www.ioirp.com
International Journal of Innovative Research in Computer Science and Engineering (IJIRCSE)
ISSN: 2394-6364, Volume – 1, Issue – 6. May 2015

TESTNET-An Integrated Testing Environment For Network Protocols Using Symbolic Execution

Kalesh E S
ME CSE, Maharaja Prithvi Engineering College, Avinashi
kaleshelavungal@gmail.com

***Abstract*— Network protocol implementations are complex and difficult to test since the packet combinations in various scenarios are difficult to create manually. Some of the popular network protocols are DNS, DHCP and Zeroconf. Those network protocol implementations are prone to flaws, security vulnerabilities and interoperability issues caused by developer mistakes and ambiguous requirements in protocol specifications. Identifying such bugs is not easy since many of them are manifest themselves only after prolonged operation; reasoning about semantic errors requires a machine-readable specification; and the state space of complex protocol implementations is large. This paper presents a different approach that combines symbolic execution and rule-based specifications to detect various types of bugs or issues in network protocol implementations. This will automatically generate high coverage test input packets for testing the protocol implementation. This is the core idea behind this. Protocol implementation using single- and multi-packet exchange symbolic execution (targeting stateless and stateful protocols, respectively) and then use these packets to detect potential violations of manual rules derived from the protocol specification, and check the interoperability of different implementations of the same network protocol. Based on these techniques I present a system called TESTNET which will evaluate it on multiple implementations of two network protocols: Zeroconf, a service discovery protocol, and DHCP, a network configuration protocol. Problems like non-trivial bugs and interoperability issues will address by TESTNET.**

I. INTRODUCTION

A communications protocol is a system of digital rules for data exchange within or between computers. Communicating systems use well-defined formats for exchanging messages. Each message has an exact meaning intended to elicit a response from a range of possible responses pre-determined for that particular situation. Thus, a protocol must define the syntax, semantics, and synchronization of communication; the specified behavior is typically independent of how it is to be implemented. A protocol can therefore be implemented as hardware, software, or both. Communications protocols have to be agreed upon by the parties involved. To reach agreement, a protocol may be developed into a technical standard. A programming language describes the same for computations, so there is a close analogy between protocols and programming languages: protocols are to communications as programming languages are to computations.

The two types of methods used for Image Processing are Analog and Digital Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

In modern sciences and technologies, images also gain much broader scopes due to the ever growing importance of scientific visualization (of often large-scale complex scientific/experimental data). Examples include microarray data in genetic research, or real-time multi-asset portfolio trading in finance.

A. Vulnerabilities in Network Protocols

Implementations of network protocols used today, such as DNS, Zeroconf and Dynamic Host Configuration Protocol (DHCP) are often prone to errors. Ambiguities in network protocol specifications can cause different interpretations by developers, leading to bugs and interoperability problems in the corresponding implementations of network services. The complexity of network protocols makes errors difficult to detect, even for well-studied and mature protocols: errors may only manifest themselves after complex sequences of network packets. For example, DNS server implementations that are vulnerable to cache poisoning attacks only exhibit problems in specific scenarios. The impact of such vulnerabilities can be severe though, and the cost of fixing them can be high.

B. Network Protocol bugs

Although a large body of work has focused on finding software errors, existing techniques have significant weaknesses when applied to network protocol implementations because many bugs manifest themselves only after prolonged operation in a production network next is reasoning about semantic errors in network protocol implementations requires a machine-readable specification of the intended protocol behavior and the final reason is the state space of complex network protocol implementations is large.

C. Symbolic Execution

Symbolic Execution is a program analysis technique that can generate inputs that explore multiple paths in a program with rule-based specifications to check automatically a network protocol implementation against its specification and discover various types of errors, which would be hard to detect manually.

D. Network Protocols

A network can be defined as a collection of entities interconnected by communication technologies that enable the exchange of information. The communicating entities require an agreement to exchange information and such agreements are called network protocols. The messages exchanged by these entities are called packets, and a sequence of packets is referred to as a packet stream. When a network protocol is designed, all the information regarding methods, behavior and packet formats are described in documents, which form the protocol specification, to be referenced by developers of a protocol implementation. In UNIX and other operating systems, implementations of network protocols are called network daemons.

We give two examples of network protocols: Zeroconf and DHCP. Both are widely used and implemented by different vendors. They are used throughout the paper to demonstrate the various problems addressed by our approach.

II. TESTNET BASED SYMBOLIC EXECUTION

Implementations of network protocols, such as DNS, DHCP and Zeroconf, are prone to flaws, security vulnerabilities and interoperability issues caused by developer mistakes and ambiguous requirements in protocol specifications. Detecting such problems is not easy because many bugs manifest themselves only after prolonged operation second Reasoning about semantic errors requires machine-readable specification; and third is the state space of complex protocol implementations is large.

The existing project presents a novel approach that combines symbolic execution and rule-based specifications to detect various types of flaws in network protocol implementations. The core idea behind its approach is to automatically generate high-coverage test input packets for a network protocol implementation using single- and multi-packet exchange symbolic execution and then use these packets to detect potential violations of manual rules derived from the protocol specification, and check the interoperability of different implementations of the same network protocol. I present a system based on these techniques, TESTNET, and evaluate it on multiple implementations of two network protocols: Zeroconf, a service discovery protocol, and DHCP, a network configuration protocol. TESTNET is able to discover non-trivial bugs as well as interoperability problems.

Disadvantages:

1. Currently the Algorithm is executed in a single client machine that may affect the performance.
2. There is no centralized system for continuous integration and job scheduling.
3. No error reporting tools in this system.

A. Rule Management

This module deals with the rule derivation, creation and validation of TESTNET System. Rule management is one of the initial settings which is to be done before we execute the TestNET framework. We can set any number of rules in the system. The rules are depends on the protocol specification. If one user wants to set the rules in their own way they can set it accordingly. There is a separate page for creating the rules. If the specified rules are matching with the given rule then that marked as a violation. And that should be addressed later to avoid the issues. The major rules which are to be set, should be listed out first and entered into the system one by one. The number of rules are increased, means the execution times increases. That mean the

number of rules are directly proportional to the execution time. The first step is to develop a rule-based specification from a protocol specification. The requirements describing behavioral properties of the protocol are extracted manually from the protocol specification and expressed in terms of the desired input-output behavior (i.e., the set of packets). The major functionalities are described below. Developers derive rules manually from the protocol specification and express them in a high-level packet stream language, which states invalid patterns in the sequence of packets exchanged between a client and a server. The language permits rules to refer to packet header fields and their relationship within a packet stream. Rules can be extracted easily from Request for Comments (RFC) network protocol specifications, thus encoding the externally-visible behavior of a network protocol in terms of input and output packets.

B. Creation of packet rules

The first step is to develop a rule-based specification from a protocol specification. The requirements describing behavioral properties of the protocol are extracted manually from the protocol specification and expressed in terms of the desired input-output behavior (i.e., the set of packets). We can set any number of rules in the system. The rules are depends on the protocol specification. If one user wants to set the rules in their own way they can set it accordingly. There is a separate page for creating the rules. If the specified rules are matching with the given rule then that marked as a violation. And that should be addressed later to avoid the issues. The major rules which are to be set, should be listed out first and entered into the system one by one. The number of rules are increased, means the execution times increases. That mean the number of rules are directly proportional to the execution time. The first step is to develop a rule-based specification from a protocol specification. TESTNET provides a packet rule language to describe correct sequences of packets. A rule basically contains the parameter and the value which it can acquire. The conditions may be logical and Boolean expressions.

C. Validation of packet rules

The rule validation happens when new packets are generated from binary implantation for the DHCP Server. The captured input and output packets from the previous step are validated against the rule-based specification. TESTNET translates the specification rules into a set of non-deterministic finite automata (NFAs). Through analyzing all captured replay packets against each NFA, TESTNET detects rule violations. Rule violations and warnings are logged in database table to analyses the efficiency of TESTNET System.

D. Generation of test packets

To validate as many packet rules as possible, TESTNET generates a good set of test packets with high code coverage. It uses symbolic execution to explore a large number of code paths in the network protocol implementation and, based on this, synthesizes a set of test input packets. To explore the state space broadly, TESTNET repeatedly marks parts of a packet as symbolic. For deep exploration, it repeatedly performs symbolic execution on selected input. Packets to generate sequences of test input packets.

E. Replay of test packets

The generated test packets are replayed on the original network implementation. Each test packet is sent to the implementation in a controlled network environment, and the output packets generated by the implementation are recorded, together with the input packets, as a packet stream. When the implementation executes in the configured environment, TESTNET selects a test input packet and then controls the client so that it can send the test packet when the implementation is ready to receive it, such as after completing service registrations. Replayed packets causing crashes of the implementation are reported during the replay process. To validate the network protocol implementation, TESTNET records all network traffic, i.e., input and output packets generated by the implementation and clients during the replay. The captured traffic is used. A tool called Wireshark is used for packet monitoring.

III. CONCLUSION

The project TESTNET has succeeded in implementing a protocol testing with in dept. code analysis. The code execution using Symbolic Execution was carried out with Microsoft PEX . The module helped me to generate excellent code analysis results that explored every in dept. functions within branches of code also. The system also checked DHCP protocol executions with verification on there The Announce, release and Offer function with proper client requests. The system could also generate a

list of generic bugs based on the rules created by an admin level user. The bugs were helpful in analyzing the system performance on level and binary level. The Interoperability functions of the framework are also tested by accepting packets from a Linux kernel. The interface and functionality of the project is a mile stone in protocol testing domain.

References

[1] JaeSeung Song, Member, IEEE, Cristian Cadar, Member, IEEE, and Peter Pietzuch, Member, IEEE; SYMBEXNET: Testing Network Protocol Implementations with Symbolic Execution and Rule-Based Specifications..

[2] R. Alur and B.-Y. Wang, 13th International Conference. Computer Aided Verification, 2001, pp. 169–181; Verifying Network Protocol Implementations by Symbolic Refinement Checking.C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data," IEEE Trans. Parallel and Distributed Systems, vol. 23, no. 8, pp. 1467-1479, Aug. 2012.

[3] Hao, D. Lee, R. K. Sinha, and N. Griffeth, IEEE/ACM Transaction. Networking. vol. 12, no. 5, pp. 823–836, Oct. 2004; Integrated system interoperability testing with applications to VoIPR. E. Shen, E. Shi, and B. Waters, "Predicate Privacy in Encryption Systems," Proc. Sixth Theory of Cryptography Conf. Theory of Cryptography (TCC), 2009.

[4] C. Cadar, D. Dunbar, and D. Engler Stanford University, 8th USENIX Conf. Operat. Syst. Des. Implementation, 2008, pp. 209–224; KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs.

[5] C. Cadar, P. Godefroid, S. Khurshid, C. S. Pasareanu, K. Sen, N.Tillmann, and W. Visser, in Proc. 33rd International. Conference Software Engineering. 2011, pp. 1066–1071; Symbolic execution for software testing in practice: Preliminary assessment.

[6] M. Duflot, M. Kwiatkowska, G. Norman, D. Parker, S. Peyronnet, C. Picaronny, and J. Sproston, in Handbook of Formal Methods in Industrial Critical Systems, Wiley-IEEE Press,USA, 2010; Practical applications of probabilistic model checking to communication protocols..

[7] Wan-SeobByoun , Cheol-Jung Yoo† , Hye-min Noh and Ok-Bae Chang, Chonbuk National University, Jeonju, Jeonbuk, South KOREA; Test Case Generation Technique for Interoperability Test of Component Based Software from State Transition Model.