Syntax-based Vulnerability Testing of Frame-based Network Protocols

Oded Tal
Department of Electrical and Computer Engineering, Royal Military College of Canada, Ontario, Canada
odedtal1@hotmail.com

Scott Knight
Department of Electrical and Computer Engineering, Royal Military College of Canada, Ontario, Canada
knight-s@rmc.ca

Tom Dean
Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada
thomas.dean@ece.queensu.ca

Abstract

Syntax-based vulnerability testing is a static black-box testing method for protocol implementations. It involves testing the Implementation Under Test (IUT) with a large number of mutated Protocol Data Units (PDUs), built by intentionally disobeying the protocol's syntax. Security vulnerabilities can be discovered by detecting anomalous behaviour or crashes in the IUT (e.g. segmentation faults, buffer, heap or stack overflows, etc.) when it attempts to parse and use a mutated PDU. Previous research has led to the development of a protocol testing framework and methodology for syntax-based testing of protocols, whose abstract syntax is based on ASN.1 (Abstract Syntax Notation), and whose transfer syntax is based on BER or DER (Basic or Distinguished Encoding Rules). These protocols have syntactic structure information embedded in the PDU. However, many protocols are not specified using such standards and do not include embedded syntactic structure information. Instead the byte sequence of the data in the PDUs is specified using frame-based PDU definitions in the protocol specification. This paper presents research that extends the previous testing tools and techniques to include frame-based protocols. OSPF is such a protocol. Several well-known OSPF protocol implementations are tested for protocol vulnerabilities. Security vulnerabilities have been found in some implementations.

1. Introduction

1.1. Preface

Computer network communications are subject to attack. Manufacturers of network products need efficient techniques to perform vulnerability testing on their product offerings. Security vulnerabilities can reveal themselves in many forms, such as segmentation faults, or stack, heap or buffer overflows, which cause the implementation of a theoretically sound protocol to fail. This can enable an attacker to gain privileges, or to disrupt or interfere with the functionality of the system implementing the protocol.

Previous work in syntax-based vulnerability testing by the PROTOS project is described in [1]. We describe in a related work [2] a general methodology and tools for security assessment of network protocol-implementations whose abstract syntax is based on ASN.1 (Abstract Syntax Notation) and whose transfer syntax is based on the BER/DER encoding schemes (Basic or Distinguished Encoding Rules) [3]. The testing framework is called Protocol-tester.

This work extends the testing framework to frame-based protocols. Frame-based protocols are those in which the structure of each PDU (Protocol Data Unit) is specified by a "frame," which explicitly defines the order of data fields in the PDU, as well as the exact length of each field. It involves testing the Implementation Under Test (IUT) with a large number of mutated PDUs, which are built by intentionally violating the protocol's syntax. The paper will describe the application of the vulnerability testing technique to the OSPF (Open Shortest Path First) protocol- a common routing protocol for local area networks.

1.2. Objective

The objective of this work is to extend the current work with Protocol-tester to frame-based and mixed protocols. A language and mechanism are required for supplying the syntactic structural information for a frame-based protocol such that individual PDUs can be parsed and test cases can be generated from them by using the existing generic Protocol-tester tools.

1.3. Paper Outline

Section 2 provides the background for the current work. Section 3 describes previous work in syntax testing and in OSPF testing. The approach and the tools are described in Section 4. Section 5 describes the test environment and test-suite design. Section 6 describes the results of testing OSPFv2 protocol implementations and Section 7 concludes the paper.

2. Background

2.1. Protocol Types

2.1.1. Frame-based Protocols: PDU structure is specified by a “frame,” which explicitly defines the order and the exact length of each data field. For example, the checksum field of the Hello packet of the OSPF protocol (Table 1) always starts at byte #13, and is 2-byte long. The number of “neighbours” can vary, but the first one always starts at byte #45, and each neighbour is 4-byte long. Other well-known frame-based protocols are ICMP, IP and UDP.

Table 1. The OSPF Hello packet

Byte	1	2	3	4
OSPF Header	Version	Type	Packet length	
	Router ID			
	Area ID			
	Checksum		Authentication Type	
	Authentication			
	Authentication			
Hello Header	Network Mask			
	Hello Interval		Options	Router Priority
	Router Dead Interval			
	Designated Router			
	Backup Designated Router			
Neighbours	Neighbour			
	……			

2.1.2. ASN.1 based protocols: The abstract syntax is specified using ASN.1- a formal notation standard. An important characteristic of ASN.1 is that the field length is not specified, and therefore a separate transfer syntax specification is needed, usually based on BER or DER. BER is a transfer syntax based on octets (8-bit bytes). DER- a subset of BER- follows the format of a TLV triplet (Type, Length, Value). Well-known ASN.1-based and BER/DER-encoded protocols are LDAP and SNMP.

2.1.3. Mixed protocols: A frame-based protocol, where at least one field consists of one or more TLV triplets or LV doublets. BGP, TCP and SSL are mixed protocols.

2.2. The OSPF Protocol

OSPF is based on the exchange of Link State Advertisements (LSAs) containing information on attached neighbours, including the cost (“metric”) of sending packets to each of them. Routers store LSAs in their databases and use Dijkstra's Algorithm to calculate the shortest path between source and target. In OSPFv2 [4] there are five types of OSPF packets (Table 2).

Table 2. OSPF packet types

#	Type name	Sub-unit	Comments
1		Neighbour	Sent periodically
2	Database Description	LSA header	Sent when a new neighbour join in
3	Link State Request (LSR)	LSA sub-header	Sent when parts of the link-state database are out-of-date.
4	Link State Update (LSU)	LSA	Sent in response to an LSR.
5	Link State Acknowledgement	LSA header	Sent in response to an LSU.

3. Previous work

3.1. Syntax testing

Syntax testing is a static, black-box testing technique for protocol implementations. Beizer [5] proposes that one specify the syntax for the protocol in a convenient notation such as Backus-Naur Form (BNF) [6]. Mutations are then made to the syntactic elements, and the modified grammar is used to produce aberrant test vectors (PDUs). Beizer suggests using an “anti-parser” to compile the grammar to produce “structured garbage.” He also suggests targeting one field of the input at a time at first, and then designing test cases with combinations of input. As in other black-box testing techniques, syntax testing does not have a clear stopping criterion. Test engineers have to use their experience, common sense or time/cost constraints while designing the test-suite.

3.2. The PROTOS project

The PROTOS project group from Oulu University, Finland, [1] followed Beizer’s approach. They have specified the grammar for the WAP, HTTP, LDAP and SNMP protocols. They parse the grammar specifications to produce a parse-tree representation of the protocol. They then manipulate the tree and use the mutated tree to generate thousands of malformed PDUs whose purpose is revealing faults in the handling of input data in the IUT. An IUT is considered to have passed the test if it has rejected the anomalous input without exhibiting such phenomena as crashing, hanging, or causing an illegal access to memory.

3.3. Protocol-tester

Protocol-tester [2] is a testing environment that uses a different approach for syntax-based vulnerability testing of protocol implementations, by mutating the PDUs rather than a complete expression of a specific protocol's grammar. The basis for PDU mutation is a general ASN.1 grammar and a general DER grammar, both written in TXL[1] [7]. A mark-up/implement architecture and various tools enable the test engineer to specify test-case parameters in a script-like language, and to automatically generate test cases, each containing one or more syntax errors, from a single, valid PDU. Test-case generation by Protocol-tester is driven by a number of mutation rules. The rules specify different kinds of changes to be made to the fields of the PDU based on the type of the field. Mutations can also involve a number of fields (e.g. swapping elements), or structural modifications (e.g. adding or removing sub-components of the PDU). PDU mutation is carried out in two phases: (1) An optional error-code is added to each ASN.1 type, in order to specify the nature of the mutation to be performed at the next phase. (2) Various mutations are executed on a valid PDU, using the transformation definitions specific to each mutation type. Each legitimate PDU can be used to generate many test-case PDUs (in the order of thousands).

The methodology was demonstrated on the X.509, PCKS7 and SNMP protocols and is suitable for ASN.1 based protocols using BER or DER.

3.4. Previous work in OSPF testing

Jou et al. [8] have used several **dynamic attacks,** called Seq++, MaxAge, MaxSeq++ and LSID attack to disrupt the operation of a target OSPF implementation. These attacks use **white-box** techniques to exploit specific perceived vulnerabilities in the protocol by using specific exchanges of PDUs with the IUT. They do not attempt to compromise the integrity of the router itself, only its routing tables. Jacob [9] carried out extensive conformance testing on an OSPFv2 package. Some of the test-cases were based on **static black-box** attacks using invalid packets. All the test cases were individually conceived, and manually constructed. Wu et al. [10] carried out conformance, interoperability and performance tests on OSPF using a **dynamic black-box** testing approach. Their main contribution was the addition of passive testing to the more traditional active testing. Passive testing focuses on protocol abnormalities such as route oscillations, useless route advertisements and exhausted routers, which usually appear only a long time after the network enters a "stable" state.

[1] **TXL** is a functional programming language, specifically designed to handle transformation tasks.

4. Approach

4.1. Overview

The general approach taken for this work is based on automatic, active, static black-box testing, (Figure 1). After using Snort [11] to gather OSPF packets from the test environment and saving them in separate binary files, each packet is transformed into a TXL-readable text file by **Packet-parser**. Each text file is then mutated by **Protocol-tester**, based on user-specified mutation commands. This results in numerous mutated PDUs, saved as binary files. The mutated packets are re-injected to the test environment by **Packet Injector and Monitor**. The same program also monitors the health of the target application by sending a good Link State Request (LSR) packet after each mutated packet, and waiting for the target's reply in the form of a Link State Update (LSU) packet. When the target fails to reply, the testing is stopped in order to investigate the cause of the failure.

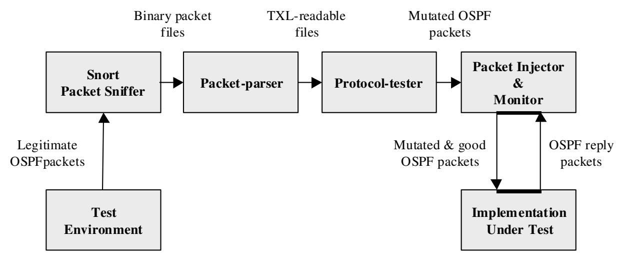

Fig. 1. The general approach

4.2. Packet-parser

Packet-parser is a C program developed as part of this work, which can parse a frame-based/mixed protocol PDU into its fields. It creates a text file, which serves as the input to Protocol-tester. This intermediate file has an ASN.1 style, so that the existing Protocol-tester tool can perform mutations on a generic, abstract representation of the PDU. Packet-parser parses the PDU by using syntactic structural specifications, specified by the test engineer using a set of **structure-files**. The structure-files are based on a Protocol Description Language (PDL). Each structure-file represents a stand-alone component of the protocol structure. For example, the structure-file for the Hello Header + Neighbours (Table 1) includes:

1) **Meta data**
- Header length: **20** bytes
- Sub-unit length: **4** bytes (for each neighbour)
- Type field number: **0** (no type field in this unit)
- Length field number: **0** (ditto)
- Number of sub-units field number: **0** (ditto).

2) **Next_files**

- Number of next_files: **0** (the Hello packet ends after the last neighbour. No other structure-files are required for describing/parsing the rest of the packet).
- Next_file numbers: none (ditto).

3) **Number of structural tokens**: **14** (see next section)
4) **Structural tokens** (numbers from 1-24):

5 (Sequence) **6** (open)
- **4** (4 byte integer- Network Mask)
- **2** (2 byte integer- Hello Interval)
- **1** (1 byte integer- Options)
- **1** (1 byte integer- Router Priority)
- **4** (4 byte integer- Router Dead Interval)
- **4** (4 byte integer- Designated router)
- **4** (4 byte integer- Backup Designated Router)

7 (close)
9 (Set of)
- **4** (4 byte integer- Neighbour)

7 (close)
12 (end of structure-file)

The Hello structure-file is a numerically encoded file, containing the above numbers in a specific order: 14H, 4, 0, 0, 0, 0, E, 5, 6, 4, 2, 1, 1, 4, 4, 4, 7, 9, 4, 7, C.

It is one of 13 structure-files, which describe OSPF packets to Packet-parser. Packet-parser parses a binary OSPF packet by consecutively reading bytes from the packet, according to the information in the relevant structure-file. It starts with the first structure-file (OSPF header) and then opens, uses and closes other structure-files recursively, as required.

The current numerically encoded structure files are created manually. Currently a friendlier, human-readable language is being developed to specify the packet structure of the protocol to be tested.

Packet-parser has been used for parsing frame-based protocols (OSPF, RIP) and mixed protocols (BGP, TCP).

4.3. Packet Injector and Monitor

Packet Injector and Monitor is a C program used to inject IP packets onto a network and monitor the health of the target IUT. The program uses the set of mutated test PDUs produced by Protocol-tester. It iterates over the set by reading a mutated packet, calculating and updating its OSPF header checksum and length, and adding an IP header. It then injects the packet onto the network. The monitor part of the tool then injects a good Link State Request (LSR) packet in order to trigger a response from the target IUT. The injection of the next mutated packet is carried out immediately after receiving a valid Link State Update packet from the target. If the target does not respond, the program halts and the test packet causing the failure in the IUT is investigated.

5. Conducting the tests

5.1. Test Environment

Figure 2 describes the network topology of the test environment. There were three OSPF IUTs involved in the testing: Zebra, Windows 2000 Advanced Server, and Cisco IOS. Workstations CSL3 and CSL7 are physical computers running Linux Red Hat 8.0 [12] and VMware 3.2 [13]. All other machines are virtual VMware machines on the respective Linux host workstations.

The Zebra 0.93 OSPF daemon [14] was tested on three platforms: Red Hat 8.0 Linux on a real physical machine and on a virtual machine and OpenBSD 3.3 [15] on a virtual machine.

The Windows 2000 Advanced Server [16] was tested on a virtual machine. A Cisco 2621 Router running IOS 12.0(7)T was also tested. A Windows 2000 virtual machine was used to run Packet Injector and Monitor.

5.2. Test-suite Design

Fifteen test-suites, each containing between 3,000 – 4,000 mutated packets, were produced for each IUT by mutating all five OSPF packet types, using three mutation strategies:

1. **Replacement**: Replacing one field at a time with boundary and mid-way values, i.e. 00, 7F, 80, FF, 0000, 7FFF, 8001, FFFF, 00000000, 7FFFFFFF, 80000001, FFFFFFFF, 0000000000000000 and FFFFFFFFFFFFFFFF.
2. **Removal**: Removing one field at a time, and then removing pairs of fields, one pair at a time.
3. **Permutation**: Permutating pairs of fields within each SEQUENCE and SET.

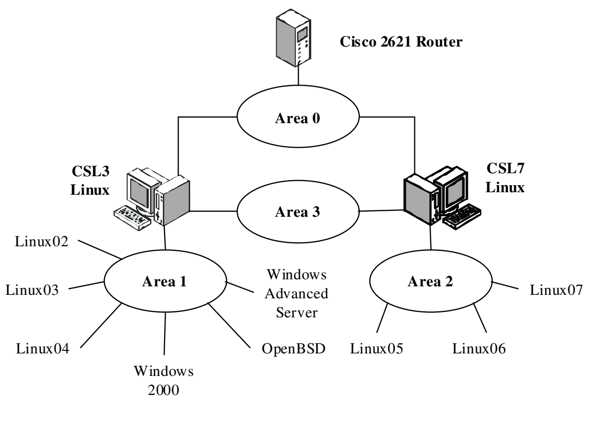

Fig. 2. Test environment topology

6. Results

6.1. General

Table 3 summarizes the results of attacking OSPF implementations on 5 different targets. It shows for each test-suite the OSPF type of the original, unmutated PDU, the mutation strategies and the results. A "+" sign means that the OSPF implementation crashed at least once.

6.2. Test Results for Zebra OSPF Daemon

6.2.1. Overview. The OSPF daemon crashed at least once on 10 out of 15 test-suites on Linux and VMware/Linux, and on 8 out of 15 test-suites on OpenBSD. After each crash the operating system continued to function normally, however the OSPF daemon had to be reset to continue with testing. Therefore, at the very least the systems seem to be vulnerable to Denial of Service (DoS) attacks.

The test cases were run against each IUT a number of times. It is interesting to note that the crashing behaviour of the OSPF daemon on the targets was inconsistent; the same test cases sometimes caused the target to crash and at other times they did not. This behaviour was investigated by using debugging tools and examining the IUT source code. The observed crashes were caused by three security vulnerabilities.

Table 3. Test results

Original packet type	Mutation strategy	OSPF daemon crashes*				
		Zebra OSPF daemon on			Win	Cis
		1	2	3		
Hello	Replace	+	+	+	+	
	Remove					
	Permutate	+	+	+		
Database Description	Replace	+	+	+		
	Remove					
	Permutate	+	+	+		
Link State Request	Replace		+	+		
	Remove					
	Permutate	+	+	+		
Link State Update	Replace	+	+	+	+	
	Remove	+				
	Permutate	+	+	+		
Link State Ack.	Replace	+	+			
	Remove					
	Permutate	+	+			
Successful test-suites	15 test-suites	10	10	8	2	0

* 1:	VMware 3.2 + Red Hat 8.0 Linux
2:	Red Hat 8.0 Linux
3:	VMware 3.2 + OpenBSD 3.3
Win:	Windows 2000 Advanced Server
Cis:	Cisco 2621 Router, IOS 12.0(7)T

6.2.2. A security vulnerability in the OSPF checksum routine. The OSPF checksum routine (in_cksum in checksum.c) does not compare the length field in the OSPF header to the length field in the IP header. When the length field in the mutated OSPF header is larger than the actual packet size, the routine can crash due to a segmentation fault. This occurs because the packet is stored in a buffer in the heap. As the checksum algorithm continues to read past the actual end of the packet due to the erroneous length field it can end up reading past the end of the heap into unauthorized memory space.

6.2.3. A security vulnerability in the LSA checksum routine. The LSA checksum routine (ospf_lsa_checksum in ospf_lsa.c) does not verify the validity of the length field in the LSA header. When the length field in the mutated LSA header is larger than the actual packet size, the routine can crash due to a segmentation fault. This can happen only with LSU packets. The causes of this error are very similar to those in section 6.2.2 above.

6.2.4. A security vulnerability in the OSPF daemon's main routine. The implementation crashed because of a segmentation fault caused while calling the dynamic memory allocation function calloc(). This class of error is often due to heap overflow. It can obviously be exploited by a hacker for DoS attack. However in some cases this class of error can be exploited with more dangerous results such as the execution of arbitrary malicious code.

6.3. Test Results for Windows 2000 Advanced Server

6.3.1. Overview. The OSPF daemon crashed at least once on 2 out of 15 test-suites. Three types of crashes were observed:

1. **Temporary crash:** the Operating System (OS) continued to function normally, and the OSPF daemon started working again after about 10 minutes.
2. **Semi-permanent crash:** the OS continued to function normally. Stopping and restarting the OSPF daemon restored it to service.
3. **Permanent crash:** the OS unexpectedly terminated several services, including System Event Notification, Routing and Remote Access, Removable Storage, Network Connections, Internet Authentication and COM + Event System. It had to be rebooted.

Again the test cases were run against the IUT a number of times. The crashing behaviour of the OSPF daemon was inconsistent. The same test-suite sometimes caused it to crash at different mutated packets, to crash differently, or not to crash at all. This suggests that the vulnerability is related to the heap and is sensitive to the location of the packet in the heap.

6.3.2. Causes for crashing. The root causes for the crashes have not been isolated since the source code is not available, and the vendor may not welcome reverse engineering of the mechanism in question. Further investigation will involve additional black-box syntax-based testing to identify recurring fault scenarios and then working with the vendor to identify the cause.

6.4. Test results for the Cisco Router

The OSPF implementation in Cisco 2621 Router did not crash. Therefore, it was also tested by manually mutated packets, based on additional mutation strategies, including:

1. **Structure-level mutations**, e.g. Router LSAs, where the number of Links is smaller or larger than specified in the Router header, including zero.
2. **Contents-level mutations**, e.g. LSA sequence number is 7FFFFFFF (MaxSeq# attack).
3. **Stress testing**: sending maximum length packets (64k bytes) for two hours.

The implementation did not crash as a result of any of the test scenarios.

7. Conclusions

The paper describes a vulnerability testing technique for frame-based network communication protocols. The vulnerability-testing framework was used to test several commercial implementations of the OSPF protocol.

The Zebra 0.93 OSPF daemon was shown to be vulnerable to Denial of Service attacks (at least) due to bugs in its OSPF checksum function, LSA checksum function and the main routine.

The OSPF daemon in Windows 2000 Advances Server was also shown to be vulnerable to Denial of Service attacks.

No vulnerabilities were discovered in the Cisco 2621 (IOS 12.0) OSPF daemon.

The test trials of these OSPF products yielded some interesting results but they are more interesting as a proof-of-concept for the technique.

This work is novel in that it is a static black-box technique specifically suited to security vulnerability testing. A vendor or system integrator using the technique does not need to know the internal structure of the network products being tested. The technique is easily portable to new protocols.

Future work includes improving the protocol description language to allow PDU structural information to be more easily specified by a test engineer. There is also important work that can be done in identifying common semantic relationships between syntactic elements of a protocol. These tend to be captured in the textual descriptions in the protocol specifications. Identifying common relationships and can be useful in identifying more complex testing strategies that can be used to improve the current mutation command libraries.

References

1. R. Kaksonen, M. Laasko and A. Takanen. (2000). “Vulnerability analysis of software through syntax testing”. University of Oulu, Finland. Available: http://www.ee.oulu.fi/research/ouspg/protos/analysis/WP2000-robustness/index.html.
2. Y. Turcotte, O. Tal, S. Knight and T. Dean, “Universal methodology and tools for syntax-based vulnerability testing of protocol implementations”. Accepted for publication in MILCOM 2004.
3. O. Dubuisson, *ASN.1 Communication between Heterogeneous Systems*. Academic Press, San Diego, 2001.
4. J. Moy, “OSPF Version 2, RFC 2328” (1988, April). Available: http://www.faqs.org/rfcs/rfc2328.html.
5. B. Beizer, *Software Testing Techniques*, 2nd Edition. Van Nostrand Reinhold, New York, 1990.
6. P. Naur, (ed.), “Revised report on the algorithmic language ALGOL 60”, *Communications of the ACM*, vol. 3, May 1960, pp. 299-314.
7. J. Cordy. (2000). “The TXL programming language v.10”. Available: http://www.txl.ca/nabouttxl.html.
8. Y.F. Jou, F. Gong, C. Sargor, X. Wu, S.F. Wu, H.C. Chang and F. Wang, “Design and implementation of a scalable intrusion detection system for the protection of network infrastructure,” in Proc. DARPA Information Survivability Conference and Exposition, 2000.
9. D. Jacob, “Testing intra-domain routing in a network simulator”. Universität des Saarlandes, Germany, Diploma Thesis, Jan. 2002.
10. J. Wu, Y. Zhao and X. Yin, “From active to passive: Progress in testing of Internet protocols”, in Proc. 21st International Conference on Formal Techniques for Networked and Distributed Systems, 2001.
11. Snort. Available: http://www.snort.org/.
12. Red Hat 8.0. (2002, Sep.). Available: http://www.redhat.com/about/presscenter/2002/press_eightoh.html.
13. VMware 3.2. Available: http://www.vmware.com/support/ws3/doc/releasenotes_ws32.html.
14. Zebra 0.93. Available: http://www.zebra.org/.
15. OpenBSD 3.3 (2003, May). Available: http://www.openbsd.org/33.html.
16. Windows 2000 Advanced Server. Available: http://www.microsoft.com/windows2000/advancedserver/default.as.