On Shaky Ground - A Study of Security Vulnerabilities in Control Protocols

Eric J. Byres, Dan Hoffman & Nate Kube

Byres Security Inc.: 7178 Lancrest Tr., PO Box 178, Lantzville, BC, Canada, V0R 2H0, eric@ByresSecurity.com
Wurldtech Analytics Inc.: 208-1040 Hamilton St., Vancouver, BC, Canada, V6B 2R9, nkube@wurldtech.com
University of Victoria: Department of Computer Science, PO Box 3055 STN CSC, Victoria, BC, Canada, V8W 3P6, dhoffman@cs.uvic.ca

Abstract – *The recent introduction of information technologies such as Ethernet® into nuclear industry control devices has resulted in significantly less isolation from the outside world. This raises the question of whether these systems could be attacked by malware, network hackers or professional criminals to cause disruption to critical operations in a manner similar to the impacts now felt in the business world.*

To help answer this question, a study was undertaken to test a representative control protocol to determine if it had vulnerabilities that could be exploited. A framework was created in which a test could express a large number of test cases in very compact formal language. This in turn, allowed for the economical automation of both the generation of selectively malformed protocol traffic and the measurement of device under test's (DUT) behavior in response to this traffic.

Approximately 5000 protocol conformance tests were run against two major brands of industrial controller. More than 60 categories of errors were discovered, the majority of which were in the form of incorrect error responses to malformed traffic. Several malformed packets however, caused the device to respond or communicate in inappropriate ways. These would be relatively simple for an attacker to inject into a system and could result in the plant operator losing complete view or control of the control device. Based on this relatively small set of devices, we believe that the nuclear industry urgently needs to adopt better security robustness testing of control devices as standard practice.

I. INTRODUCTION

The recent introduction of information technologies such as Windows®, Ethernet® and TCP/IP in nuclear industry control devices has resulted in significantly less isolation from the outside world. Both anecdotal evidence [1] [2] and research [3] indicates that SCADA protocols, particularly those running over top of transport protocols such as TCP/IP, have vulnerabilities that could be exploited by network hackers or terrorists to cause considerable disruption to our critical infrastructures. Little is known about these vulnerabilities and there are limited security tools or methodologies available for vendors or users to detect these flaws prior to equipment deployment.

As highly integrated control systems are relatively new, there is shockingly little data, good or bad, on network security for these industrial devices. The current methodologies for security testing focus on business systems and their dependence on common operating system such as Windows and UNIX. Similarly, vulnerability reporting such as CERT or BugTraq primarily addresses IT products and rarely includes issues with industrial control products. In order to determine the security robustness of integrated control systems new testing methodologies are required.

This paper describes a new test framework which enables the economical creation of security suites targeted at testing the dominant SCADA application layer protocols.

To demonstrate the effectiveness of the framework we employed it to generate a test suite for the MODBUS/TCP protocol. We then exercised the resulting test suite against the MODBUS/TCP implementations on two representative SCADA devices. The results were unsettling.

II. ORGANIZATION OF THIS PAPER

This paper is designed to introduce and support the need for new and efficient tools to test the network security robustness of industrial control devices. We begin by briefly presenting some background information on protocol testing and the tools which are available. We then introduce blackPeer, an innovative testing framework for communication protocols. We then discuss the employment of blackPeer in conformance testing MODBUS/TCP implementations. We conclude with a summary of the MODBUS/TCP implementation errors found by blackPeer in two representative SCADA devices and some observations on how the SCADA community can better ensure the security of its control systems.

III. BACKGROUND AND RELATED WORK

Many communication protocols are highly complex and their implementations may be written to a specification that contains areas of ambiguity. Experience tells us that incorrect assumptions or carelessness of the implementer are common sources of protocol vulnerabilities. Protocol vulnerabilities can reveal themselves as segmentation faults, stack, heap or buffer overflows, etc., all of which can cause the protocol implementation to fail resulting in a potential exploit.

Tools to scan for known vulnerabilities in traditional IT systems have been available for at least a decade. The market for these vulnerability scanners has been significant and products such as Nessus, FoundScan and Internet Security Scanner (ISS) have been popular with IT administrators trying to locate unpatched computers on their networks. Unfortunately these tools offer little in the way of security testing for new products with new vulnerabilities - they only check for known vulnerabilities available in vulnerability lists such as CERT or BugTraq. As a result, most vendors have little knowledge of possible vulnerabilities in new systems until after the product is released to the public.

This is particularly true for the SCADA systems used in critical infrastructures such as the nuclear, oil and gas, water and electrical generation/distribution industries. The embedded devices used in these systems are not the usual Windows or UNIX-based platforms and the vulnerabilities are not available in the IT-focused CERT or BugTraq vulnerability lists. As a result, SCADA operators have little knowledge of possible vulnerabilities in their critical systems until a disaster (such as the August 2003 blackout in the Eastern U.S.) strikes.

In the academic world there have been several test tools that have had success in locating new vulnerabilities in network devices based on grammar and fuzzy techniques. Considerable work has been done by the PROTOS project group [4] and by Tal, Knight and Dean [5]. Each considers the syntax-based generation of *protocol data units* (PDU's). A PDU translates into a single test packet to be sent to the *device under test* (DUT). Their methods have proven effective in finding vulnerabilities [4][5] however, they only allow for the construction of simple single-packet test cases. Such test cases are not sufficient for testing protocol functionality requiring interaction between the test case generator and the DUT. To perform such testing the test case generator must be able to generate test cases consisting of semantically meaningful sequences of PDU's.

A sequence of PDU's in which each PDU is preceded by its communication direction relative to the test case generator is termed a *protocol test sequence* (PTS). PTS's can direct test case executors to not only transmit certain PDU's but also to compare received packets against indicated PDU's. Such ability greatly enhances the effectiveness of fuzzer-based methods.

Furthermore, as highly integrated control systems typically consist of many different devices and because these devices may contain implementations of many different protocols, a truly valuable protocol vulnerability testing tool must be easily applicable to a wide variety of protocols. As well, a valued tool must be employable by users with varying skill sets. For example, the tool should be employable by the vendor, by a field engineer or by a plant floor worker. To the best of our knowledge, no such tool exists.

IV. THE BLACKPEER TEST FRAMEWORK

The automated generation of test data for the purposes of software validation and verification has been a major aim of testing research[6]. Path-oriented methods, data flow methods, random testing, adaptive testing, and syntax-based testing are a few of the methods employed for this task.

Syntax-based testing tools process a description of the desired test data expressed in notation such as *Backus Naur Form* (BNF). Given the test data syntax these tools generate test sets which satisfy the syntax. A major problem in syntax-based testing is respecting contextual dependencies. Solutions include the use of *dynamic syntax*, addition of rules to test data syntax during test data generation, and *attributed grammars*, grammars whose definition is overloaded with attributes.

In attribute grammars, the attributes represent contextual information associated with terminals and non-terminals. Attribute grammars can solve the test oracle problem as the contextual information can allow for the production of test data along with the output expected from the DUT. Hence, attribute grammars can also solve the problem of generating semantically meaningful sequences of PDU's, i.e. PTS's.

blackPeer is an attribute grammar based PTS generator and executor. The basic file structure of blackPeer is similar of that employed by Sirer and Bershad [7] and is shown in Figure 1. PTS's are described by an attributed grammar. This grammar is passed as input to the *Code Generator*. The Code Generator pareses the grammar and creates an executable program called the *TestCase Generator*. When supplied with an initialization file called the *variable init* file, the TestCase Generator writes out the PTS's encoded by the grammar to a file called the *testCases* file. The *TestCase Executor* then reads in the testCases file along with basic information about the DUT and executes the PTS's one by one, reporting and recording the results.

The modular nature of the blackPeer tool allows different levels of functionality to be made available to different categories of users. For example, a plant worker who knows little of communication protocols but much

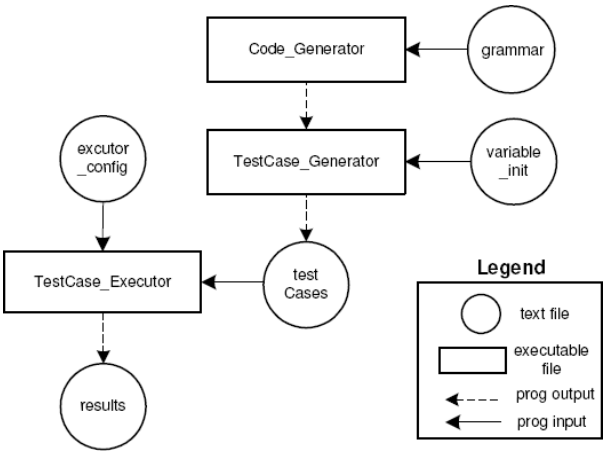

Fig. 1. blackPeer PTS generation and execution.

about PLC's may be given a testCases file and an appropriate testCase Executor. An industrial engineer on the other hand may be given the testCase Generator thus allowing him to create a variety of testCases files depending on the nature of the variable init file he creates.

This modular nature also allows for the easy transition between different protocols. That is, to test a different protocol all that is needed is a new grammar file and an appropriate testCase Executor.

All grammar files are written using an extended Backus Naur Form (BNF) syntax:

- non-terminals specified as identifiers and terminals as string constants
- alternatives specified by a vertical bar and concatenation by one or more spaces
- the required portion of a production is terminated by a newline character
- the right hand side expression can contain parenthesis one level deep for grouping logical expressions, square brackets for denoting optional expressions, and braces to delimit expressions that can occur zero or more times
- an optional range after the opening brace allows repetition a fixed number of times
- comments are started by a // and terminated by the end of the line
- the start symbol is the non-terminal on the left hand side of the first production

The syntax for a grammar is as follows:

```
grammar ::=
      rule {rule} "\n"
rule ::=
      nonterminal "::=" rhs "\n"
rhs ::=
      ( nonterms | terms ) "\n" [pre] [post] "\n"
nonterms ::=
      "(" nonterms ")" | nonterminal {("|" | "
      ")nonterminal}
terms ::=
      "(" terms ")" | terminal {("|" | " ")terminal}
pre ::=
      "%pre{\n" program_statements "\n}%"
post ::=
      "%post{\n" program_statements "\n}%"
nonterminal ::=
      letters_and_or_digits
terminal ::=
      """ value ":" value """
value ::=
      ["$"] letters_and_or_digits
```

The syntax for a test case is as follows:

```
testcase ::=
       tx {tx|rx} "\n"
tx ::=
      "TX:0 " digits ":" digits" " {digits ":" digits " "}
rx ::=
      "RX:0 " ("*"|digits) ":" digits " " {("*"|digits)
      ":" digits " "}
```

V. CONFORMANCE TESTING MODBUS/TCP

The blackPeer framework is well-suited to conformance testing. Given a protocol's specification, one can easily construct a grammar defining the correct operation of an implementation to this specification. An excerpt of the grammar written for the MODBUS/TCP protocol is shown in Example I. In conjunction with the creation of a grammar comes the definition of the variability found across implementations. An example of one such variability is the number of holding registers present on the DUT. Such variability's are defined in a separate variable initialization file and are passed as input to the test case generator, see Figure 1. An Excerpt of the variable file for the MODBUS/TCP grammar is shown in Example II.

EXAMPLE I. Excerpt from MODBUS grammar file

```
REQ ::= TX HDR FC DATA
%post{
int i = Utilily.getVariableValue("currLen");
int tot = Utility.sumOfLengths(tokens)-6 ;

if(i==-1)//sub with real length
      Utility.setRuleResolve(tokens[1],2, tot, 2 );

else if( tot != i )
      Utility.setVariableValue("exceptCode",4);
}%
```

EXAMPLE II. Excerpt from variable init file

```
rQuan = {0,1,$rhmaxconsec-1,$rhmaxconsec,
    $rhmaxconsec+1, $cmaxconsec-1, $cmaxconsec,
    $cmaxconsec+1}

wAddr={0,1,$numIReg-1,$numIReg,$numDisc-
    1,$numDisc,$numCoils-1,$numCoils,
    $numHReg-1,$numHReg,3000}
```

V.A. Test Environment

The basic test bench consists of (i) the DUT (in this case the PLC whose MODBUS/TCP implementation we wish tested), and (ii) the platform computer running blackPeer.

The network configuration is illustrated in figure 2.

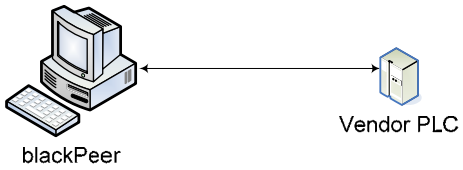

Fig 2. The hardware test bench

We selected two representative SCADA devices for test. We will refer to them as PLC M and PLC Q.

PLC Q is a modular backplane PLC. At its most basic level it consists of a variety of input/output (I/O), processor and communications modules that are installed in a common backplane. The central processor unit (CPU) module is separate from the other modules. The Ethernet Communications module provides an interface between an Ethernet-based network, the backplane, and the CPU module. It supports the MODBUS protocol over TCP.

PLC M is a small versatile PLC. It is used for PC-based control, distributed control, distributed I/O and traditional, stand-alone PLC Control. It employs Ethernet as its primary external communications protocol and supports MODBUS/TCP.

VI. RESULTS

We start with grammars covering read and write function codes. Then we examine if the device supports the function codes that are designated for serial only. We finish off with a grammar that examines miscellaneous and invalid function codes.

VI.A. MODBUS/TCP Read Grammar Test

The MODBUS/TCP Read grammar examines the DUT's behavior in response to valid and invalid read requests. The function codes tested are:

- 01 Read Coils
- 02 Read Discrete Inputs
- 03 Read Holding Registers
- 04 Read Input Registers

TABLE I. Read Grammar Test Observed Behavior

	PLC M	PLC Q
Testcases	352	352
Passed	231	298
Failed	121	54

We now break down the failed tests by function code.

VI.A.I. Function Codes 01 and 02

PLC Q and PLC M returned incorrect error codes when:

1. the starting address and quantity of outputs was valid but the starting address + quantity of outputs was out of range
2. the starting address was invalid but the quantity of outputs was within range

VI.A.II. Function Code 03

PLC Q conformed to function code 03's specification whereas PLC M returned incorrect error codes under the following circumstances:

1. when the starting address was valid but the quantity of registers to read was set to zero
2. when the starting address was valid but the quantity of registers was out of range
3. when the starting address was invalid and quantity of registers was 0 or out of range

VI.A.III. Function Code 04

PLC Q returned no error while PLC M returned incorrect error codes under the following erroneous circumstances:

1. when the starting address and the quantity of registers was valid but the quantity of registers + starting address was out of range
2. when the starting address was invalid but the quantity of registers was within range

VI.B. MODBUS/TCP Write Grammar Test

The MODBUS/TCP Write grammar examines the DUT's behavior in response to valid and invalid write requests. The function codes tested are:

- 05 Write Single Coil
- 06 Write Single Register
- 15 Write Multiple Coils
- 16 Write Multiple Registers

TABLE II. Write Grammar Test Observed Behavior

	PLC M	PLC Q
Testcases	1384	1384
Passed	504	1040
Failed	880	344

We now break down the failed tests by function code.

VI.B.I. Function Code 15

Both PLC Q and PLC M returned incorrect error codes under the following circumstances:

1. when the starting address, the quantity of outputs and the data was valid but the starting address + quantity of output was out of range
2. when the starting address and the quantity of outputs was valid but the starting address + quantity of outputs was out of range and the data byte length was not equal to the actual length of the data
3. when the starting address was invalid and the quantity of outputs was 0 and data length was correct or incorrect
4. when the starting address was invalid and the quantity of outputs was non zero and the data length was incorrect

VI.B.II. Function Code 16

PLC Q incorrectly reported an error when:

1. the starting address was valid and the starting address + quantity of registers was within range and the quantity of registers was 122 or 123 (limit is 123)

PLC Q returned incorrect error codes under the following circumstances:

1. when the starting address was valid and the quantity of registers was 122 or 123 and the starting address + quantity of registers was out of range and the data was valid

PLC M returned incorrect error codes under the following circumstances:

1. when the starting address was valid but the quantity of registers was zero
2. when the starting address and quantity of registers was valid and the starting address + quantity was with in range but the actual data length and specified data length were not in agreement
3. when the starting address was valid and the quantity of registers was invalid and the staring address + quantity of registers was within range
4. when the starting address was valid and the quantity of registers was invalid and the actual data length and the specified data length were or were not in agreement

VI.C. MODBUS/TCP Serial Grammar Test

The MODBUS/TCP Serial grammar examines the DUT's behavior in response to valid and invalid serial requests. The function codes tested are:

- 07 Read Exception Status
- 08 Diagnostics
- 11 Get Comm Event Counter
- 12 Get Event Log
- 17 Report Slave ID

TABLE III. Serial Grammar Test Observed Behavior

	PLC M	PLC Q
Testcases	28	28
Passed	0	11
Failed	28	17

We now break down the failed tests by function code.

VI.C.I. Function Codes 07, 11, 12 and 17

Both PLC Q and PLC M responded to function code 07, 11, 12 and 17 requests even though they are serial line only functions. The correct response would have been to return an error code of 1 indicating function not supported.

VI.C.II. Function Code 08

Under all circumstances PLC M responded to function code 08 requests by returning an error code of 3 (indicating an incorrect data value). The correct response would have been to return an error code of 1 indicating function not supported.

PLC Q returned incorrect error codes in response to function code 08 sub functions 3, 14, 15 and 20. More disturbing, PLC Q carried out the requests of function code 08 sub functions 0, 1, 2, 4, 10, 11, 12, 13, 16, 17 and 18.

Function code 08 sub function 4 forces a PLC into listen mode. It will not leave listen mode until it receives a "restart communications option" (function code 8 sub function 1). However, since MODBUS is being executed over TCP PLC Q must first engage in a TCP three-way handshake prior to receiving a "restart communications command". As PLC Q is in listen mode it will not engage in the 3-way handshake, hence PLC Q must be power-cycled before it will come back online.

This brings to light an interesting point. Function 08 sub function 4 was created to allow the engineer to isolate a network flooding PLC. This is especially useful in the case of older low speed wireless networks where the

underlying transport medium was easily saturated. However, when executing MODBUS over TCP such saturation is easily mitigated by TCP's back-off and retransmit strategies.

Furthermore, as demonstrated above, executing function code 08 sub function 4 over a TCP connection is dangerous. It was intended to be executed over a stateless medium, one in which the special "reset communications" packet could be received at any time and not over a medium in which a stateful communications startup is required.

This is indeed a good example of the problems that one faces when transmitting one protocol inside another. Investigating each protocol individually does not shed light on the problems that may arise when the protocols become coupled in the way that MODBUS and TCP have been.

VI.D. MODBUS/TCP Miscellaneous Grammar Test

The MODBUS/TCP Miscellaneous grammar examines the DUT's behavior in response to valid and invalid miscellaneous function code requests. It also examines the DUT's behavior in response to invalid function code requests. The valid function codes tested are:

- 20 Read File Record
- 21 Write File Record
- 22 Mask Write Holding Registers
- 23 Read / Write Multiple Registers
- 24 Read FIFO Queue
- Fuzzed Headers

TABLE IV. Serial Grammar Test Observed Behavior

	PLC M	PLC Q
Testcases	3433	3433
Passed	531	2171
Failed	2902	1262

We now break down the failed tests by function code.

VI.D.I. Function Code 20

Both PLC Q and PLC M returned incorrect error codes under the following circumstances:

1. when the byte count < 7 or greater than 245
2. when the reference type !=6
3. when the record number >10000 or the record number + register length >10000

VI.D.II. Function Code 21

Both PLC Q and PLC M returned incorrect error codes under the following circumstances:

1. when the byte count < 7 or greater than 245
2. when the reference type !=6
3. when the record number >10000 or the record number + register length >10000

VI.D.III. Function Code 23

PLC Q incorrectly reports an error when:

1. the starting write address was valid and the starting write address + quantity of registers to write was within range and the quantity of registers was 120 or 121 (limit is 121)

PLC Q returned incorrect error codes under the following circumstances:

1. when the starting write address was valid and the quantity of registers to write was 120 or 121 and the starting address + quantity of registers to write was out of range and the data was valid

PLC M returned incorrect error codes under the following circumstances:

1. when the read starting address was valid but the quantity of registers was zero
2. when the read starting address was valid and the read quantity of registers was out of range and the read starting address+ read quantity of registers was valid or invalid
3. when read starting address was invalid and the read quantity of registers was zero or out of range
4. when the write starting address was valid and the write quantity of registers was zero
5. when the write starting address and the write quantity of registers was valid and the write quantity + write starting address was within range but the actual data length and the and the specified data length were not in agreement
6. when the write starting address was valid and the write quantity was invalid and the write quantity + write starting address was or was not within range and the actual data length and the specified data length were or were not in agreement

VI.D.I. Fuzzed MODBUS Headers

When fuzzing the MODBUS header PLC Q displayed the following incorrect behaviors:

1. invalid function codes over 70 caused PLC Q to send a TCP Reset thereby terminating the communication
2. incorrectly specified MODBUS packet lengths caused PLC Q to send a TCP Reset thereby terminating the communication

When fuzzing the MODBUS header PLC M displayed the following incorrect behaviors:

1. incorrectly specified MODBUS packet lengths caused PLC M to send a TCP Reset thereby terminating the communication

VII. CONCLUSIONS

The blackPeer test framework allows for the economical creation of powerful test suites capable of quickly and efficiently testing a protocol's implementation against its specification. The blackPeer test framework is superior to conventional testing methods in two key areas:

1. the framework not only automates the generation of test cases but also automates the interpretation of the DUT's behavior in response to these test cases. This is achieved by the novel approach of statefully generating a test case oracle in conjunction with each generated test case
2. the framework provides a formal language medium in which the tester can express a test suite. This ability allows the tester to make quantifiable claims about the coverage of his tests

We ran approximately 5000 conformance tests against each PLC. The design of the test framework enabled efficient execution of the tests and quick interpretation of the results. The amount of coverage offered by other conformance testing tools, such as those distributed by the MODBUS-IDA, is much smaller than that offered by blackPeer and the interpretation of the resulting tests is a much more onerous task.

blackPeer discovered more than 60 categories of errors between the two PLC's tested. This is unacceptable for devices that may be deployed in critical safety systems. Most of the errors blackPeer detected came in the form of incorrect error responses. Human Machine Interface (HMI) software, such as WonderWare , interprets error responses and displays their meaning to the plant operator; such errors result in the operator receiving incorrect information. The receipt of this incorrect information could have catastrophic consequences.

The errors blackPeer discovered surrounding the use of illegal serial function codes was also troubling. The fact that PLC Q could be taken offline indefinitely by the simple execution of a MODBUS function code 8 sub function 4 request is extremely dangerous. This failure could result in the plant operator losing complete control of potentially safety critical systems. This error could not have been discovered by formally investigating the TCP or MODBUS protocols in isolation. A test framework capable of testing protocols embedded within protocols, such as blackPeer, is required for such discoveries.

The SCADA industry urgently needs to adopt better security robustness testing as standard practice. Industry bodies like the American National Standards Institute (ANSI) and the International Electromechanical Commission (IEC) need to mandate standardized security/conformance testing and certification for these critical devices. The number of errors detected in the two PLCs and the errors' significance shows that the security testing/certification of SCADA devices is critical to protect our national infrastructures from both accidental and deliberate attacks. As well as demonstrating the need for such testing, this paper also illustrates how it can be successfully conducted.

ACKNOWLEDGMENTS

We would like to thank Roman Shaffer of the U.S. Nuclear Regulatory Commission for his support and helpful suggestions. We would also like to thank the US TSWG for its partial funding of this project.

REFERENCES

[1] Vulnerability Note VU#190617: LiveData ICCP Server heap buffer, US Computer Emergency Response Team, May 16, 2006, http://www.kb.cert.org/vuls/id/190617

[2] D. P. DUGGAN, M. BERG, J. DILLINGER and J. STAMP; "Penetration Testing of Industrial Control Systems", Sandia National Laboratories, March 7, 2005.

[3] E.J. BYRES, J. CARTER, A. ELRAMLY and D. HOFFMAN; "Worlds in Collision: Ethernet on the Plant Floor", ISA Emerging Technologies Conference, Instrumentation Systems and Automation Society, Chicago, October (2002).

[4] R. KAKSONEN, M. LAASKO and A. TAKANEN, "Vulnerability analysis of software through syntax testing," University of Oulu, Finland, *Tech. Rep.* (2000).

[5] O. TAL, S. KNIGHT and T. DEAN, "Syntax-based vulnerability testing of frame-based network protocols," *Privacy, Security and Trust* (2004).

[6] D. INCE, "The automatic generation of test data," *The Computer Journal*, **30**, 1 (1987).

[7] E. G. SIRER and B. N. BERSHAD, "Using production grammars in software testing," *PLAN '99: Proceedings of the 2nd conference on Domain-specific languages*, New York, NY, pp 1-13, ACM Press, (1999).