Enhanced Analysis Method for Suspicious PDF Files

Suleiman J. Khitan, Ali Hadi, Jalal Atoum

Computer Science Dept., Princess Sumaya University for Technology, Amman, Jordan

Summary
This paper presents an enhanced method for analyzing suspicious PDF files. Since recently these files are considered as common, reliable and secure documents used by attackers as a container to attack users. Attackers have shifted their methods from server-side to client-side attacks. The attackers used them to carry out malicious code on the computer systems of the users. This attack makes a threat to the institution's asset that could be exploited. The enhanced method is based on scan the PDF file structure according to predefined set of keywords together with the new defined keywords. Also define the vulnerabilities and the most common techniques the attackers use to be protected from discovery. The new defined keywords are identified as objects, have been used by attackers, recently embedded in the PDF files. The enhanced method identifies malicious PDF documents by searching for embedded objects that are considered as suspicious keywords in the documents. The importance of this paper lies on develop a method to detect suspicious PDF files which depends on extracting and pointing out malicious objects that are often used for attacks. This enhanced method will be of great importance to users who deal with threat every day.

Key words:
Malware analysis, PDF documents, Malicious PDF, Suspicious PDF, Structure Scan.

1. Introduction

The feature of PDF file, especially the dynamic content may lead to several security issues that can be used to hold malicious elements to install malware and steal data, these features may contain code written in JavaScript. This will allow the attacking persons to insert advanced features as multimedia files, to connect with outside sites. Unfortunately the attacker can use the features provided by JavaScript to exploit the vulnerabilities in the PDF viewer applications. By using JavaScript the attacker can be able to do two things: trigger the vulnerable code and then point the execution to arbitrary code of his choice to gain privileges of the user to run or stop the application, deny service to legitimate user as heap-spraying [1], or other memory manipulation techniques.

Attackers have shifted their methods from server-side to client-side attacks which take advantages of social engineering, non-technical techniques and applications that are not up-to-date where the goal is to deceive the less awareness users into opening PDF file content using applications found on most personal computers [2].

One group of client applications is PDF documents that have become the most common in exchanging the documents. PDF documents are used in many sectors like business proposals, product manuals and legal documents. This is referred to because of the advantages it offers as a portable document, visually appealing and interactive, as well it contains text and images in addition to the ability to embed JavaScript, Flash and to open external sites locally from the computer, or the internet [3] which referred as static and dynamic contents.

In addition to the vulnerabilities on the PDF viewer, the attackers also took the advantages of the advanced PDF features as /Launch option which execute an embedded script automatically, or the/URI and /GoTo options which can open external resources from the same computer [4].

Client-side attack focuses on the weakness in applications and resumes rising in away faster than the server-side attack, which is used in online crime like identity theft and creation botnet.

2. Background

PDF is a file format developed by Adobe in 1993 and released as an open standard by the International Organization for Standardization in 2008 as ISO 32000-1 [5] to enable individuals to easily transfer electronic documents in trusted ways without depending on a specific platform.

Regarding to the PDF specification, PDF file contains mainly four sections: the header, the main body, the cross-reference table and the trailer [5].

The Header: It is a single line every PDF file must have at somewhere within the first 1024 bytes of the file, called magic number that specifies the version of PDF programming language.

The body: Contains a list of objects (data, text streams, images, fonts, etc.) that form the most part of the PDF file. It will be discussed in more details in next section.

Cross-reference table (xref table): It's a table that works as a pointer for each object in the PDF file to determine its location, so the PDF viewer application can identify the position of an object randomly without needing to scan the whole file to find that object. The position is identified as a byte offset which is the number of bytes from the beginning of the file to the beginning of the object.

Manuscript received May 5, 2015
Manuscript revised May 20, 2015

Trailer: Identify the location of the of the cross-reference table (byte position) besides to some certain objects like root object, as the PDF render check at first the version number in the header to identify it as a PDF file and use the trailer to find the cross-reference table and some objects such as catalog object. The last line of the file contains a label %%EOF that identifies the end of the document.

The basic format of PDF is made up of objects as a type of data. There exist nine different types of objects:

Boolean Object: Can be represented by the keywords "true" or "false".

Numeric Object: PDF uses numbers as integers and real's.

String Object: It is set of bytes used to represent text data, which is either as "literal" characters enclosed within parentheses or as "hexadecimal" characters enclosed within angle brackets.

Name Object: It is defined by a slash (/) followed by a set of strings in which the slash is not part of the name but it works as an identifier to this object.

Array Object: It is a one dimensional array of any other types of objects in addition to other arrays enclosed between square brackets ordered one after another.

Dictionary Object: It's the main component of a PDF document which is a key-value pairs. The key is always a name object and the value is any other objects in addition to another dictionary. Dictionary object in enclosed within double brackets (<< … >>).

Stream Object: It is a set of bytes like string object but the only thing that they differ from each other that stream object is not limited so it can be used to hold large data like images. The general form for the stream object is that it begins with a dictionary object that indicate the size of the stream followed by the data of the stream which is placed between the two keywords 'stream' and 'endstream'.

The Null Object: As the name implies there is no existence object, or having no value in a dictionary.

Indirect Object: It is possible to point to an object from the other location in the file by other one and this is can be done by labeling the object with an identifier that composed of the object number which should be unique and a generation number.

3. Literature Review

There are number of studies that are related to detecting malicious PDF files. one of the related work depend on JavaScript code within PDF file [6]. They present the tool PJScan which is able to detect malicious PDF with JavaScript related malware which relies on extraction the JavaScript from the malicious files to obtain lexical properties of the Script by the tokenizer. The output which is the token sequence is fed as input to the learning algorithm machine; this one is learned on known malicious PDF file to produce a model which is used for classification of unknown malicious files. In the second stage every unknown malicious PDF file will pass in the same stages from extracting of JavaScript, tokenize it and apply the token sequence to the learning algorithm in which the detector compare this output with a learned model to measure the deflection from a predefined threshold so values that are close to a learned model are considered as malicious and else they are benign. Other work depends on static and dynamic analysis like Wepawet which is a web-based service implements a dynamic analysis for malware of PDF documents depends on JavaScript contained within it [7]- [8], which utilize JSAND to detect malicious JavaScript code based on lexical analysis.

Uploading a PDF file for analysis gives a report which contains details about the files that flagged as malicious like MD5 of the file, exploits, detection results if the file is malicious or suspicious or benign, malwares and shellcode. The detection results are identified based on the usage of well-known vulnerabilities to classify a file as a malicious PDF file while suspicious files are identified based on the existence of shellcode and JavaScript which is obfuscated.

Didier Steven has developed some utilities for analyzing PDF documents and one of these utilities is PDFiD which searches for 21 keywords [9], which enable you to determine if there is a JavaScript embedded in the file or there is an action when the file is opened. PDFiD searches for the keywords presented in Table 1:

Table 1 Features Extracted using PDFiD

obj
endobj
stream
endstream
xref
trailor
startxref
/Page
/Encrypt
/ObjStm
/JS
/JavaScript
/AA
/OpenAction
/AcroForm
/JBIG2Decode
/RichMedia
/Colors>2^24
/Launch
/EmbeddedFile

4. Enhanced Analysis Method

The overall design of the method is illustrated in Fig. 1. Given a PDF file as input file for analysis, the user selects the structure scan to be performed on the PDF file. Structure scan requires the keywords used for scanning that are available within keywords file identified by the user. Then the PDF file and the keywords file are read to calculate the hash values for them.
The hash value of the PDF file is checked if it is available in hash value database. If no the hash value is added to the hash value database, then structure scan is performed to add the output to output database folder to display the output to the user. If the hash value of the PDF file is included in the hash database, the PDF file is checked if it has been analyzed with the keywords file before. If it is, the output is displayed. If no the scan is performed using this keywords files, then the output is added to the output database and finally the output is displayed to the user.

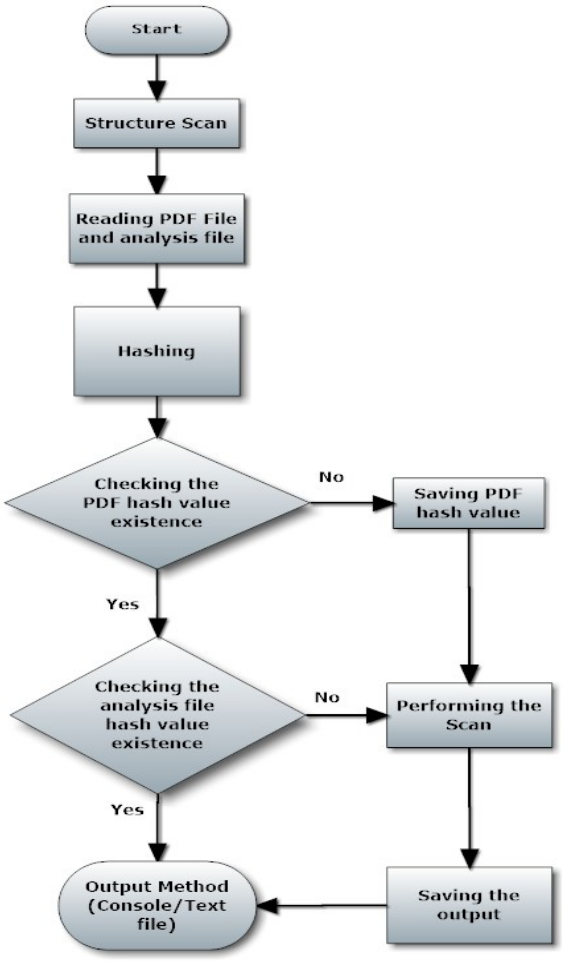

Fig.1 System Architecture

4.1 Structure Scan

The objective of this phase is inspecting the PDF document and searching for features which are important for labeling PDF documents as suspicious. PDF contains data represented in ASCII and binary format, therefore the PDF document is read as a byte sequence to easily parse it. The method does the scan on the chosen document and searches for keywords that help in giving a brief idea about the structure of the document like number of pages, and list possibly suspicious objects in it. On basis of Didier Steven's work [10], his suspicious keywords had been chosen in the method as they are the most significant features when scanning malicious PDF files [11] in addition to some more, with the ability of the analyst to search for another keywords as these keywords are presented in a text file and identified by the user.
The PDF structure enables encoding and compressing data within PDF files like images, for that attackers use filters to obfuscate JavaScript. From the filters used in PDF format, six of them are used for the objective of maliciousness [12]. In addition to the filters which are used for images compression (i.e. CCITTFaxDecode, DCTDecode) [3]. These filters which have been added as another set of keywords in the proposed method are:

- /FlateDecode
- /LZWDecode
- /RunLengthDecode
- /JBIG2Decode
- /ASCII85Decode
- /ASCIIHexDecode
- /CCITTFaxDecode
- /DCTDecode

As a JavaScript is found in a PDF file directly, it can be exist in another file or may be called from remote sites through the two keywords [6], which are used in the proposed method as a further set of keywords:

- /URI
- /GoTo

4.2 Calculate Hash value

The first action that should be performed is to calculate the hash value for the PDF document in addition to the keywords file, which will be used to identify and classify the malware samples as well as the elements inside PDF documents. It can be used when inquiring for documents that have been already analyzed which can save the user's time if the document has been analyzed before.

4.3 Reporting

This phase enables the user to choose how to display the output of the scan either on the console or copy the output to text file named by the name of the scanned PDF document.

5. Results and Analysis

The experiment was implemented using Windows 8 in Hyper-V virtual machine. The virtual machine has been configured to use one virtual processor and 1GB RAM with Python 2.7.8. The Hyper-V virtual machine has been used for the static analysis of PDF documents and to keep the host operating system safe from malicious dataset.
To test the method, an experiment was performed using a dataset which consists of 19593 benign and malicious PDF documents with total size of 918 MB, downloaded from the site; Contagiodump [13] which is a website contains up-to-date malware samples, threats and tests. Table 2 shows the properties of the dataset used in the experiment.

Table 2 PDF Documents collected for the experiment

Category	# of files	Size of files
Benign Files	8800	761 MB
Malicious Files	10793	157 MB
Total	19593	918 MB

When gathering samples from intermediary websites, to some extent it is not assured that some of them are malicious. The presence of malicious files in benign samples or contrariwise will produce negative results on the studied experiment. For that a copy of all documents in the malicious as well as in benign dataset was scanned using Kaspersky Endpoint security 10 antiviruses in a Hyper-V virtual machine.

5.1 Experiment

In this experiment regular expression (RegEx) in Python are used to search for the suspicious features and compute their frequencies in both the malicious and benign dataset.
After running the scan over the provided dataset, the results were achieved as shown in Table 3. The Percentages are calculated by dividing the number of files with a certain feature over the total number of sample.

Table 3 Structure Scan Results

	Malic-ious	Clean	%Malic-ious	%Clean
JavaScript	2766	298	14.12%	1.52%
JS	2758	290	14.08%	1.48%
mismatched objects	58	0	0.30%	0.00%
mismatched streams	29	7	0.15%	0.04%
PDFs with no Cross reference table	647	1560	3.30%	7.96%
PDFs with no Startxref	284	0	1.45%	0.00%
FlateDecode	3067	8597	15.65%	43.88%
LZWDecode	58	359	0.30%	1.83%
ASCII85Decode	205	57	1.05%	0.29%
ASCIIHexDecode	402	408	2.05%	2.08%
RunLengthDecode	53	0	0.27%	0.00%
JBIG2Decode	3	143	0.02%	0.73%
DCTDecode	96	1672	0.49%	8.53%
Encrypt	5	58	0.03%	0.30%
CCITTFaxDecode	1	471	0.01%	2.40%
OpenAction	1762	610	8.99%	3.11%
Launch	68	12	0.35%	0.06%
AA	89	352	0.45%	1.80%
Acroform	1714	2658	8.75%	13.57%
URI	1	1241	0.01%	6.33%
RichMedia	2	0	0.01%	0.00%
ObjStm	34	2924	0.17%	14.92%
EmbeddedFile	908	979	4.63%	5.00%
Page = 1	3144	3406	16.05%	17.38%
%EOF missing	6394	0	32.63%	0.00%
Bad Header	718	0	3.66%	0.00%
XFA	906	2	4.62%	0.01%
GoTo	8	485	0.04%	2.48%

Total Dataset = 19593

After rerunning the scan on the same dataset to find how the features present in the dataset and their relationships, the results presented in Table 4. Each feature has a symbol to simplify its representation.

Table 4 Features Presence in the files

Features	Symbol	Frequency in Malicious	Frequency in Clean
Bad Header	H	718	0
%%EOF missing	E	6394	0
JavaScript		2766	298
JS		2758	290
OpenAction		1762	610
XFA		906	2
(JavaScript ∩ JS) – (OpenAction ∪ XFA)	L	998	250
(JavaScript ∩ JS ∩ OpenAction) – XFA	R	1754	7
JavaScript – (JS ∪ OpenAction ∪ XFA)	Y	8	39
JS – (JavaScript ∪ OpenAction ∪ XFA)	Z	0	31
OpenAction – (JavaScript ∪ JS ∪ XFA)	V	5	603
JavaScript ∩ JS ∩ OpenAction ∩ XFA	J	3	0
(JavaScript ∩ JS ∩ XFA)	O	3	2

– OpenAction			
XFA – (JavaScript ∪ JS ∪ OpenAction)	X	900	0

According to the results presented in Table 4, the predicted number of suspicious files (P.Fs) are calculated as following:

$$P.Fs = H + E + L + R + Y + Z + V + J + O + X \quad (1)$$

By applying Eq.(1) on the results of the malicious files listed in Table 4, predicted number of suspicious files are:

P.Fs = 10783 suspicious files in malicious dataset

By applying Eq.(1) on the results of clean files listed in Table 4, predicted number of suspicious files are:

P.Fs = 932 suspicious files in clean dataset

5.2 Detection Accuracy

Prior to explaining the detection rates detected through the experiment, number of terms are presented here [14]:
True Positive (TP): The number of files detected as malicious from malicious samples.
True Negative (TN): The number of files detected as benign from benign samples.
False Positive (FP): The number of files detected as malicious from benign samples.
False Negative (FN): The number of files classified as benign from malicious samples.

In the experiment, the performance of the method is evaluated with regard to false positive and true positive rate:

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP+FN} * 100\% \quad (2)$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{TN+FP} * 100\% \quad (3)$$

The false positive and true positive rates of the method were evaluated regarding to the presence of JavaScript, JS, OpenAction and XFA tags and to the missing of the keyword (%%EOF) and missing the PDF header within the first 1024 bytes of the file, as shown in Table 5.

Table 5 Detection Results for Structure Scan

		Known Samples	
		Benign	**Malicious**
Detected Samples	**Benign**	TN = 7868	FN = 10
	Suspicious	FP = 932	TP = 10783

Regarding to Eq.(2) and Eq.(3) and Table 5, the true positive and the false positive rates are :

True Positive Rate (TPR) = 99.91%

False Positive Rate (FPR) = 10.59%

From the results above the method detected 10783 (99.90%) as suspicious files from the 10793 malicious files, and falsely detected 932 (10.59%) as suspicious from 8800 benign files.
To evaluate the method and how it can detect suspicious PDF files, it is compared with Wepawet. Wepawet analyzes PDF files by using interpreter to run JavaScript [15]-[8].
The comparison is conducted on 5000 known malicious PDF files regarding false negative which means the dataset is known to be malicious. To carry out the comparison, the experiment was performed using the keywords that were used in the authors' hypothesis mentioned in experiments results section. The relationship between the keywords used in the experiment is shown in Table 6 and the results of comparison are shown in Table 7.

Table 6 Keywords relationship - Structure Scan

Features	**Frequency in Malicious**
Bad Header	347
%%EOF missing	3010
JavaScript	1236
JS	1232
OpenAction	755
XFA	404
(JavaS1811+cript ∩ JS) – (OpenAction ∪ XFA)	478
JavaScript ∩ JS ∩ OpenAction ∩ XFA	2
(JavaScript ∩ JS ∩ OpenAction) – XFA	749
(JavaScript ∩ JS ∩ XFA) – OpenAction	3
JavaScript – (JS ∪ OpenAction ∪ XFA)	4
JS – (JavaScript ∪ OpenAction ∪ XFA)	0
OpenAction – (JavaScript ∪ JS ∪ XFA)	4
XFA – (JavaScript ∪ JS ∪ OpenAction)	399

Table 7 Comparison with Wepawet

		Detected Suspicious	**Detected Benign**	**False Negative (FN)**
Wetawet	4859	4693	166	3.41%
Proposed method using Structure Scan	5000	4996	4	0.08%

Regarding the analysis of the results presented in the table above, Wepawet missed 3.41% of the known malicious PDF files compared to the authors' method where the false negative rate is 0.08% using structure scan.
From the method evaluation with Wepawet, it can be seen that the results support the authors' hypothesis in selecting six features from 28, which are presented in Table 3, to be used as a significant features in detecting suspicious PDF

files which have a positive results on the performance of the classification method.

6. Conclusion

While the PDF documents are used by many users nowadays as stable and reliable document exchange technique format, it is highly used by hackers to run harmful code on computers. As the structure of PDF gives the ability to embed codes like JavaScript and communicate with outside sites.

In this paper, the structure format of PDF has been studied in addition to the techniques which are used by hackers to keep their harmful code away, and hidden from security specialist and security software like antivirus.

Enhanced analysis method was presented to detect suspicious PDF files via choosing the significant features that commonly found in malicious PDF files. As an additional step, an experiment was implemented to classify the PDF documents based on these keywords.

It can be notice from the results that missing the (%%EOF) is only in malicious file, which can be used as an indication for maliciousness.

The continuation of this paper is to utilize the proposed method to use YARA tool in order to scan PDF files looking for the keywords that have been used within them.in addition to use learning machine technique to classify PDF files as suspicious or benign.

Adobe needs to rethink about and limiting the features which are used to execute malware and adds a sandbox container that opens the PDF file and id any malicious activity is detected alerts the user and stop the execution.

7. Future Work

There are many fields that can be added to improve our work regarding to the PDF analysis process and its function.

- *Enhance the analysis method*

The system doesn't contain any JavaScript analysis functionality so applying a deobfuscation on JavaScript would further extend the analysis approach and to be able to extract the encrypted or encoded JavaScript.

- *Apply Fuzzy Hashing*

Using fuzzy hashing as another way to check for previously analyzed documents, in which the traditional hashes are often used to match the identical files and are unsuitable to match the files if one bit is changed.

- *Using Dynamic Analysis*

To create a module that analyzes PDF files using dynamic analysis by running them in a monitored virtual machine and analyzing it for any vulnerable behavior. The system presents detailed information about the files and their execution in the virtual machine.

8. Limitations

Even many malicious PDF files use the suspicious keywords mentioned in the experiments, there are also many benign PDF files use them, which make the method unable to differentiate between malicious and benign PDFs. The authors' method cannot detect any malicious PDF files that don't use these keywords as attack vector.

References

[1] M. Egele, P. Wurzinger, K. Christopher and E. Kirda, "Defending browsers against drive-by downloads:Mitigating Heap-Spraying Code Injection Attacks" Springer, pp. 88-106, 2009. doi: 10.1007/978-3-642-02918-9_6.

[2] Symantec, "Malware security report: Protecting your business, customers, and the bottom line," Symantec, 2010.

[3] Z. Tzermias, G. Sykiotakis, M. Polychronakis and E. P. Markatos, "Combining Static and Dynamic Analysis for the Detection of Malicious Documents", ACM, in *In Proceedings of the Fourth European Workshop on System Security*,No. 7, 2011. doi: 10.1145/1972551.1972555.

[4] F. Eric, A. Blonce and F. ,. L. Frayssignes, "Portable Document Format (PDF) Security Analysis and Malware Threats," 2008.

[5] Adobe, "PDF Reference and Adobe Extensions to the PDF Specification," 2008. [Online]. Available: www.adobe.com/devnet/pdf/pdf_reference.html http://www.adobe.com/devnet/pdf/pdf_referece.html. [Accessed 16 Fabruary 2014].

[6] P. Laskov and N. Šrndi´c, "Static Detection of Malicious JavaScript-Bearing PDF documents" , ACM, in *Annual Computer Security Applications Conference*, pp. 373-382, 2011. doi: 10.1145/2076732.2076785.

[7] M. Cova, C. Kruegel and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code", ACM, in *In Proceedings of the 19th international conference on World wide web* ,pp. 281-290, 2010. doi: 10.1145/1772690.1772720.

[8] S. Ford, M. Cova, C. Kruegel and a. G. Vigna, "Wepawet," [Online]. Available: http://wepawet.cs.ucsb.edu/. [Accessed 8 December 2014].

[9] D. Steven, "PDF Tools," [Online]. Available: http://blog.didierstevens.com/programs/pdf-tools/. [Accessed 21 February 2015].

[10] D. Stevens, "Didier Stevens," 29 March 2010. [Online]. Available: http://blog.didierstevens.com/2010/03/29/escape-from-pdf/. [Accessed 13 December 2014].

[11] Stevens, D. Malicious-pdf-analysis-ebook.pdf. http://didierstevens.com/files/data/malicious-pdf-analysis-ebook.zip. [Accessed 21December 2014].

[12] K. Itabashi, "Portable Document Format Malware," Symantec Security Response, 2010.

[13] Mila, "CVE-2013-0640 samples listing," 24 April 2013. [Online]. Available: http://contagiodump.blogspot.com. [Accessed 21 November 2014].

[14] S. Sayed, R. R. Darwish and S. A. Salem, "A Real-Time Approach for Detecting Malicious," Springer, in *Advances in Intelligent Systems and Computing*, Vol. 240, pp. 355-364. 2014 doi: 10.1007/978-3-319-01857-7_34.

[15] M. Cova, C. Kruegel, Vigna and a. G., "Detection and analysis of drive-by-download attacks and malicious javascript code", ACM, in *International World Wide Web Conference (WWW)*,pp. 281-290, 2010. doi: 10.1145/1772690.1772720.

Suleiman J. Khitan received the B.S. in Computer Engineering from Mutah University, Jordan in 2006. He is currently pursuing Final year M.Sc. Information System Security and Criminology from Princess Sumaya University for Technology, Jordan. His area of interest includes PDF analysis and network security.

Dr. Ali Hadi received the B.S. degree in computer science from Philadelphia University, Jordan, in 2002 and the M.Sc. and Ph.D. degree in computer information system from University of Banking and Financial Sciences, College of Information Technology, Jordan, in 2004 and 2010, respectively. He's a Senior Level Information Security Officer with 14+ years of professional experience working for different high-reputed companies. Since 2011 he's been teaching different computer security, digital forensics, and networking courses. He's also an author, speaker, and freelance instructor. His research interests include digital forensics, operating systems internals, malware analysis, and network security.

Prof. Jalal Atoum is currently the Vice President at Princess Sumaya University for Technology (PSUT). He had received his B.S. degree in computer science from Yarmouk university-Jordan in 1984. He had received his Master degree in computer science from University of Texas at Arlington-USA in 1987. He had received his PhD in computer science from University of Houston-USA in 1993. He had worked as an assistant professor at Yarmouk University from 1993 to 1995. He had been appointed as the Computer Science department Chairman at PSUT. He has supervised or co-supervised several students on their Ph.D. dissertations and several M.S. theses and has supervised numerous undergraduate graduation projects. Finally, he have been involved in several committees for degree plans, proposed and developed the Master program in Information System Security and Digital Criminology at PSUT.