1-7-2022

Hands-on introductory training in Backdoor and SQL injection attacks

Anil Singh
The University of Texas Rio Grande Valley

Sandra Henderson
University of Texas at Arlington

Recommended Citation

Singh, A., & Henderson, S. (2022). Hands-on introductory training in Backdoor and SQL injection attacks. The Accounting Educators' Journal, 31(1). Retrieved from https://aejournal.com/ojs/index.php/aej/article/view/629

THE ACCOUNTING EDUCATORS' JOURNAL
Volume XXXI
2021
pp. 49-62

Hands-on Introductory Training in Backdoor and SQL Injection Attacks

Anil Singh
University of Texas Rio Grande Valley

Sandra Cherie Henderson
University of Texas at Arlington

Abstract

Software, though vital to organizations, come with risks attached. Malicious use of software has caused a great deal of damage to individuals, companies, and even countries. Accounting students generally do not acquire detailed understanding of such threats. To introduce accounting students to software vulnerabilities, we provide two MS Access exercises illustrating simplified versions of Backdoor attacks (BDA) and SQL injection attacks (SQLIA). These quick and simple-to-do introductory exercises help accounting students get a closer look at software vulnerabilities. The COSO framework, to address such risks is discussed. Post exercise survey revealed increased understanding of software vulnerabilities.

Introduction

Ubiquitous use of computers has increased IT related risks. Consequently, governing and auditing information technology is required. Apart from looking for fraud within the organization, auditors need to consider vulnerabilities in other areas - such as in Information Technology - that are not within their domain. While accounting professionals are expected have mastery over the accounting domain, professions like auditing require some breadth of knowledge in Information Technology too. Extant research in accounting reveals an emphasis on bridging such gaps between accounting and IT (Debreceny 2013; Murthy 2016).

Learning Objectives

Learning objectives of this teaching resource are

1) Hands-on introductory experience in intricacies of Backdoor attacks (BDA) and SQL injection attacks (SQLIA).
2) Obtain greater understanding of software vulnerabilities and the need to audit software.
3) Understand roles COSO framework and the Sarbanes–Oxley Act play in addressing such vulnerabilities.

Information Technology Risk

Information Technology has become pervasive with heavy usage in both e-business (internal) and e-commerce (external). IT can be a great enabler. Yet, it often becomes a source of risk, leading to catastrophic consequences. This makes software related risks a significant part of the auditor's focus. Accountants are generally not accustomed to the highly technical IT domain (O'Donnell and Rechtman 2005). Hence, a simple exercise to understand the system is vital.

Microsoft Access

Microsoft Access has been used to teach accounting concepts (Borthick and Bowen 2008; Henderson, Lapke, and Garcia 2016). It is readily available, simple to use, and accounting professionals are more likely to have used it. Learners with basic knowledge in Microsoft Access – such as creating a database, tables, and forms - can use this easy-to-do exercise to build a simple database and create examples of SQLIA and BDA.

Back Door Attacks (BDA)

Exercise

Backdoor attacks are associated with hidden access in code. This exercise, without delving too much into the technicalities of programming, provides learners with a firsthand account of backdoors attacks. We develop hidden access in code that bypasses usual methods.

1) Create a new database named COMPANY in Microsoft Access.
2) Create a table named EMPLOYEE in COMPANY database.
3) In the design view of the table, create fields with the following details:
   a. Primary Key Field: UserID; Datatype: short text (leave the rest as default)
   b. Field Password, Datatype: short text; input mask: password (leave the rest as default)
4) Populate the table - in the datasheet view - with the three User IDs and passwords as shown in Table 1. Figure 1 provides a screenshot.
5) Create a Form named Login (Figure 2) with the following controls:
   a) Textbox - Name: UserID (accompanying label - UserID)
   b) Textbox - Name: Password; Input Mask: password (accompanying label - Password)
   c) Button - Name: Login; Caption: Login
6) Create a Form named Welcome (Figure 3) with the following controls:
   a) Label with the following content:” Welcome! You have now entered the system”.
7) Go to design view of the Login form, click on the code of the Login button and click on view code (Figure 4)
8) In the code for the login button (in Form Login), type the following case-sensitive code:

```
Private Sub Login_Click()

If Me.Password.Value = DLookup("Password", "Employee", "UserID = '" &

Me.UserID.Value & "'") Then

DoCmd.OpenForm "Welcome"

Else

MsgBox ("Login not successful")

End If

End Sub
```

9) Go to form view and use the Login form to test the code. If User ID and password are correct, the Welcome form (Figure 3) will open. Close the welcome form and test the code with random incorrect User IDs and passwords. This will open a message: “Login not successful”.

10) Return to the code and add the following code between Else and MsgBox ("Login not successful"):

```
If Me.UserID.Value = "hacker" And Me.Password.Value = "hacker123" Then

DoCmd.OpenForm "Welcome"

Exit sub

End If
```

Compiling the Database

To convert the database to an executable file (compile) go to File - > Save As -> Save Database As - > Make ACCDE (Figure 5).

Test the software

Close the database. You will now see two database files in the folder. In icon view, the icon with the lock symbol in the top right corner is the executable file. It has .accde extension. Once you run the executable file, you will observe that the database opens but some developer tools/design are grayed out and you don't have access to the code. (Note: This is the case with closed source software). Test the code with the available User IDs and Passwords in Table 1. Correct User IDs and passwords will open the Welcome form (Figure 3). Close the welcome form. Now test the code with random incorrect User IDs and passwords. This will open the message: "Login not successful". Now use the User ID and passwords hidden in code (UserID: hacker and Password: hacker123). This will open the Welcome form (Figure 3).

Lessons learned

Userids and passwords hidden in code are inaccessible to the user. Generally, users get access to the executable file and not the code. The code hides the UserID (hacker) and Password (hacker123), thus allowing 'Backdoor' access to the software. Such vulnerabilities can only be identified when source code is available.

Backdoor Attack Details

Backdoors are User IDs, passwords or other data in the source code that are - for practical or malicious purposes - used to bypass regular security protocols. They can result from a programmer manipulating the original code or an intruder with access to change the code (Landwehr, Bull, McDermott, and Choi 1994). Not all Backdoors are malicious. There are non-malicious practical applications of Backdoors too. In non-malicious cases, the intention to maintain Backdoors is functional, whereas in malicious cases, the purpose is to use available Backdoors or create a Backdoor in the code (if one has access to it) for malicious purposes. It is a tradeoff between security and practicality.

Benefits of Backdoors

It may be surprising to know that hard coded passwords in many cases were intentional but non-malicious (Landwehr, Bull, McDermott, and Choi 1994). Backdoors created with good intent - often called features, golden keys, front doors, default passwords, static passwords or admin passwords, were intended to serve a practical purpose. With Backdoor-free software, if admin passwords were lost, there would be no way to perform administrative operations on the software and in certain situations all data may be lost. There is a risk of the software becoming useless. To the programmer, Backdoors provide more control. Logic-bombs, a type of Backdoor even allows vendors to disable software after a point in time or if payments for software do not happen (Roditti 1995). Backdoors can help in policing too. Encrypted systems, while good for law-abiding citizens may – without a Backdoor- be a powerful weapon in the hands of terrorists (Corn 2015). Communications between them are so well encrypted that surveillance – without the help of Backdoors – is not possible. Many times, unintentional non-malicious developers intend to help others solve glitches. Anticipating such situations, software developers created Backdoors. Often admin IDs and passwords are available in forums on the internet where developers find solutions, troubleshoot, meet to discuss, and solve software related problems.

Risks of Backdoors
Backdoors are good if their purpose is to facilitate. It is, however, obvious how such functional Backdoors can be misused. While Backdoors are useful if admin password is lost, they have great potential for damage. Such practical Backdoors have increasingly come under criticism for obvious reasons, as they can be manipulated or misused for malicious intent (Ngo 1999). Increasingly, such facilitating Backdoors are misused or new Backdoors created for malicious intent. Instances of misuse were found in VAX computers (Opaska 1986), MySQL, PHPMydmin, JBoss, Sercomm DSL, Linksys and Netgear (Pewny, Garmany, Gawlik, Rossow, and Holz 2015). Backdoor risks can be divided into risks from misusing functional Backdoors and risks of hidden code with malicious intent. Backdoors can lead to ransomware attacks. Table 2 provides a representative list of such attacks, gathered from news database Factiva.

SQL Injection Attacks (SQLIA)

Exercise
SQL injection attacks are associated with exploiting weaknesses in code. This exercise, without delving too much into the technicalities of programming, provides learners with a firsthand account of SQL injection attacks. We continue where we left off in the BDA exercise.
In the database file (not the ACCDE file), go to the login button (in Form Login) code from the previous example and replace the entire code with the following code:

```
Option Compare Database

Private Sub Login_Click()

Dim rs As Recordset

Dim str As String

Set db = CurrentDb

Set rs = db.OpenRecordset("Select * from employee where UserID = '" &

Me.UserID.Value & "' and Password = '" & Me.Password.Value & "'")

If rs.RecordCount = 0 Then

MsgBox "Login not successful"

Else

DoCmd.OpenForm "Welcome"

End If

End Sub
```

Test the software

Save the database again as an ACCDE file (Figure 5). Test the code with the available User IDs and Passwords in Table 1. Correct User IDs and passwords will open the Welcome form (Figure 3). Close the welcome form and test the code with random incorrect User IDs and passwords. This will open a message: "Login not successful". Now copy-paste: 'or "=' (include single quotes) in the User ID and password textboxes. Welcome form will open. Often the User ID is known and so only the password needs to be populated with: 'or "='. You can also use: ' or ' 1=1 (from first single quote to last 1) for User ID and password.

Lessons learned

Bad quality code creates vulnerabilities that can be exploited by hackers. Such vulnerabilities can be exploited at the user input level.

SQLIA Details

Structured Query Language (SQL) is the most widely used language to manage databases. It joins the users to the database providing access to data and embeds in internet friendly languages like Java and ASP. SQL commands are often inaccessible to users. The user's only contribution to the command is the input. When a user inputs data such as User ID and Password, that data - called parameter - becomes part of the SQL command. This is where SQL's vulnerability lies. Inputs can be manipulated or even masquerade as commands to illegally access and destroy. Such manipulations are called SQL Injection Attacks (SQLIA). In the exercise, we used: ' or ' 1=1 meaning true in SQL. Similarly, an SQL injection with ' or "=' too is always true. All rows in the employee table return a true, thereby allowing access.

Various types SQL injections take advantage of the vulnerabilities in SQL- some to gain illegal entry, some to destroy (Halfond, Viegas, and Orso 2006). The above-mentioned exercise, for example, can be tweaked to delete entire tables. Unlike BDAs, SQLIAs are not hidden code. They are manipulations at the user input level (front-end). While BDAs can only be from someone who has developed or has access to change the code, SQL injection intruder could be anybody who has access to the front-end. Such attacks target databases that are accessible through a web front-end and take advantage of flaws in the input validation logic (Boyd and Keromytis 2004). SQL injection attacks are among the biggest threats for applications written for the Web (Halfond, Viegas, and Orso 2006). Table 3 provides a representative list of SQLIAs gathered from news database Factiva. Good code to validate inputs will prevent such attacks. Good code and methods to prevent such attacks are beyond the scope of this training.

Software Risks

Malicious use of hidden code manifests in many forms including backdoor attacks, viruses, worms, spyware, adware, logic bombs and trojan horses. Hidden commands include creating duplicates, destruction of files, spying, deploying unwanted advertisements, taking the system hostage by encrypting etc. Trojan horses, for example, use hidden code to trigger distributed denial of service attacks. Weak codes are unintentional and careless mistakes that create errors and vulnerabilities that open the software to attacks. Computer security that will not address such vulnerabilities can have a widespread impact on businesses and lose money (Ngo 1999).
Both hidden and weak codes are a threat as they compromise internal controls of the organization. They allow intruders to illegally access information and even destroy systems. As is evident from the two exercises, some attacks don't need sophistication and can be used with basic knowledge. Typical methods to find malicious code or detect bad quality code require access to software code and do not work with executable files (Pewny, Garmany, Gawlik, Rossow, and Holz 2015). Therefore, efforts are being made to make software source accessible (Evans and Reddy 2002; De 2009). Table 4 provides a representative list of vulnerabilities in software.

Managing Software with COSO

The Sarbanes-Oxley Act of 2002 (SOX) requires companies to choose and implement an established IT security framework (Wallace, Lin, and Cefaratti 2011). One such framework is the Committee of Sponsoring Organizations of the Treadway Commission (COSO) framework. IT security, being at the crossroads of IT and management, requires a combination of both management and technical policies. To this end, the COSO framework provides an outline for IT governance. In Table 5, we have mapped components of COSO to candidate policies in managing software.

Conclusion

Many organizations had the displeasure of experiencing IT attacks in one form or the other. This training helps students understand two vulnerabilities in software code: malicious hidden code and bad quality code. Backdoor Attacks (BDAs) use malicious hidden code while SQL Injection Attacks (SQLIAs) target bad quality code. The accounting curriculum typically does not include specifics of software vulnerabilities. This teaching resource - without getting into the complexities of programming - provides hands-on introductory experience on the two software vulnerabilities. Such vulnerabilities justify the need to manage and monitor software through established risk management frameworks such as the Committee of Sponsoring Organizations of the Treadway Commission (COSO) framework.

References

Borthick, A.F., and P. L. Bowen. 2008. Auditing System Development: Constructing the Meaning Of "Systematic and Rational" In the Context of Legacy Code Migration for Vendor Incentives. *Journal of Information Systems* 22(1): 47-62.

Boyd, S., and A. Keromytis. 2004. SQLrand: Preventing SQL Injection Attacks. In Applied Cryptography and Network Security 292-302, Springer Berlin/Heidelberg.

Corn, G.S. 2015. Averting the Inherent Dangers of "Going Dark": Why Congress Must Require a Locked Front Door to Encrypted Data. *Washington and Lee Law Review* 72(3): 1433.

COSO. 1992. Internal Control—Integrated Framework. *The Committee of Sponsoring Organizations of the Treadway Commission.*

De, R. 2009. Economic Impact of Free and Open Source Software: A Study in India. *Interop, Mumbai, October*: 7-9.

Debreceny, R.S. 2013. Research on IT Governance, Risk, and Value: Challenges and Opportunities. *Journal of Information Systems* 27(1): 129-35.

Evans, D.S., and B. J. Reddy. 2002. Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem. *Michigan Telecommunications and Technology Law Review* 9: 313-457.

Halfond, W.G., J. Viegas, and A. Orso. 2006. A Classification of SQL-Injection Attacks and Countermeasures. Paper presented at Proceedings of the IEEE International Symposium on Secure Software Engineering,

Henderson, D., M. Lapke, and C. Garcia. 2016. SQL Injection: A Demonstration and Implications for Accounting Students. *AIS Educator Journal* 11(1): 1-8.

Landwehr, C., A. Bull, J. Mcdermott, and W. Choi. 1994. A Taxonomy of Computer-Program Security Flaws. *ACM Computing Surveys* 26(3): 211-54.

Murthy, U.S. 2016. Researching at the Intersection of Accounting and Information Technology: A Call for Action. *Journal of Information Systems* 30(2): 159-67.

Ngo, H.H. 1999. Corporate System Security: Towards an Integrated Management Approach. *Information Management & Computer Security* 7(5): 217-22.

O'Donnell, J.B., and Y. Rechtman. 2005. Navigating the Standards for Information Technology Controls. *The CPA Journal* 75(7): 64.

Opaska, W.P. 1986. Closing the VAX default password "Backdoor". *EDPACS: The EDP Audit, Control, and Security Newsletter* 14(3): 6-9.

Pewny, J., B. Garmany, R. Gawlik, C. Rossow, and T. Holz. 2015. Cross-Architecture Bug Search in Binary Executables. *IEEE Symposium on Security and Privacy.*

Roditti, E.C. 1995. Is Self-Help a Lawful Contractual Remedy? *Rutgers Computer & Technology Law Journal* 21(2): 431.

Wallace, L., H. Lin, and M. A. Cefaratti. 2011. Information Security and Sarbanes-Oxley Compliance: An Exploratory Study. *Journal of Information Systems* 25(1): 185-211.

APPENDIX A- Assessment of Efficacy

After debriefing, students were asked to formally respond to the following questions using a Likert scale of 1 (strongly disagree) to 5 (strongly agree). Results (Table A) show that the learning objectives were met and students found the training to be of value.

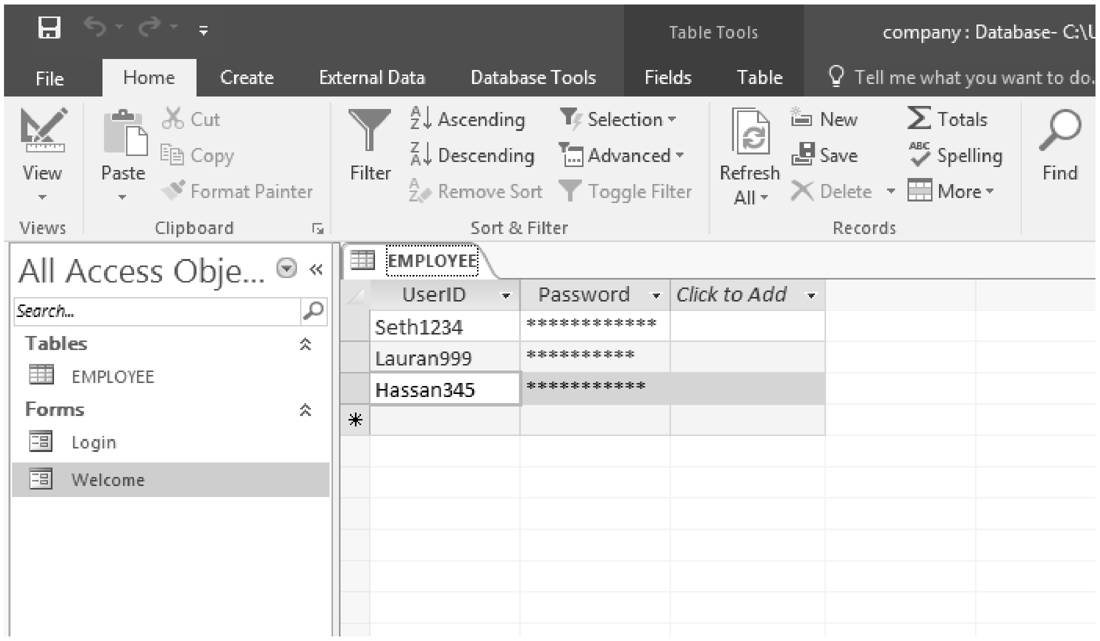
Table A. Assessment of Efficacy (N=155)

Item	Average	Std. Dev	Min	Max
This task gave me a better understanding of backdoor attacks.	4.115	0.894	1	5
This task gave me a better understanding of SQL injection attacks.	4.012	0.912	1	5
This task gave me a better understanding of importance of vulnerabilities in software	4.320	0.894	1	5
This project gave me a better understanding of the need to audit software	4.122	0.940	1	5

Table 1. User IDs and Passwords

UserID	Password
Seth1234	superman*123
Lauran999	Batgirl333
Hassan345	Cooldude222

Figure 1. User ID and Password Data



EMPLOYEE

UserID	Password	Click to Add
Seth1234	************	
Lauran999	**********	
Hassan345	***********	

Figure 2. Login Form

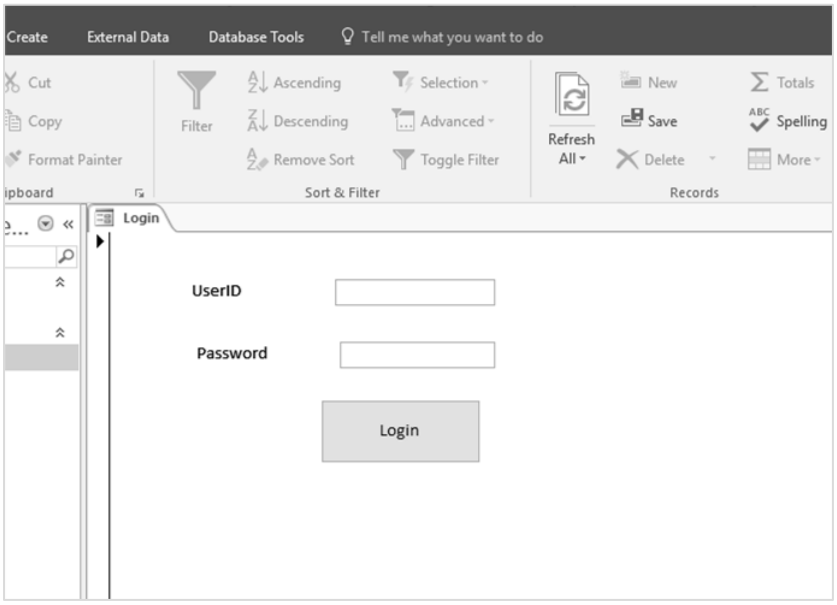
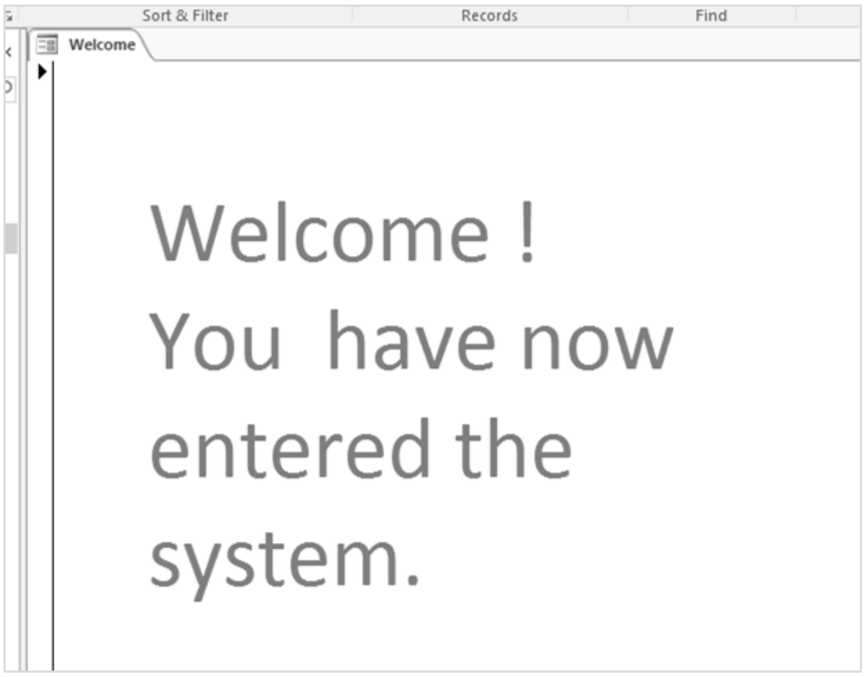
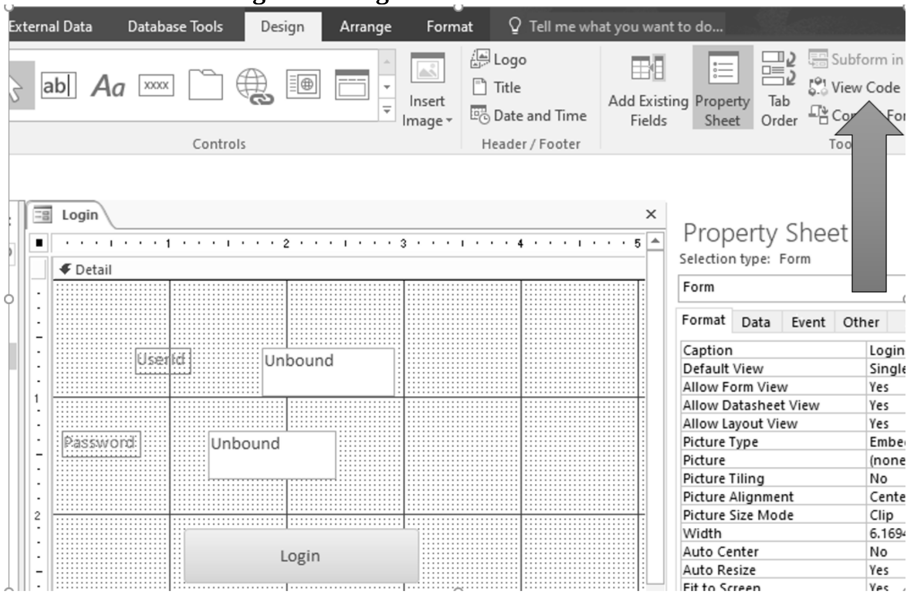


Figure 3. Welcome Form



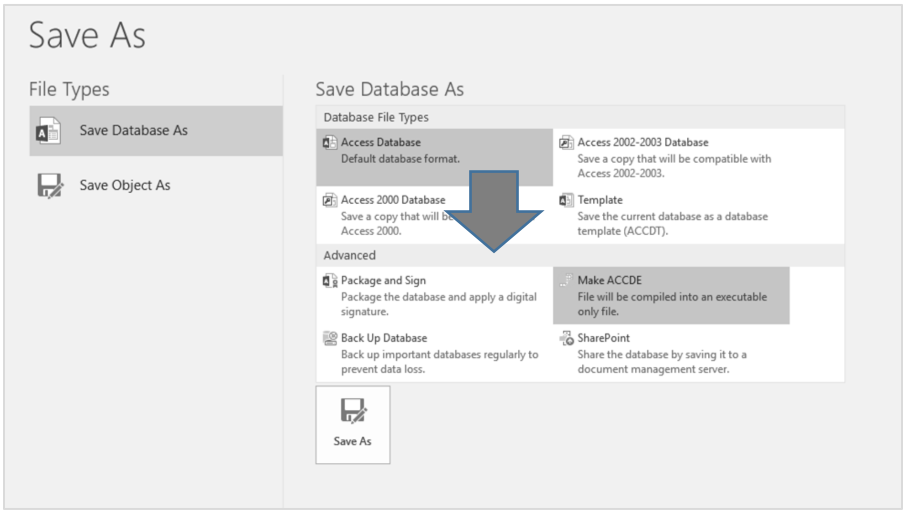
Welcome !
You have now
entered the
system.

Figure 4. Login Form view code button



Caption	Login
Default View	Single
Allow Form View	Yes
Allow Datasheet View	Yes
Allow Layout View	Yes
Picture Type	Embe
Picture	(none
Picture Tiling	No
Picture Alignment	Cente
Picture Size Mode	Clip
Width	6.169
Auto Center	No
Auto Resize	Yes
Fit to Screen	Yes

Figure 5. Save database as an executable file



Save As

File Types

Save Database As

Save Object As

Save Database As

Database File Types

Access Database
Default database format.

Access 2002-2003 Database
Save a copy that will be compatible with Access 2002-2003.

Access 2000 Database
Save a copy that will be
Access 2000.

Template
Save the current database as a database template (ACCDT).

Advanced

Package and Sign
Package the database and apply a digital signature.

Make ACCDE
File will be compiled into an executable only file.

Back Up Database
Back up important databases regularly to prevent data loss.

SharePoint
Share the database by saving it to a document management server.

Save As

Table 2. Representative list of Backdoors (Source: Factiva)

Date	Source	Article Title
6/28/2017	CNN	Ransomware attack hits global companies.
5/13/2017	FARS News Agency	74 Countries hit by NSA-powered WannaCrypt Ransomware Backdoor.
2/8/2016	SC Magazine	Skype targeted by T9000 Backdoor Trojan.
11/26/2014	SC Magazine	Pirated Joomla, WordPress, Drupal themes and plugins contain CryptoPHP Backdoor.
10/202014	SC Magazine,	Modular malware for OS X includes Backdoor, keylogger components.

Table 3. Representative list of SQL injection attacks (Source: Factiva)

Date	Source	Details
7/5/2017	Cyber Defense Magazine	SQL Injection flaw in WordPress Plugin WP Statistics potentially exposed 300,000+ Sites
6/27/2017	SC Magazine	One of the most prevalent cyber-attack methods observed are SQL Injections.
5/19/2017	Cyber Defense Magazine	Critical SQL Injection vulnerability found and patched in Joomla.
9/13/2016	Open Source for You	Newest MySQL vulnerability lets attackers inject malicious settings
8/12/2015	The New York Times	Mr. Turchynov used SQL injections were used to on Marketwired on at least 390 occasions.
10/16/2014	SC Magazine	Drupal core contains 'highly critical' SQL injection vulnerability
7/23/2014	SC Magazine	Wall Street Journal website vulnerable to SQL injection, gets hacked

Table 4. Representative list of vulnerabilities

Vulnerability	Details
Adware	Code that renders unwanted advertising material.
Backdoor	Hidden code to bypass security protocols
SQL Injection	Weak code result from the mixing of code allowing access to intruders
Buffer overflow	Vulnerability that allows writing data beyond buffer boundary
Virus	Malicious code
Worms	Self-propagating malicious code in operating systems.
Trojan horse	Malicious programs that pretend to be useful.
Logic-bombs	May trigger based on a time or event.
Bug	Flaw in the program
Zero-day attacks	Involve exploiting vulnerabilities before they are patched-up.

Table 5. Mapping components of COSO to software code management

Component of COSO	**Brief Definition (source COSO, 1992)**	**Software Code Management**
Control environment	"*set of standards, processes, and structures that provide the basis for carrying out internal control across the organization"*	- Attract, develop, and retain competent individuals. - Maintain competent IT partnerships. - Insist on access to software Code.
Risk assessment	*"dynamic and iterative process for identifying and assessing risks to the achievement of objectives"*	- Programmers with access to the code could exploit the code. - Intruders (external) can exploit vulnerabilities. - Potential violations of usage license.
Control activities	*"actions…… ensure that……objectives are carried out."*	- Restrict access to program code. - Check software for authenticity. - Restrict employees from downloading software. - Scan for illegally downloaded codes. - Scan for and update latest patches as soon as they become available. - Restrict access to admin passwords. - Identify vulnerabilities in code. - Use code management software. - Periodically reconcile code. - Change default root passwords as soon as possible.
Information and communication	*"information from both internal and external sources ….providing, sharing, and obtaining necessary information."*	-Train employees on legal issues of software usage. -Train employees on reporting bugs. -Scan internet for vulnerabilities and patches. -Monitor changes to program code. -Identifying known malicious patterns.
Monitoring	*"Ongoing evaluations… to ascertain … five components of internal control……present and functioning."*	-Review above mentioned internal control components.