Improving Operating System Fingerprinting using Machine Learning Techniques

Taher Al-Shehari and Farrukh Shahzad

***Abstract*—Operating System (OS) detection is one of the main concerns for computer security. The previous works that have been done on operating system detection, exploit some features of TCP/IP traffic based on a single packet. In this work, we built a system where TCP/IP communication is setup between machines to capture and analyze TCP/IP packets for more accurate and fine grained OS detection using our novel packet correlation approach. We used existing signature matching methods, extend it and employed machine learning techniques to detect remote operating systems with improved accuracy. We also employed mobile systems like smart phones and tablets to perform mobile OS fingerprinting. The tools we created also established encrypted communication using Secure Socket Layer (SSL) network protocol to investigate the effect of SSL communication on OS fingerprinting. The result of our experimental work showed that fine grained OS detection can be achieved for modern and mobile OSs using our approach.**

***Index Terms*—OS fingerprinting, remote operating system detection, vulnerability assessment, mobile operating system.**

I. Introduction

Today everyone is connected to the internet so the need to secure him from the intrusions is very important. What would happen if a business company that sells its goods on the internet went down for only one day? Or what happen if a bank was hacked and taken down? This external threaten for the companies trigger them to use multiple security applications like firewalls/intrusion detection systems (IDSs) in order to secure themselves from the hackers.

The operating system fingerprinting is a process of remotely detecting and determining the identity of a target system by observing the TCP/IP packets that are generated by that system. The operating system detection can be viewed from two sides. First, from the negative point of view for the hackers needs. For example, the hackers detect OS in order to exploit its vulnerabilities for their hacking purposes. Second, from the positive side for the network administrators needs because it is important for them to collect as much information as possible about their networks. It is also necessary for the system administrator to have certain statistics about the components that they have in their environment. For example, if there is a machine in the network that runs an old version of operating system which could be an easy target to be exploited by the hackers. By using OS fingerprinting, network administrators can know which machine's OS need an upgrade. Moreover, it is very difficult for the network administrators to have full control of what are connected to the network especially for large networks. For the system administrator, it's always important to be one step ahead of the attacker. This way, the attacker can't make use of the latest vulnerabilities. It is also important for the network administrator to be sure that each OS in the network satisfies the applied policies. For instance, when a user formats his PC and reinstalls an old version which violates the company policies. Detecting such situation in an automated way is very important, especially for large networks. "Having access to an up-to-date network inventory could allow a company to save money by canceling the license and support service for an OS that is no longer used"[1].

Now a days, network administrators also want to know which mobile devices, like smart phones and tablets, are accessing his/her network. It may be more difficult to respond to network attacks initiated by a wireless device. In some cases, the mobile users may not be authorized and can cause network overload as network load estimation might have not included on-the-fly wireless users.

There are two basic method of performing OS fingerprinting. The *active* detection is achieve by sending a special packet to the target machine and get the response that can be analyzed to identify the OS type of the target machine. The main weakness of active OS fingerprinting method is that it cannot be done if the target system has firewall and intrusion detection systems (IDSs) [2]. On the other hand the passive scheme of OS fingerprinting is done by sniffing the network packets remotely instead of sending a crafted packets to a target machine [3]. The idea of passive OS fingerprinting is to analyze the headers of TCP SYN packets (or other specific packets) to determine the operating system. After the needed packets are sniffed they are compared with predefined database that contains signatures of different operating systems, and determine the type of the OS that these packets come from. It is important for network administrators to do OS fingerprinting in a passive way in order to overcome the limitation of active method due to firewalls/IDSs.

The three way handshake is the main step for the initiation the TCP connection. First, the client initiates the connection by sending a request with SYN flag set to a server. If server is ready to open the connection, it replies with SYN+ACK packet, or if it is not ready, it replies to the initiator with RST packet. Then finally client replies with an ACK. The passive OS detection can exploit some parameters in the TCP/IP packets when SYN, SYN+ACK or RST flags are set [4]. When communication is done, client terminates the connection by sending the packet with FIN+ACK flag set.

Manuscript received June 23, 2013; revised August 20, 2013.

The authors are with ICS department at King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia (e-mail: g200905290@kfupm.edu.sa, farrukhshahzad@kfupm.edu.sa).

The TCP header has multiple flags that are set indicating the TCP connection status [5]. In the passive OS detection the main focus is in the parameters of the packet headers which are time to live (TTL), window size (WS), don't fragment bit (DF), and TCP options/flags. The main advantage of passive OS detection for the attackers is that they can detect the remote host without leaving any traces [5].

There are few tools that were developed to perform OS fingerprinting. These tools have limitations that need to be solved. For example, the active OS fingerprinting tools face a firewall or IDS in front of the target system which can be detected only using passive OS fingerprinting tools. Also passive tools have some limitations. The signature databases need to be updated continuously otherwise the newer operating systems will not be recognized on the internet any more [5]. The establishment and maintaining a good up-to-date fingerprint database requires some serious research in the area of OS security. Many performance measurements for evaluating passive and active OS fingerprinting are described by Thomas and Greenwald [6].

The rest of the paper is organized as follows. Section II presents the literature review. Section III demonstrates our proposed framework and section IV provide details of implementation. In Section V, the results of our experimental work are analyzed and compared. Finally, conclusion and future work are discussed in Section VI.

II. Literature Review

There is some research in the field of passive and active OS fingerprinting in the last 10-12 years. In [7] Gordon Lyon proposes several programs: checkos, sirc, and SS which are capable of fingerprinting various types of OSs by using TCP/IP traffic. The limitation of these tools is that they are not be referenced anymore because the information that is available by them is too limited.

Michal Zalewski [8] writes the first version of p0f tool for doing passive OS fingerprinting. There are four fingerprinting methods that are used in different scenarios as follows:

1) What is the system that is connecting to yours?
2) What is the system that you are connecting to?
3) What is the system that is refusing your connection?
4) What systems do you have a connection with?

Only the first one is supported well because it detects OS by analyzing the headers of the initial SYN packet.

Lanze Spitzner in [1] identifies what passive OS fingerprinting is, how it works and how to use it. He also compares between passive and active fingerprinting in terms of differences and similarities. He also talked about knowing your enemies and your assets, because when you know your enemies it is much easier to protect yourself against danger.

Gerald A. Marin in [9] looks at the general network security by covering the crucial basics of system security. He describes different attacks such as Distributed Denial of Service (DDos) attack, land attack and Smurf attack. Several countermeasures are discussed in the paper like what IDS is and how to stop malicious code, Trojans and worms.

Authors in [10] propose a masking approach to secure systems from OS fingerprinting. The paper also discusses the main steps that the operating system fingerprinting tools go through in order to detect the remote OS. They describe some active operating system fingerprinting tools like Xprobe2 and Nmap. The paper also discusses the countermeasure for preventing operating system detection.

Greg Taleck in [3] entitled paper Ambiguity Resolution via Passive OS Fingerprinting looks at exploiting the differences in the common OSs to evade intrusion detection systems (IDSs) detection for attacking. He proposes an approach that uses passive OS detection in order to resolve the ambiguities between different networks stack implementations in a correct way. A new technique that this paper looks at is to increase the level of confidence of OS detection by looking closer at the TCP connection negotiations.

In [11] Vladimir Lifschitz identifies ASP as "representing a given computational problem by a logic program whose answer sets correspond to solutions, and then use an answer set solver to find an answer set for this program". The author presents a scenario to claim that this approach is optimal and the test results of this ASP fingerprinting is very promising. The accuracy of recognizing 95 OSs tests is more than 80%.

Esfandiari, Bertossi, and Gagnon in [12] perform OS fingerprinting using Answer Set Programming (ASP). The main idea is that they do not consider just a single packet for determining the target OS but they analyze more packets in order to improve the accuracy of OS detection.

We found no published work that fingerprint operating systems based on correlation of multiple packets during the same communication session. Our main contribution in this work includes:

1) We build a client-server system which makes capturing, the appropriate packets for fingerprinting, simplified and automated. This is a 'hybrid' approach as active communication is initiated (but no special packets were injected) to perform passive fingerprinting.
2) The system also implemented packet capturing over Secure Socket Layer (SSL) encrypted communication network to analyze the effect of SSL on OS fingerprinting.
3) Due to exponential rise of mobile computing, we also captured packets from mobile devices for fingerprinting using third party socket client apps.
4) We used the latest p0f signature database [13] and convert it into a relational table to improve the performance of signature matching algorithm.
5) We found that by correlating the SYN and FIN+ACK packets during the same communication session leads to more accurate OS fingerprinting.
6) For new OS releases and Mobile OS, we employed machine learning techniques on extended p0f datasets.

III. Proposed Framework

In Our framework, OS fingerprinting is achieved in multiple phases. The main components of our framework are shown in Fig. 1. The first phase is to capture relevant TCP/IP packets from network traffic. Then these samples are passed to a matching component to compare with the existing fingerprint database. If the exact match is found the process ends. Otherwise, the data is processed using machine learning techniques by trained classifier which tries to find the closest match. In our framework, we are only interested in the SYN and FIN packets.

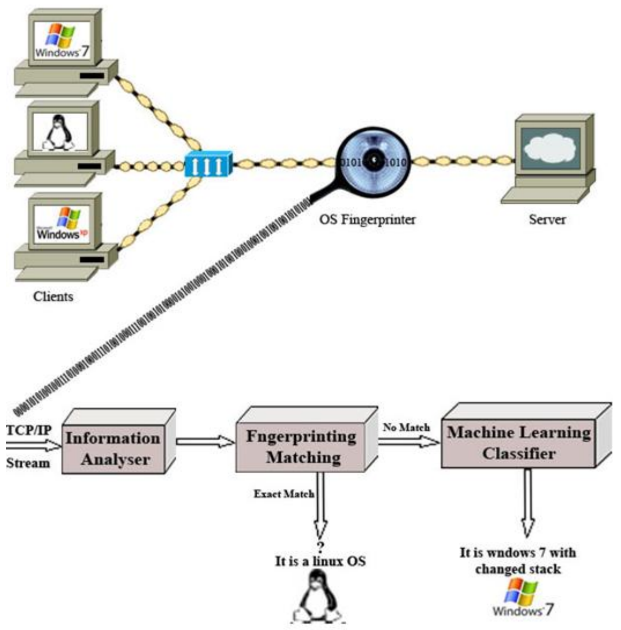

Fig. 1. Framework for OS fingerprinting architecture.

A. Data Extraction

Extracting the relevant information from packets stream is the first step in our model which can be captured live or from stored traffic. The most important information of TCP/IP headers that are interesting for our fingerprint is TCP SYN segments. The collected headers are transformed into p0f format for matching and classification phases.

Table I shows an example of p0f database format where TTL is the initial time to live, D is do not fragment flag either 1=true/0=false, WSS is the window size that represents the whole size of TCP/IP headers. As explained earlier, these parameters are OS dependent which makes it possible to perform OS detection.

TABLE I: AN EXAMPLE OF P0F FINGERPRINT

OS	WSS	TTL	D	size	options
Linux	S4	64	1	60	M*,S,T,N,W7

B. Fingerprint Matching

The next step is to match p0f fingerprint with p0f signature database. If there is an exact match the target OS is identified. If there is no exact match the newer system will be considered as unknown so this fingerprint will be passed through a trained classifier for OS detection.

C. Machine Learning Classifier

In case there is no exact match, the classifier is triggered and the heuristics are exploited to find a match between the target fingerprint and the extensive predefined operating systems database. In this phase many classification algorithms can be used to find the closest match among the known classes. It has been noted that some training algorithms are better suited for OS fingerprinting [14]. We utilize C4.5 algorithm which is based on decision tree-based approach.

This classification problem can be stated as follows: Consider a set P of TCP/IP packets and a set of client machines M, where each machine $m \in M$ has a known, labeled operating system OS(m). Each machine m sent SYN packet $p_{\mathrm{SYN}} \in P$ to server machine. The data collector records the packet $p_{\mathrm{SYN},}$, the server response $p_{\mathrm{SYN+ACK}}$ for each p_{SYN}, and corresponding $p_{\mathrm{FIN+ACK}}$ (on socket close). This yields a set of samples S for the classifier C.

A classifier C takes as input the set of samples S and produces a fingerprinting detection tool D_t. The tool D_t takes as input a sample s and returns the best OS label for the sample's machine $s(m)$. The tool is a function f such that $f(s) = \mathrm{OS}(s(m))$ for all $s \in S$.

To solve the OS classification problem, this tool D_t should not only correctly return the OS of all samples in S, but it should also correctly return the OS of previously encountered samples not in S.

D. Preprocessing Step for Data Classification

Before the classification step the data must be transformed into the format that is compatible with WEKA tool which called Attribute-Relation File Format (ARFF). This format starts with a header for its description. Then all events are stored in ARFF file with comma separated values each on their own row. The ARFF format is based on p0f format rules so for each field in p0f fingerprint, a specific attribute is defined. The order in ARFF format is not considered but it is important in TCP options so it is necessary to encode the order in ARFF file. To tackle this issue ten separate attributes are specified for each option in order to allow them to have any of the options. The result of the classification (detected OS) is the final attribute in the ARFF file which represents the target system that generates the transformed p0f fingerprints.

E. Defining Relevant Parameters

Determining the most relevant information from TCP/IP headers is an important step for OS system detection. These relevant parameters are chosen dynamically for the classifier because there may be a new OS fingerprint contains some header fields that are not considered before in the database to be able to match it with the predefined OS classes.

In Weka, the complete set of samples is partitioned into subsets. A single subset is used to validate the model, while the other subsets are used to train the model. We choose a ten-fold cross-validation, so ten subsets are created. The complete process is repeated ten times, each time with a different subset used as the validation subset and the rest as the training data for the model.

F. Decision Tree/C 4.5 Classification Algorithm

In our experiments we select C4.5 classification algorithm [14] because it is well known with its high accuracy of classification. This algorithm goes over samples of training set many times in order to build an optimal classification model. This algorithm handles the continuous and discrete attributes where the continuous are supported by using thresholds. Furthermore, the training set with missed attribute values can be handled using this algorithm. The algorithm goes up the tree when it counters an instance of a new class. Then the algorithm creates a decision node in the tree for the attribute that will give the highest information gain. After that it will recurs down the tree and removes the sub-trees that are not needed by replacing them with leaves. The pruning feature of this algorithm makes it possible to create the model in seconds and classifies with better accuracy.

IV. Implementation

To evaluate our proposed framework, we built a Java package *edu.kfupm.ccse.osfp* with several classes to process and transform relevant packets from pcap format to p0f and ARFF (for Weka learning tool) format.

The p0f is one of the commonly used signature format for SYN based OS fingerprinting (Table I). We converted the most recent p0f signature data file [13] into MySQL relational table which makes matching and adding new signature easier and streamlined. We also extended p0f to include other fields as discussed in later section. The original p0f SYN signature can classify the remote OS into genre like Linux, BSD, Windows, etc. It can also distinguish some older OS versions accurately. But most of the newer OS can't be classified at the version level. We found that our approach can lead to more accurate and fine grained OS detection.

A. Packet Capturing and Extraction

We developed two sets of client/server Java socket applications. One set used normal java socket API and other used SSL socket API. The server runs on a certain machine and multiple client applications connect to the server simultaneously (from other machines). The client application binds to the server (SYN) and then disconnects (FIN+ACK). There was no actual data communication. Therefore only 3 types of packets were captured namely SYN, SYN+ACK and FIN+ACK as it is shown in Fig. 2. The wireshark, Windows network monitor and/or network miner tools are used to capture the packets on the server machines. The captured packets are saved in the tcpdump's pcap format. For mobile devices, third party TCP/IP client apps were utilized. Although above setup basically performed passive fingerprinting as no special packets were injected, but one can argue that only specific packets between selected machines (which are executing custom made applications) are captured. Therefore we can call it 'hybrid' fingerprinting.

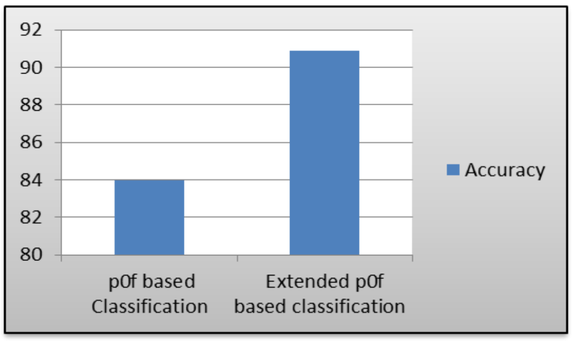

Fig. 2. Normal vs. Extended OS fingerprinting Accuracy (%).

To extract the right information from TCP/IP headers we build a Java application based on JnetPcap library, which is a Java wrapper for native libcap library [15], to generate fingerprint entries in p0f format.

B. C4.5 Classifier

The Weka classification tool needs the training dataset to be in the ARFF format. For this purpose, we build a converter that converts p0f fingerprints into ARFF format. This dataset is fed to the Weka application for classification. For normal p0f based classification, we have total of 31 attributes.

C. Extended P0f

Now we present how we correlate packets from same communication session to extend the p0f signature format. Basically, we link SYN packet and FIN+ACK packet using hash map. The key used for hashing is the contacted string containing source IP, destination IP, source port and destination port from the SYN packet. This key is matched with the same concatenated string from the few succeeding FIN+ACK packets. Our capturing model dictates that those two packets should not be far apart in the pcap file.

We also use Weka tool to classify OS's using extended p0f format. The idea is that with more attributes, a more accurate classification can be achieved. Table II shows an example of extended p0f.

TABLE II: An Example of Extended P0f Fingerprint

OS	WSS	TTL	D	size	options	FIN-WSS	FIN-TTL	FIN-D	FIN-size
Win-8	8192	128	1	52	M*,N,W8,N,N,S	260	128	1	40

V. Experimental Evaluation and Analysis

TABLE III: Platform Specifications

OS	Hardware Spec	Network
Windows 8 Pro (2 machines on separate network)	Intel(R) Core(TM) 2 Quad CPU Q9400 @ 2.67 GHz, 4/8 GB RAM	Ethernet/LAN
Windows 8 32 bit	Intel core 2 CPU 2.13 GHz, 4 GB RAM	Wi-Fi
Windows 7 Ent. (Multiple units)	Intel(R) Core(TM) 2Quad CPU Q9400 @ 2.67 GHz, 4 GB RAM	Ethernet/LAN
Linux Red hat 5.4		Ethernet/LAN
Windows XP 2002 SP2	Intel Pentium 1.86 GHz, 512 MB RAM	Ethernet/LAN
Android 2.2.2	Sharp-AD51, Kernal 2.6	Wi-Fi
iOS 5.1.1	iPad 3	Wi-Fi
Win CE 6.0	AMTEL, T7A HMI panel	Ethernet/LAN

TABLE IV: Tools and Libraries

Tool/Library	Version
JAVA for programming	1.7
Wireshark for capturing	1.8.6
Network Miner	1.4.1
Microsoft network Monitor	3.4
WEKA for classification	3.6
SSL protocol for encryption	Java Keytool (RSA)
JnetPcap library	1.3

For our experiment, we select two different environments. In the university setting, we select few machines running windows-8 or windows-7 as servers and we setup our client application on few window machines, one Linux machine and one android smart phone. In home setup, we use two windows-8 machines (64 bit and 32 bit), windows XP machine, one android device, one win-CE device and an iPad (Table III). The tools used are specified in Table IV.

A. *Results and Analysis*

We captured TCP/IP packets on different networks for 10 days. First we ran p0f matching algorithm on some sample pcap files. Table V shows the summary of result. As discussed earlier, we utilize relational table for p0f signature matching. About 30% of packets in sample 1 were not matched to any OS. Furthermore, the matching is very coarse as the existing p0f database mapped multiple OS releases to same signature.

TABLE V: P0F DATABASE MATCHING

OS matched	Sample 1	Sample 2
Total SYN packets captured/processed	30060	1259
Windows Vista SP1, 7 SP1	1099	100
Windows Vista SP0/SP2, 7 SP0+, 2008 SP0	16161	658
Windows 2000 SP4, XP SP1+, 2003	1931	491
Windows 2000 SP2+, XP SP1+ (seldom 98), Vista SP1, 7 SP1, 2008 SP2	650	10
Linux and Others	724	0
Unknown	9495	0

TABLE VI: NORMAL VS. EXTENDED OS FINGERPRINTING COMPARISON

Parameters	Normal p0f	Extended p0f
Instances	2078	2078
Attributes	31	35
No. of leaves	10	11
Size of tree	13	15
Correctly Classified Instances	1745	1888
Accuracy	83.97%	90.86%

Next, we employed our Java application to generate p0f and ARFF files from 8 raw pcap files. These files were captured as describe in Section III. Similarly, we use a separate Java application to generate extended p0f and ARFF files from same 8 raw pcap files.

Finally, we execute the Weka tool with combined ARFF dataset separately for p0f and extended p0f based instances. We use J48 (an implementation of C4.5 algorithm) with 10-fold cross-validation test mode.

The results for two classifications are compared in Table VI. With four more attributes, the extended p0f classifier creates 15 trees as compared to 13. The detection accuracy for extended p0f based classification is about 91% as compared 84% for normal p0f based classification (Fig. 2). This result shows higher accuracy when compared to related work [14], [16] especially with extended p0f based classification.

VI. CONCLUSION

In this paper, we presented a hybrid approach for automated and more accurate OS fingerprinting. Several Java tools were built to capture, process, transform, match, analyze and classify appropriate TCP/IP packets. Our research showed that by correlating packets from same TC/IP session, fine-grained OS detection can be achieved for modern operating systems and mobile devices. We also noted that SSL TCP/IP communication doesn’t show any significant differences which can effect fingerprinting.

We believe that we can achieve even finer OS detection if we have resources like computers/devices running different releases of operating system. This means we may be able to distinguish between Windows-8 64 bit and Windows-8 32 bit or iOS 5.1 and iOS 6.x. Since we have tons of smart devices in the market today, including smart phones, tablets, game consoles, consumer electronics etc., more research is needed to remotely detect the OSs running on these devices. Furthermore, new tools need to be built, if these devices use communication protocol other than TCP/IP.

REFERENCES

[1] L. Spitzner, *Passive fingerprinting*, vol. 3, pp. 1–4, May 2003.

[2] L. G. Greenwald and T. J. Thomas, “Understanding and preventing network device fingerprinting,” *Bell Lab. Tech. J.*, vol. 3, pp. 149–166, 2007.

[3] G. Taleck, “Ambiguity resolution via passive OS fingerprinting,” in *Proc. the 6th International Symposium on Recent Advances in Intrusion Detection*, 2003, pp. 192–206.

[4] P. B. Falch, “Investigating passive operating system detection,” M. S. thesis, University of Oslo, May 24, 2011.

[5] G. Francois and E. Babak, "A hybrid approach to operating system discovery based on diagnosis theory," in *Proc. Network Operations and Management Symposium, 2012 IEEE*, pp.860–865, 2012.

[6] G. G. Lloyd, and T. J. Thomas, “Method and system for evaluating tests used in operating system fingerprinting,” US patent 11/888,925, Jan. 8, 2013.

[7] G. Lyon, “Remote OS detection via TCP/IP Stack Finger-Printing,” *Phrack Magazine*, vol. 8, no. 54, December 1998.

[8] M. Zalewski, *p0f 2*, README, 2006.

[9] G. A. Marin, “Network security basics,” *Security & Privacy, IEEE* , vol. 3, no. 6, pp. 68,72, Nov.-Dec. 2005.

[10] S. Kalia and M. Singh, “Masking approach to secure systems from operating system fingerprinting,” *TENCON 2005 2005 IEEE Region 10* , vol. 1, no. 6, pp. 21-24, Nov. 2005.

[11] V. Lifschitz, “Answer set programming and plan generation,” *Artificial Intelligence*, vol. 138, issues 1–2, pp. 39-54, June 2002.

[12] G. Francois, E. Babak, and L. Bertossi, "A hybrid approach to operating system discovery using answer set programming," *Integrated Network Management*, pp. 391- 400, May 21, 2007.

[13] Software Engineering Institute Carnegie Mellon. [Online]. Available： https://www.tools.netsa.cert.org/confluence/display/tt/p0f+fingerprints

[14] J. Schwartzenberg, “Using machine learning techniques for advanced passive operating system fingerprinting,” MS thesis, 2010

[15] TCPDUMP/LIBPCAP team. (2010). [Online]. Available: http://wwww.TCPDUMP/LIBPCAP public repository

[16] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

Taher Al-Shehari is a master student at King Fahd University of Petroleum and Minerals. He got his B.Sc. degree in Computer Science from King Khalid University in 2007. He granted the Upper Second Class Honour by the rector of the university. His research interests include plagiarism detection, wireless video streaming, website and OS fingerprinting on anonymity protocols. His 2012 paper, "Wireless video streaming over Data

Distribution Service middleware" (with Al-madani, B.; Al-Roubaiey, A.), published in Software Engineering and Service Science (ICSESS), was invited for presentation in the IEEE 3rd International Conference on June 2012.

Farrukh Shahzad was born in Karachi, Pakistan. He did his BE(Electrical Eng..) from the NED University of Engineering & Technology, Karachi, Pakistan in 1992 and his MSEE from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in 1996. He is currently a Ph.D. student and lecturer at KFUPM.

In 1996, he moved to USA. He has 16 years field experience in product design, software development, engineering, and implementation of many M2M and satellite based remote monitoring systems. His current interests include Cloud storage security, Big Data analytics, Data sciences, Machine learning and business intelligence. He is US copyright holder of four Engineering softwares. His research activities resulted in publication of more than 10 technical papers in IEEE and other journals.