data Collection Method

The study used online-administered survey approach. The questionnaire was designed in Google form and distributed to our respondents through WhatsApp and email addresses. The primary targets of respondents were students and employees of the Universities since they were perceived likely to be the most informed users of online Password-based authentication system. Prior to the distribution of the questionnaire, an ethical approval was obtained for the study from the approved authority. The responses to the questionnaire were collected through the spreadsheet designed for this purpose. A total of 299 responses were received out of the target population of 1200 and were subjected to data cleaning and normalisation. The extracted and normalised data were subsequently subjected to analysis.

3.0 Analysis, Results and Discussion

3.1 Data Analysis

Data from respondents were coded into numerical data in a spreadsheet. Processing and analysis of captured data was carried out using SPSS (Statistical Package for Social Sciences) version 20.0 and reported by means of descriptive statistics, frequency distribution, and Chi-Square tests.

3.2 Results

The demographic data is presented in Table 1 below. The table shows that 157 male respondents (54%) and 134 female respondents (46%) participated in the survey. Ages of the participants were divided into five categories. Participants that are 20 years and below has a distribution of 86 out of 291 participants, which represents 29.6% of all participants. The second category captured participants that are between 21-30. This group has a distribution of 148(50.9%) of 291 respondents. The third category captured respondents between 31-40 years and had a distribution of 28 representing 9.6% of all participants. In category 4, respondents between 41-50 were captured with a distribution of 21, representing 7.2% of entire respondents. The last category captured respondents within the ages of 51-60. A distribution of 8 was observed, representing 2.7% of the entire participants. The third demographic data collected is the highest education qualification obtained by the respondents. The first category captured are respondents with secondary school certificate. 187 respondents out of the 291, representing 64.3% of participants belong to this category. The second category had OND (Ordinary National Diploma) certificate owners covered. This category had a distribution of 16 respondents, which is 5.5% of the entire 291 respondents. Other categories are HND (Higher National Diploma and B.Sc. (Bachelor of Science) with distributions of 42 (14.4%) and those with Ph.D. (Doctor of Philosophy) had a distribution of 46 (15.8%) respectively. Demographic data on the employment status of respondents showed that students/unemployed respondents had a distribution of 243 out of 291 participants, giving 83.5% of all participants. Contract/temporary staff had a distribution of 3 (1.0%), while permanent staff and undefined categories had a distribution of 39 (13.4%) and 6 (2.1%) respectively.

Table 1: Demographic Information

Frequently Asked Questions

What is stack-based allocation?

In Stack-Based Allocation, A fixed-sized stack data structure is allocated to a memory location during the run-time execution. A variable is assigned a memory location on the stack when they are declared and can be freed when the variable's scope goes out.

What is heap-based allocation?

In Heap-Based Allocation, Variables that are created and memory will be dynamically allocated in memory (heap) at run-time execution. The variables that use heap-based allocation are generally known as dynamic variables that can be changed anytime during the run-time.

Why heap-based Allocation is slower than stack-based allocation?

In Heap-Based Allocation, the data structure is created dynamically so that it is allocated in memory at the run-time, which means the data of the data structure can be changed anytime. This means more time will be taken to update the data in a memory location, which slows the execution time in heap-based allocation.

Why Storage Allocation is important in Compiler Design?

By doing storage allocation, we can determine how a program utilizes memory (if it's efficient or inefficient); assigning memory locations in a computer's memory is one of the important tasks that is performed by compilers and operating systems.

Conclusion

Storage Allocation is a process of assigning memory addresses to different data structures, such as variables, arrays, and objects. This article was all about the "Storage Allocation Strategies in Compiler Design", where we discussed what storage allocation is and several strategies of storage allocation that are used by the compiler automatically with their examples. We also discussed memory management and the challenges in these allocation strategies.

study, the chosen external variables will represent unique characteristics of the sample that could have impact on the online end-users using a password-based authentication system (Venkatesh *et al.*, 2003).

1.6.2 Online End-User Behaviour

The external variables influence the end-user's behaviour construct. From the research, the experiences that the consumer displays due to external variables could give a user a more positive/negative attitude towards using the password-based authentication system (Venkatesh *et al.*, 2012). According to Brock and Khan (2017) behavioural intention to use is an essential step towards the actual usage of any new system. In this study, online end-users' behaviour will be used to evaluate online consumers' experiences while using a password-based authentication system.

1.6.3 Actual Use

Actual use of a system/technology is determined by users behaviour (Venkatesh *et al.*, 2012). In this study, actual use indicates the end-user's usage of a password-based authentication system while transacting online. The end-user experiences will determine the predictions of the usage. According to Butler and Butler (2015), many computer security password breaches result from poor user security behaviour. The password creation and management practices that online consumers apply have a direct effect on the level of computer security and are often targeted in attacks.

2.0 Research Methodology

This study was conducted using some Universities within the southeastern region of the country with *ethical approval obtained* from one of the concerned authorities. The study used online-administered *survey approach and the questionnaire* was designed in Google form and distributed to our respondents *through WhatsApp and email addresses.* Data from respondents were coded into numerical data in a *spreadsheet. Processing* and analysis of captured data was carried out using SPSS (Statistical Package *for Social Sciences) version 20.0* and reported by means of descriptive statistics, frequency distribution, *and Chi-Square tests.* *In all,* about 291 responses were used after data cleaning in accordance to the required sample size of the respondent population (Krejcie & Morgan (1970)).

2.1 Instruments and Methods

For the researchers to verify the hypotheses formulated, quantitative research methodology was adopted in the study which according to Adegbuyi *et al.*, (2015) is the most suitable for any exploratory investigation for better understating of a particular problem under scrutiny. The variables needed to measure the main constructs that are part of the conceptual framework were identified through extensive review of literature. A cross-sectional research design in conjunction with a well-structured questionnaire with closed-ended questions as used by Bowen *et al.*, (2010) was adopted to gather information from the respondents. The questionnaire is comprised of two sections. Section A has questions that focused on obtaining the demographics of the respondents such as age, gender, educational qualification, marital status, and employment information while section B examined the online end-user experiences with the Password-based authentication systems. The questions in section B were designed in such a way that data can be collected based on the respondents' experiences with Password-based authentication systems.

2.2 Sample Size and Sampling Technique

A population of 1200 possible respondents were drawn from Universities in the Southeastern Nigeria for the purpose of this study with an expected sample size of 291 as stipulated in Krejcie & Morgan (1970). Since the possibility of collecting data from the entire population is not possible according to