I.J. Modern Education and Computer Science, **2014, 12, 41-46**
Published Online December 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2014.12.06

A Secure Framework for Discovering the Liabilities of a Network Server

Ulya Sabeel
Amity University, Amity School of Engineering and Technology, Haryana, India
Email: ulya.sabeel4@gmail.com

Saima Maqbool
Islamic University of Science and Technology, Awantipora, Pulwama, Jammu & Kashmir, India
Email: saimabhat02@gmail.com

Abstract—The role of network and information systems is increasing in our day today infrastructures that provide critical services like banking and commerce applications, telecommunications, distribution, transportation, etc. This increases the levels of dependability on such applications, which brings about several pros and cons as well. If these infrastructures are damaged, it can compromise the availability of the critical services. Thus, it is the need of the hour to secure the available network and information systems from malicious attacks. In this paper, we have proposed a secure technique named as Network Liability Detection (NLD) and a software we named as Network Liability Tool (NLT), that points out the liabilities or vulnerabilities of your system that can be exploited to compromise its security. We propose a model based approach where the behavior of each component of the system is carefully monitored to find out the well known as well as as-yet-unknown loopholes in the system. A prototype of the application is built in Windows Platform using Java, to demonstrate the entire functioning of this system and helps in solving the security related loopholes in the network servers. The speculative results affirm that the proposed technique is effective in detecting the liabilities of the network servers.

Index Terms—Information System, Liability detection, loopholes, Network security, Secure Framework

I. Introduction

The increased credence on sophisticated computer systems is increasing day by day. Various emerging technologies, enhancements in network storage and security as well as Internet are used as flagstones for paving the way for the reliable use of computers for large organizational as well as personal level. Every business or organization use software development life cycle (SDLC) for the development of system software. At each level of this SDLC, certain personnel are allotted to carry out different tasks. The system software built may have some flaws/ bugs that are hidden. These hidden flaws that create weaknesses in the overall security of a computer or network as a whole are called vulnerabilities/ liabilities. These may also be caused by improper configurations [1]. Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw [2]. These loopholes are exploited by the attackers/ adversaries to gain illegitimate access to the network which results in potential damage to the system and the entire network as a whole.

The main aim of the proposed technique, Network Liability Detection (NLD) is to zealously introduce attacks on target server and inspecting the process of execution of the server to find out the flaws in the system that lead to failures. If the normal behavior is deviated from the expected behavior, it indicates the presence of errors in the system server. Afterwards, it is the responsibility of the debugging team to identify the types of flaws in the system and suggest procedures to rectify them such that the server becomes more secure. One of the best features of this Network Liability Tool (NLT) is that it just needs the protocol specifications of the target server to derive test cases and launch attacks on it to detect any liabilities. The proposed methodology supports three phases- Attack Generation Phase, Attack Induction phase and Monitoring and removal phase. The two types of attacks that we have used here are: Bulk Data and Redundancy, which if done on large scale can result in Denial of Service Attacks (DoS). The DoS attack launched by the adversaries provides illegitimate access to them and they consume the entire resources rendering the legitimate users with poor response or even no services. The rest of the paper has been divided as follows: section II consists of the related work, section III consists of the proposed methodology, section IV provides the experimental results and finally we come to conclusion in section V.

II. Related work

In this paper we have proposed a methodology and a tool for discovering the liabilities of a network server by launching attacks on it. We have chosen a network server because it is the entity that needs maximum protection. Our work has been influenced by many previous research works. Some of them are given below:

In [3], the authors have proposed a methodology to automatically discover vulnerabilities in network servers using a tool called AJECT. They have conducted 58 attack experiments with 16 email servers running POP and IMAP services. In [4], the authors have designed a framework for discovering the router protocol vulnerabilities and proposed a mathematical model Two-Stage Fuzzing Test Cases Generator (TFTCG) and RPFuzzer. In case of [5], the authors have proposed a mechanism to find out the vulnerabilities and given its solutions for packet sampling. A simple mechanism called Buttercup has been proposed in [6] to counter against the attacks on buffer overflows. Another method to detect the vulnerabilities of a server has been proposed using SQL Injection by prototype tool SQL InjectionGen [7]. Other approaches for fault removal and fault forecasting have been given in [8], [9]. Some of the tools like Xception and FTAPE have been proposed that can inject hardware and software faults in target system [10], [11]. These tools are relatively simple and it is difficult to apply these to more complex faults. However, our software Network Liability Tool (NLT) can induce attacks as complex as DoS to check the functioning of our network server thus making it more secure.

Fuzzers inject random input sample codes to the software to check their vulnerabilities. In [12], the authors have proposed an approach to send scrambled command line characters and degrade the execution of an application. Other approaches that use fuzzers for detecting vulnerabilities have been given in [13], [14], [15]. However, there are some disadvantages of fuzzers. They can be too simple or too specialized to be reused again in different applications. Also, they do not provide any monitoring mechanisms. On the other hand, our tool NLD can be used to provide security to the target system which lies in any platform and can work with any underlying protocols.

Some commercial vulnerability scanners have also being described in literature such as Nessus [16], SAINT[17] and QualysGuard[18]. They have a database with the server vulnerability possibilities and several attacks for detection. Certain other vulnerability analyzers have been proposed that find out flaws in the program code itself that are usually associated with buffer overflows [19], [20], [21]. These tools also have some limitations of generating false positives and missing certain vulnerabilities. Certain Runtime Prevention Mechanisms which protect from Buffer Overflows, have been proposed such as StackGuard [22], PointGuard[23].

III. The Proposed Methodology

We propose a new methodology of inducing attacks on the target server by itself in order to discover the loopholes of the system under observation. It behaves like an adversary and launches attacks on the target server while monitoring its functioning. If any abnormal behavior is detected, it indicates the occurrence of some flaws that need to be removed for the security enhancement of the target server. A series of different attacks are imposed on the server and its behavior is monitored continuously. All these details are stored in the database provided with the server. This database may be referred later by the debugging team to find out the major flaws in the system and proceed with their elimination.

In order to be highly confident about the lack of loopholes in our system, the attacks should be exhaustive and should look out for as many loopholes as possible. For this purpose, a large number of test cases may be generated.

A. Phases Involved

There are 3 phases in this technique: Attack Generation Phase, Attack Induction Phase and Monitoring & Removal Phase.

i) Attack Generation Phase:

Attacks such as Redundancy and Bulk data (used to launch DoS) are generated using the proposed tool, Network Liability Tool (NLT). Redundancy, as the name indicates, means sending the replicated copies of data and Bulk data means sending large chunks of data, higher than the capacity of the network. Each attack represents a single test case that exercises some part of target system.

ii) Attack Induction Phase:

In this phase, both Redundancy and Bulk Data, generated above, are used to flood the network on a large scale to launch Denial of Service attacks, which render the legitimate hosts with poor response or no service at all and provide services to the adversary launching the attack. The NLT behaves like a superficial attacker that attacks the target server, while monitoring its functioning. This methodology can be a useful aid in the increasing reliance on computers because it helps in discovery of such ambiguities in our systems. Many test case definitions are used at different phases.

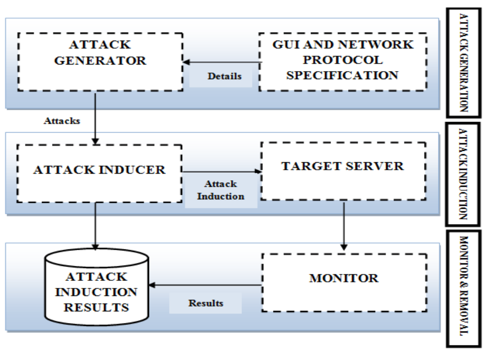


Fig.1: Architectural Diagram

iii) Monitoring and Removal Phase:

Monitor judges the server's performance whether it's adequately handling user needs and running well. It identifies any deviation from the expected behavior and stores all the results in the server database. This information can be later processed by the debugging team to find out the areas of flaws, types of flaws, ramification of flaws and their remedies.

The diagrammatic representation is depicted by the architectural diagram given in fig.1.Fig.2 depicts the sequence diagram for the entire process of NLD.

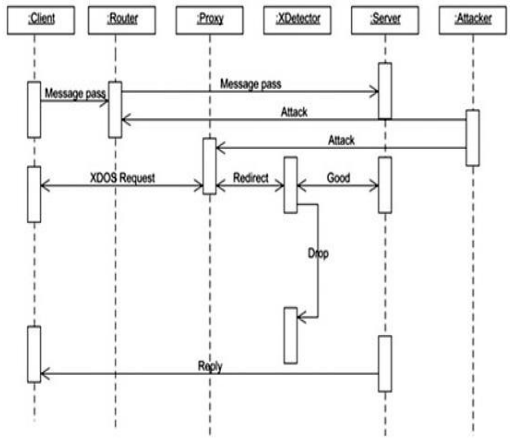
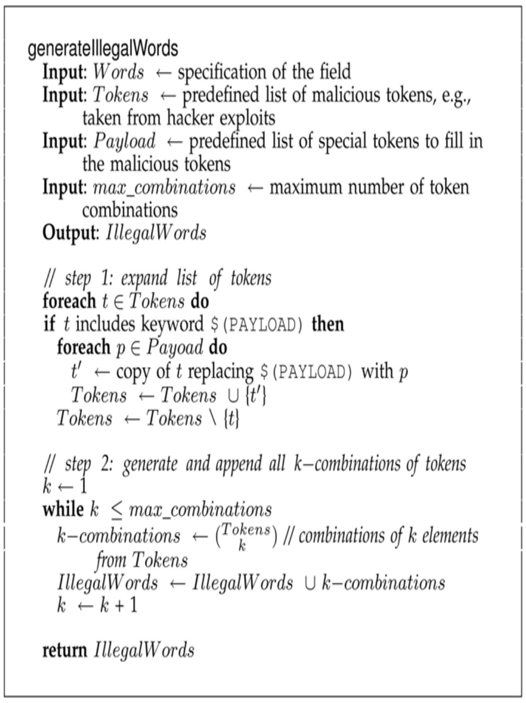


Fig. 2: Sequence diagram for NLD

B. Algorithms Used

The basic algorithms used here have been proposed in [3]. The first algorithm is used for test case generation (attacks) as shown in fig.3. This algorithm is used to find out whether the target server can grapple with the abnormal data being sent to it. All states and message types of the protocol are traversed, maximizing the protocol space; then each test case is generated based on one message type [3].

The second algorithm referred is meant for the generation of malicious strings as shown in fig.4. This method is used to produce illegal keywords by combining many tokens from two input files. The payload file is populated with already generated random data and strings. The resulting combination from both files makes the illegal word fields [3].



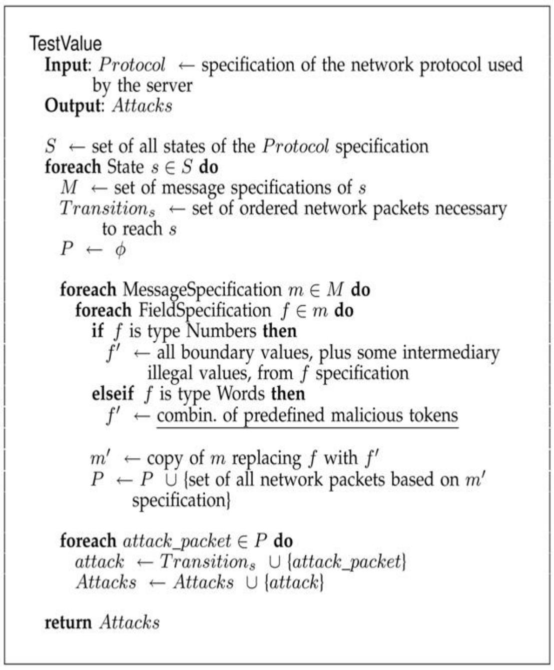
```
generateIllegalWords
  Input: $Words$ ← specification of the field
  Input: $Tokens$ ← predefined list of malicious tokens, e.g.,
      taken from hacker exploits
  Input: $Payload$ ← predefined list of special tokens to fill in
      the malicious tokens
  Input: $max\_combinations$ ← maximum number of token
      combinations
  Output: $IllegalWords$

  // step 1: expand list of tokens
  foreach $t \in Tokens$ do
  if $t$ includes keyword $(PAYLOAD) then
    foreach $p \in Payoad$ do
      $t'$ ← copy of $t$ replacing $(PAYLOAD) with $p$
      $Tokens \leftarrow Tokens \cup \{t'\}$
    $Tokens \leftarrow Tokens \setminus \{t\}$

  // step 2: generate and append all k−combinations of tokens
  $k \leftarrow 1$
  while $k \leq max\_combinations$
    $k\text{−}combinations \leftarrow \binom{Tokens}{k}$ // combinations of k elements
        from Tokens
    $IllegalWords \leftarrow IllegalWords \cup k\text{−}combinations$
    $k \leftarrow k + 1$

  return $IllegalWords$
```

Fig.3: Algorithm for generating Attack test cases



```
TestValue
  Input: $Protocol$ ← specification of the network protocol used
      by the server
  Output: $Attacks$

  $S$ ← set of all states of the $Protocol$ specification
  foreach State $s \in S$ do
    $M$ ← set of message specifications of $s$
    $Transition_s$ ← set of ordered network packets necessary
        to reach $s$
    $P \leftarrow \phi$

    foreach MessageSpecification $m \in M$ do
      foreach FieldSpecification $f \in m$ do
        if $f$ is type Numbers then
          $f'$ ← all boundary values, plus some intermediary
              illegal values, from $f$ specification
        elseif $f$ is type Words then
          $f'$ ← combin. of predefined malicious tokens

        $m'$ ← copy of $m$ replacing $f$ with $f'$
        $P \leftarrow P \cup$ {set of all network packets based on $m'$
            specification}

    foreach $attack\_packet \in P$ do
      $attack \leftarrow Transition_s \cup \{attack\_packet\}$
      $Attacks \leftarrow Attacks \cup \{attack\}$

  return $Attacks$
```

Fig. 4: Algorithm for generating malicious strings

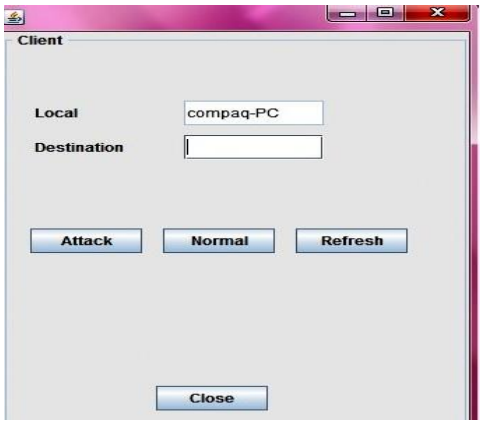
IV. Experimental Results

This section describes the environment, hardware and software requirements that have been used to design our tool NLT and to perform the server liability experiments successfully. The environment that we have worked upon is Java (JDK 1.6). The GUI for the tool has been developed using Java Swings and AWT on Windows Platform. The database used to store the attack induction results and other details is Microsoft Access 2007.

The common Intrusion Detection Applications are used to detect intrusions made by attackers. In contrast, our tool proactively finds out loopholes or liabilities in our server. This tool uses the attacks based upon the test cases generated by the facts given in protocol specifications of the target server.

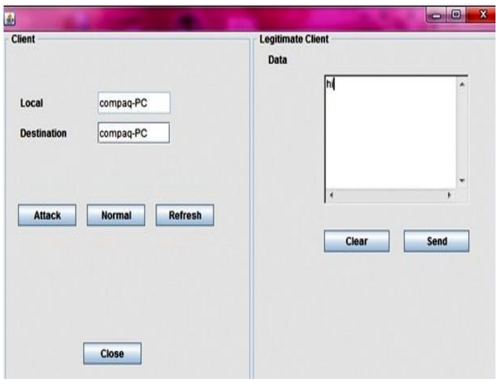
The basic GUI for the tool on client site is given in fig.5. Fig.6 shows the legitimate client sending message. Fig.7 depicts NLT attacker node sending redundant data. Fig.8 depicts how redundant data attack affects the victim server. Fig.9 depicts the warning message for the victim server. Fig.10 depicts the database at the router that stores all details. Fig.11 depicts NLT Attacker node sending bulk data, fig.12 shows database at the router that saves length, hop count and PID, fig.13 depicts attack packets blocked at victim server and fig.14 depicts NLT Victim node blocked for attack packets

Our target server was also built using Java and tested on Windows platform. It can be seen that the GUI of our tool is similar to the architecture given in figure 1 with three basic modules Attack Generator, Attack Inducer and Monitoring and Removal Phase. The empirical results reveal that our tool is successful in finding the liabilities of the target server



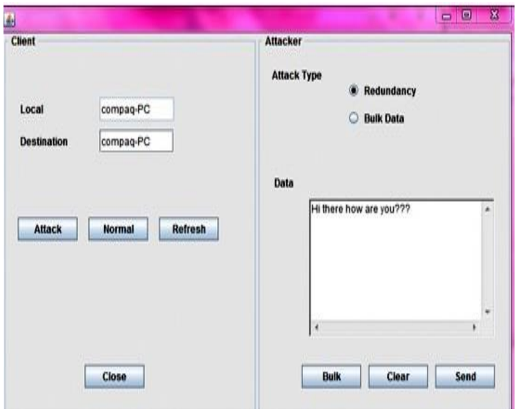
Local compaq-PC
Destination

Fig.5: GUI for Network Liability Tool on client site



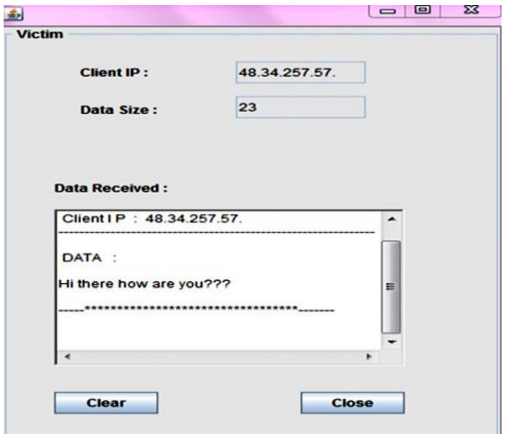
Local compaq-PC
Destination compaq-PC
Data
hi

Fig.6: Legitimate client sending message



Local compaq-PC
Destination compaq-PC
Attack Type [✓ Redundancy] Bulk Data
Data
Hi there how are you???

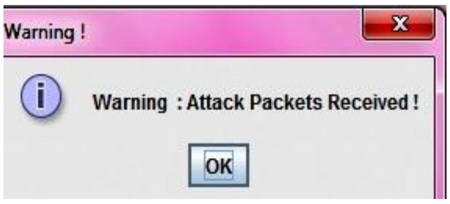
Fig.7: NLT attacker node sending redundant data



Client IP : 48.34.257.57.
Data Size : 23
Data Received :

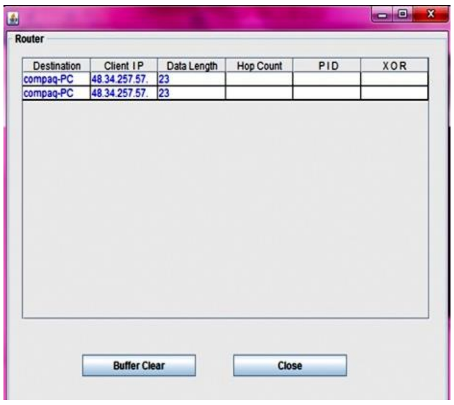
```
Client I P  :  48.34.257.57.
------------------------------------------------
DATA  :
Hi there how are you???
-----********************************-------
```

Fig.8: Redundant Data Attack on our victim server



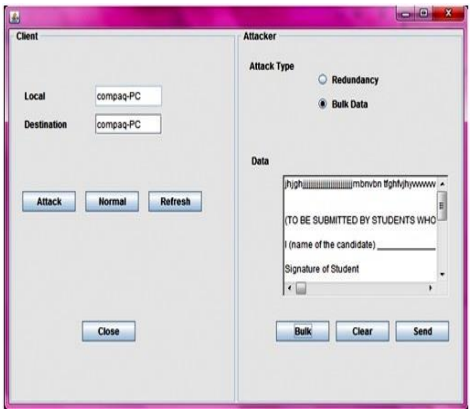
Warning : Attack Packets Received !

Fig.9: Warning message for victim server



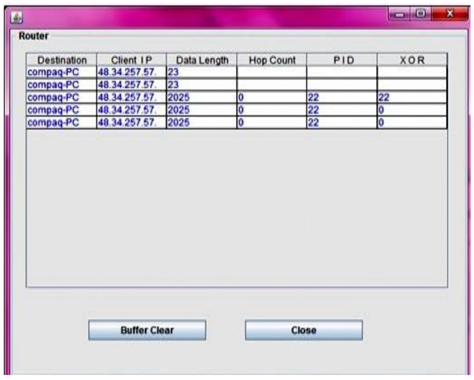
Destination	Client I P	Data Length	Hop Count	P I D	X O R
compaq-PC	48.34.257.57.	23			
compaq-PC	48.34.257.57.	23			

Fig.10: Database at the router that saves all details



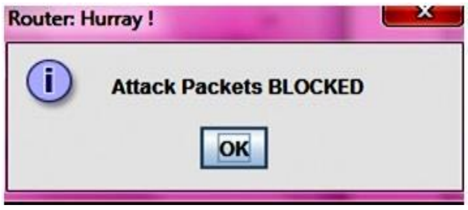
Local compaq-PC
Destination compaq-PC
Attack Type Redundancy [✓ Bulk Data]
Data
jhjghjjjjjjjjjjjjjjjjjjjjjjjmbnvbn tfghfvjhywwww
(TO BE SUBMITTED BY STUDENTS WHO
I (name of the candidate) ____________
Signature of Student

Fig.11: NLT Attacker node sending bulk data



Destination	Client I P	Data Length	Hop Count	P I D	X O R
compaq-PC	48.34.257.57.	23			
compaq-PC	48.34.257.57.	23			
compaq-PC	48.34.257.57.	2025	0	22	22
compaq-PC	48.34.257.57.	2025	0	22	0
compaq-PC	48.34.257.57.	2025	0	22	0

Fig.12: Database at the router that saves length, hop count and PID



Attack Packets BLOCKED

Fig.13: Attack packets blocked at victim server

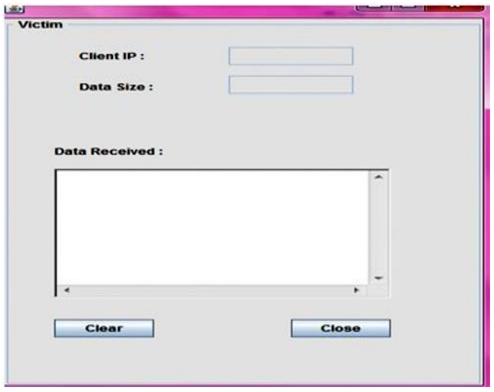


Fig.14: NLT Victim node blocked for attack packets

V. Conclusion

In this paper, we have proposed a methodology (NLD) and tool (NLT) for discovering the liabilities in the network servers.NLT launches self generated attacks such as bulk data and redundancy (as in case of DoS and worm replication) on the target server and examine its working. If any abnormal behavior is detected, this indicates the presence of liabilities, which are saved in the database such that they can be used to fix the flaws in the software in which they have been detected. The experiments are done on windows environment using Java Server to demonstrate the entire functioning of this system and helps in solving the security related loopholes in the network servers. The speculative results affirm that the proposed technique is effective in detecting the liabilities of the network servers. In future we will focus on how to eradicate these detected loopholes from our target servers. Also, we will work upon some more sophisticated attacks like spoofing, sniffing, Denial of service, etc. that can be induced in the system.

References

[1] http://in.norton.com/security_response/vulnerabilities.jsp

[2] Joa˜o Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, Rui Neves, “Vulnerability Discovery with Attack Injection”, 2009 IEEE.

[3] Joa˜o Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, Rui Neves,“Vulnerability Discovery using Attack Injection”, IEEE Transactions on Software Engineering, vol. 36, no. 3, May/June 2010.

[4] Zhiqiang Wang, Yuqing Zhang, Qixu Liu “A Research on Vulnerability Discovering for Router Protocols Based on Fuzzing”, 2012 7th International ICST Conference on Communications and Networking in China (CHINACOM), © 2012 IEEE.

[5] Sharon Goldberg, Jennifer Rexford, “Security Vulnerabilities and Solutions for Packet Sampling”, Sarnoff Symposium, 2007 IEEE.

[6] [6] A. Pasupulati, J. Coit, K. Levitt. S. F. Wu, S.H. Li, J.C. Ku0, K.P. Fan, “Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities”, 2004 IEEE.

[7] MeiJunjin, “An approach for SQL injection vulnerability detection”, 2009 Sixth International Conference on Information Technology: New Generations, IEEE.

[8] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems,” IEEE Trans. Computers, vol. 42, no. 8, pp. 913-923, Aug. 1993.

[9] M.-C. Hsueh and T.K. Tsai, “Fault Injection Techniques and Tools,” Computer, vol. 30, no. 4, pp. 75-82, Apr. 1997.

[10] J. Carreira, H. Madeira, and J.G. Silva, “Xception: Software Fault Injection and Monitoring in Processor Functional Units,” Proc. Int’l Working Conf. Dependable Computing for Critical Applications, pp. 135-149, http://citeseer.ist.psu.edu/54044.html; http:// dsg.dei.uc.pt/Papers/dcca95.ps.Z, Jan. 1995.

[11] T.K. Tsai and R.K. Iyer, “Measuring Fault Tolerance with the FTAPE Fault Injection Tool,” Proc. Int’l Conf. Modeling Techniques and Tools for Computer Performance Evaluation, pp. 26-40, http:// portal.acm.org/citation.cfm?id=746851&dl=ACM&coll= &CFID=15151515&CFTOKEN=6184618, Sept. 1995.

[12] B.P. Miller, L. Fredriksen, and B. So, “An Empirical Study of the Reliability of UNIX Utilities,” Comm. ACM, vol. 33, no. 12, pp. 32- 44, 1990.

[13] Univ. of Oulu, “PROTOS—Security Testing of Protocol Implementations,” http://www.ee.oulu.fi/research/ouspg/protos/,1999-2003.

[14] M. Sutton, “FileFuzz,” http://labs.idefense.com/labs-software.php?show=3, Sept. 2005.

[15] M. Sutton, A. Greene, and P. Amini, “Fuzzing: Brute Force Vulnerability Discovery”, Addison-Wesley, 2007.

[16] Tenable Network Security, “Nessus Vulnerability Scanner,” http://www.nessus.org, 2008.

[17] Saint Corp., “SAINT Network Vulnerability Scanner,” http:// www.saintcorporation.com, 2008.

[18] Qualys, Inc., “QualysGuard Enterprise,” http://www.qualys.com, 2008.

[19] D. Wagner, J.S. Foster, E.A. Brewer, and A. Aiken, “A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities,” Proc. Network and Distributed System Security Symp., Feb. 2000.

[20] E. Haugh and M. Bishop, “Testing C Programs for Buffer Overflow Vulnerabilities,” Proc. Symp. Networked and Distributed System Security, pp. 123-130, Feb. 2003.

[21] J. Dura˜es and H. Madeira, “A Methodology for the Automated Identification of Buffer Overflow Vulnerabilities in Executable Software without Source-Code,” Proc. Second Latin-Am. Symp. Dependable Computing, Oct. 2005.

[22] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks,” Proc. USENIX Security Conf., pp. 63-78, https:// db.usenix.org/publications/library/proceedings/sec98/cowan.html, Jan. 1998.

[23] C. Cowan, S. Beattie, J. Johansen, and P. Wagle, “PointGuard: Protecting Pointers from Buffer Overflow Vulnerabilities,” Proc. USENIX Security Symp., pp. 91-104,http://www.usenix.org/publications/library/proceedings/sec03/tech/cowan.html, Aug. 2003.

Authors' Profiles

Ulya Sabeel holds degrees M.Tech (Computer science and engineering) from Amity University, Noida, India, B.Tech (Information Technology) from Bharath University, Chennai. Currently she is working as an Assistant Professor at Amity University, Haryana, India. The areas that interest her are Network security, Adhoc and sensor networks and Ubiquitous Computing.

Saima Maqbool has completed M.Tech (Computer science and engineering) from Amity University, Noida, India, B.E. (Computer Engineering) from Kashmir University, Jammu & Kashmir. Currently she is working as an Assistant Professor at Islamic University of Science and Technology, Awantipora, Pulwama, Jammu & Kashmir, India. Her areas of interest are Network security and Wireless sensor networks.