Volume 3, Issue 9, September 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Vulnerability Discovery with Attack Injection Software Vulnerability Discovery

Dr. B. Raveendranath Singh[*]
Principal and Professor in VCET
India

***Abstract-** Software systems are prone to security vulnerabilities. Security is very important aspect of any software. In spite of security mechanisms available, there is increasing security threats revealed every year. This is a continuous problem with software systems as new threats are tried out by attackers. This paper solves this problem by developing a framework with an attack injection methodology which continuously discovers vulnerabilities in software systems and allows security administrators to resolve the issues. A prototype application is built to demonstrate the methodology. The application gets protocol details from networked server and makes attacks to discover vulnerabilities and the vulnerabilities thus discovered are persisted to a database. This helps in rectifying problems or fixing bugs in the software that causes security vulnerabilities. The empirical results revealed that, the proposed methodology is effective in locating vulnerabilities.*

Keywords:

I. Introduction

Computer usage has become part of human beings in all walks of life. It has become an essential part of any work. This does mean that reliance on computer systems is increasing dramatically each and every year. The emerging technologies, advancements in storage and security, the Internet and all such things paved way for this reliable usage of computers in all businesses and personal pursuits. The software development processes have been around ever since man started using systems to develop software. However, every software built by humans based on some systematic approach known as software process model, may have bugs hidden. With respect to security, the bugs are known as vulnerabilities. These vulnerabilities are exploited by hackers and adversaries to perform security attacks for monetary or other gains. This is the reason for this research paper. This paper aims at developing a new methodology that helps in discovering security vulnerabilities of networked servers. This paper also presents a Vulnerability Prediction Tool that is meant for generating attacks on target server based on the protocol specifications of target server. Once the vulnerabilities are discovered, they are handed over to the team involved in the development of target server. The software development team follows traditional debugging and other techniques to spot the vulnerabilities and rectify them.

The proposed methodology and vulnerability detection tool are meant for proactively making attacks on target server, in this case it is Red Hat Linux, with a good intention to make a list of vulnerabilities and let the development and security personnel of the target server to identify and fix the problems so as to avoid the security threats from real adversaries who make attacks for monetary or other gains. While making attacks the expected behavior and abnormal behaviors of server against various attacks are observed. The expected behavior does mean that there is no security vulnerability exists with respect to the attack made based on the test definition which is driven by protocol specification. Any behavior of server that deviates from expected one confirms the presence of vulnerability. These results are valuable to the team that takes care of security of the target server. The nice thing about the proposed vulnerability detection tool is that, it does not need the source code of the target server in order to make attacks on it. Instead it treats server OS as block box and perform attacks. The tool simply needs protocol specifications of target server that give enough information to derive test definitions, generate attacks and inject them to complete the experiments and discover vulnerabilities in the target server. The tool follows certain phases such as generate attacks, inject attacks and look for errors or failures. The generate attacks phase takes protocol specifications as input and generates a set of test definitions. In turn these test definitions are used to generate attacks. The inject attacks phase is actually injecting attacks generated by generate attacks phase. Once injecting attacks is completed, it results in discovery of vulnerabilities. The following sections throw light into review of literature, proposed methodology and tool creation details.

II . Related Work

This paper deals with vulnerability discovery in networked servers. This work has been influenced by previous researches that are reviewed in this section. Broadly the related work of this paper can be classified into fault injection, fuzzers, vulnerability scanners and static vulnerability analyzers, runtime prevention mechanisms and software rejuvenation. Fault injection is an experiment that is meant for injecting faults intentionally and estimating the handling mechanism built into a software system. As discussed in [2] and [3] error latency and fault coverage characterize the fault injection mechanisms. Fault injection mechanisms are traditionally used to discover software and hardware bugs in the IT systems. Such bugs may range to simple memory errors to complex ones. The tools Xception and FTAPE as described in [4] and [5] respectively can effectively inject hardware and software faults into the target system that is under evaluation. This enables the users of the tools to find out and fix bugs in the system at hardware and software levels. In this paper, the proposed tool by name Vulnerability Detection Tool is meant for injecting attacks into a network server and discovers its vulnerabilities. This is something different from fault injections as it can inject more complex faults and also monitor vulnerabilities more effectively. This tool really emulates attack as if it is made by an attacker or hacker in the real world. As reviewed in [6] the origin of vulnerabilities might from POSIX APIs and interfaces related to device drivers. The simple fault injectors are not sufficient to discover faults in networked servers. This is the reason why the proposed tool is built for making sophisticated fault injections into Red Hot Linux server and discovers its vulnerabilities.

Another way of discovering faults in software systems is by using fuzzers. The fuzzers are used to inject random samples interactively into software components. However, they are not as good as fault injection mechanisms. Fuzz [7] scrambled command line characters and thus spoiled the execution of an application. It generates random characters in large quantities. When tested with fuzzers, many programs failed or crashed that indicates dangerous flaws in the software system. As discussed in [8] by running millions of iterations it was possible to bring about many security flaws hidden in the system. Fuzzers need to know the software behavior before hand. Instead they test the behavior of systems randomly. They lack in mechanisms for monitoring when compared with the proposed tool in this paper. The proposed tool is independent of target server and its underlying protocols. However, it needs the protocol specification knowledge and it gets it on the fly automatically. That way it is not dependent on the target servers.

Vulnerability scanners are the tools for discovering vulnerabilities in software systems. Most of the time these tools discover vulnerabilities in network machines. In the literature there are many such tools reviewed. They include Qualys-Guard [9], SAINT [10] and Nessus [11]. These tools maintain databases that contain known vulnerability details and a collection of attacks in order to discover the vulnerabilities. Their approach is that they get information about the execution environment of the target system and that is correlated with the information available in the database of these tools in order to determine security vulnerabilities. After knowing the vulnerabilities, these tools make corresponding attacks on the vulnerabilities and generate statistics about successful attacks. That information is very useful to find and fix bugs in the software system that has been evaluated. The drawback of these tools is that they can discover only known vulnerabilities.

Static vulnerability analyzers are other kinds of tools that analyze the source code or binary code of the applications under evaluation. When compared with attack injection, they follow different approach in which source code is analyzed for vulnerabilities [12], [13]. Once the detection of vulnerabilities is made, the programmers can examine only the parts in which code is suspected. This idea in [14] is extended where the binary code is analyzed. Race conditions and other such errors are also fixed using static analysis. As per the literature available these tools are very effective in locating such bugs or problems in the software systems. However, they are having limitations as they produce many false positives in the process of detecting vulnerabilities.

Runtime prevention mechanisms were also used in the past for discovering vulnerabilities in software systems. These mechanisms change the environment at runtime in order to exploit the presence of vulnerabilities. This assumption is that it is not feasible to remove all bugs from a system. It does mean that it is preferable to have damages caused by these explorations. Most of the systems that follow this mechanism target the errors like buffer overflows. Fes such examples include PointGaurd [15] and StackGuard [16]. These tools are static analyzers and they are compiler based. However, at runtime in case of buffer overflow, the program execution gets stopped before actual attack code is executed. As per [17] these tools are capable of preventing only a subset of attacks. A tool introduced in Windows XP service pack 2 is known as DEP (Data Execution Prevention) that prevents an executable code to run in a non executable region [18]. In [19] pointer prediction approach has been discussed as PaX provide program memory randomization that helps in thwarting attacks. Libsafe tool follows another approach that intercepts library calls that prevent hijacking of the main control flow of an application under evaluation [20]. Bad resource management problem is addressed by many techniques that lead to software aging. Software rejuvenation is an approach used to reduce the effects of software aging [21]. This technique is used to effectively remove effects of software aging. The problem with software rejuvenation is that even after rectifying the problem the software aging problem may occur again.

III. Proposed Methodology

We propose an attack injection methodology that discovers vulnerabilities in select networked servers. Networked servers have inherent and latent vulnerabilities that are not unearthed unless some attacks are made. When attacks are made by adversaries, the network servers fail to perform their functionalities. The security attacks made by hackers cause the IT systems to lose important sensitive information. It is essential to test the servers to find whether there are security vulnerabilities. The fig. 1 shows architecture of proposed methodology.

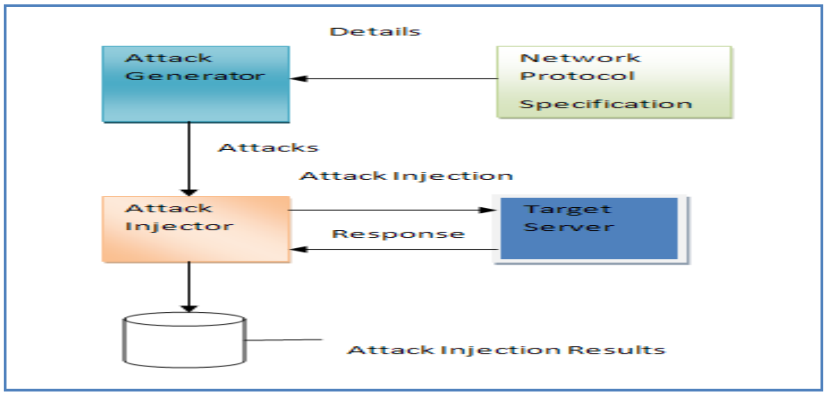
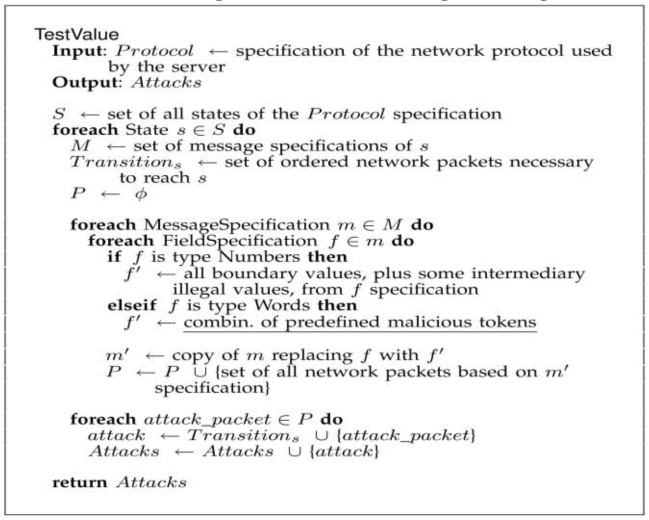


Fig. 1 – Architecture of Proposed Methodology

As can be seen in fig. 1, the architecture shows the methodology used in discovering vulnerabilities in the target servers. Networked servers are of many types such as Windows, Linux, and Apple Mac and so on. Before making attacks it is required to know the protocols used by the target server. For this reason network server protocol specifications are used. From the protocol specifications, the proposed system takes details of a particular protocol. The extracted protocol specification details are kept in a database for further usage. Based on the specification, various kinds of tests are defined. The tests thus defined are used in the proposed framework. The attack generator component of the proposed architecture makes use of the defined tests and generates various attack instances. Such attacks are saved to a database for further use. These attacks are used by attack injector which is responsible for actually injecting attacks into target server and get the responses. The attack injection result may be positive or negative. Positive does mean that attack is made successfully and certain vulnerability has been discovered. Negative does mean that attack was not successful and that indicates robustness of target server in that area. The attack injector component various sub components for the purpose of processing attacks, injecting packets, and collecting response and execution data. The results of the attack injection process are saved to database known as attack injection results database. Target server is the component that existed in the real world. It is a networked server with security in place. However, there might be unknown security vulnerabilities that are discovered by the proposed framework.

IV. Algorithms Used

The algorithms proposed in [1] are used for the experiments. They are given in this paper. The first algorithm is meant for generating attacks while the second algorithm is meant for generating of malicious strings.



```
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 $s$
    $P$ ← $\phi$

    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 $f'$
        $P$ ← $P$ ∪ {set of all network packets based on $m'$
            specification}

    foreach $attack\_packet \in P$ do
      $attack$ ← $Transition_s$ ∪ {$attack\_packet$}
      $Attacks$ ← $Attacks$ ∪ {$attack$}

  return $Attacks$
```

Fig. 2 – Algorithm for generating of attacks

This algorithm traverses all message types and states of protocol in order to maximize the attack space. Based on the message type each and every test case is generated. This algorithm is different from others as it populates each field with wrong values systematically instead of using only legal values. Overall, this algorithm takes specification of network server protocol and generates various security attacks to discover vulnerabilities in the specified server.

```
generateIllegalWords
  Input: $Words$ $\leftarrow$ specification of the field
  Input: $Tokens$ $\leftarrow$ predefined list of malicious tokens, e.g.,
      taken from hacker exploits
  Input: $Payload$ $\leftarrow$ predefined list of special tokens to fill in
      the malicious tokens
  Input: $max\_combinations$ $\leftarrow$ 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
      $t'$ $\leftarrow$ 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 $k \leq max\_combinations$
    $k{-}combinations \leftarrow \binom{Tokens}{k}$ // combinations of k elements
        from Tokens
    $IllegalWords \leftarrow IllegalWords \cup k{-}combinations$
    $k \leftarrow k + 1$

  return $IllegalWords$
```

Fig. 3 – Algorithm for generating malicious strings

As can be viewed in fig. 3, this algorithm takes specification of the field, list of malicious tokens, predefined list of special tokens and maximum number of token combinations as input and generates malicious strings as output. This output is used in discovering vulnerabilities in the target server.

V. Experiments

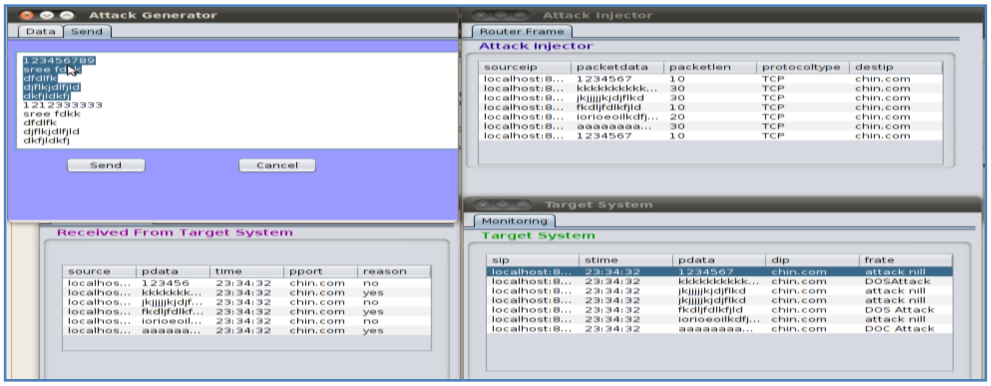
This section describes the environment used for conducting experiments and the way experiments are conducted.

Environment

The environment used to build a prototype application for the purpose of making experiments include JDK 1.6 (JSE 6.0), Net Beans IDE, Oracle 10G Express Edition, a PC with Linux Server running. No special hardware or software is required apart from this.

Prototype Application

An application is built with GUI (Graphical User Interface) that facilitates to achieve the goal of this paper. The aim of this paper is to make attacks on networked server with the intention to discover vulnerabilities and let security personnel to fix the discovered vulnerabilities. General IDS (Intrusion Detection System) applications are meant for detecting intrusions made by adversaries. However, the proposed tool is to make attacks proactively and discover vulnerabilities. The attacks are devised based on the knowledge gained from the protocol specifications of target



```
123456789
sree fdkk
dfdlfk
djflkjdlfjld
dkfjldkfj
1212333333
sree fdkk
dfdlfk
djflkjdlfjld
dkfjldkfj
```

sourceip	packetdata	packetlen	protocoltype	destip
localhost:8...	1234567	10	TCP	chin.com
localhost:8...	kkkkkkkkkk...	30	TCP	chin.com
localhost:8...	jkjjjjjkjdjflkd	30	TCP	chin.com
localhost:8...	fkdljfdlkfjld	10	TCP	chin.com
localhost:8...	iorioeoilkdfj...	20	TCP	chin.com
localhost:8...	aaaaaaaa...	30	TCP	chin.com
localhost:8...	1234567	10	TCP	chin.com

sip	stime	pdata	dip	frate
localhost:8...	23:34:32	1234567	chin.com	attack nill
localhost:8...	23:34:32	kkkkkkkkkk...	chin.com	DOSAttack
localhost:8...	23:34:32	jkjjjjjkjdjflkd	chin.com	attack nill
localhost:8...	23:34:32	jkjjjjjkjdjflkd	chin.com	attack nill
localhost:8...	23:34:32	fkdljfdlkfjld	chin.com	DOS Attack
localhost:8...	23:34:32	iorioeoilkdfj...	chin.com	attack nill
localhost:8...	23:34:32	aaaaaaaa...	chin.com	DOC Attack

source	pdata	time	pport	reason
localhos...	123456	23:34:32	chin.com	no
localhos...	kkkkkkk...	23:34:32	chin.com	yes
localhos...	jkjjjjjkjdjf...	23:34:32	chin.com	no
localhos...	fkdljfdlkf...	23:34:32	chin.com	yes
localhos...	iorioeoil...	23:34:32	chin.com	no
localhos...	aaaaaa...	23:34:32	chin.com	yes

server. The vulnerability detection tool or the application developed in this paper ensures that attacks are made based on test definitions that are driven by the facts given in the protocol specifications of target server.

As can be seen in fig. 4, the tool is built in Linux environment using Java programming language. Java's SWING API is used to build graphical interface while the network API is used for communication with target server. IO is used to perform read and write operations on local file system while JDBC (Java Database Connectivity) is used to perform database operations. The application has two important components or modules. They are known as attack generator and attack injector. The attack generator module is used to define various tests meant for generating attacks. These tests are based on the protocol specifications collected from target server. In our experiments we used Red Hot Linux server as target server. The vulnerability discovery tool also tested in Linux environment. The procedure followed by the tool is very much similar to the description given about the architecture of proposed methodology as shown in fig. 1. The attack generator module is responsible to make use of tests defined and generate possible attacks. It does mean that an attack is generated for each test definition. The collection of attacks that generated is persisted to database. These attacks are used by the attack injector module. The attack injector module is responsible to actually inject attacks into target server. It monitors the attacking process and also results there of. The attack results are saved to database as attack injection results. The results of attacks are then used by security personnel of target server to focus on the vulnerabilities and take steps to ensure that those vulnerabilities are rectified and they no longer give chance to security threats in future. As can be seen in the results shown in fig. 4, it is evident that there are some vulnerabilities discovered in case of Red Hot Linux server which is freely available for download over Internet.

VI. Conclusion

This paper presents a methodology that makes attacks on networked servers and discovers security vulnerabilities. To demonstrate this prototype application is built that takes protocol specification details from server and performs various attacks on the server and discovers vulnerabilities. The discovered vulnerabilities are then persisted into database for further steps to fix the bugs in the software in which vulnerabilities are found. The experiments are made on Linux server and the empirical results revealed that the proposed tool is capable of discovering vulnerabilities effectively.

References

[1] Joa˜o Antunes, Student Member, IEEE, Nuno Neves, Member, IEEE, Miguel Correia, Member, IEEE, Paulo Verissimo, Fellow, IEEE, and Rui Neves. Vulnerability Discovery with Attack Injection. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 36, NO. XX, XXXXXXX 2010

[2] 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.

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

[4] 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.

[5] 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.

[6] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems," Proc. Int'l Symp. Fault-Tolerant Computing, pp. 30-37, June 1999.

[7] 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.

[8] P. Oehlert, "Violating Assumptions with Fuzzing," IEEE Security and Privacy, vol. 3, no. 2, pp. 58-62, http://ieeexplore.ieee.org/ xpls/abs_all.jsp?arnumber=1423963, Mar./Apr. 2005.

[9] Qualys, Inc., "QualysGuard Enterprise," http://www.qualys. com, 2008.

[10] Saint Corp., "SAINT Network Vulnerability Scanner," http://www.saintcorporation.com, 2008.

[11] Tenable Network Security, "Nessus Vulnerability Scanner," http://www.nessus.org, 2008.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] J. Wilander and M. Kamkar, "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention," Proc. Network and Distributed System Security Symp., pp. 149-162, Feb. 2003.

[18] Microsoft, Corp., "A Detailed Description of the Data Execution Prevention (DEP) Feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003," http://support.microsoft.com/kb/875352, Sept. 2006.

[19] "PaX," http://pax.grsecurity.net/, 2009.

[20] T. Tsai and N. Singh, "Libsafe 2.0: Detection of Format String Vulnerability Exploits," white paper, Avaya Labs, 2001.

[21] K. Vaidyanathan and K.S. Trivedi, "A Comprehensive Model for Software Rejuvenation," IEEE Trans. Dependable and Secure Computing, vol. 2, no. 2, pp. 124-137, Apr.-June 2005.