ISSN:0975-9646

Verifications of Network Protocol Implementations - A Review

Saleema D[1], Bejoy Abraham[2]

[1]PG Scholar,College of Engineering Perumon,
Perinad P.O,Kollam,Kerala,India
[2]Associate Professor in CSE,College of Engineering Perumon,
Perinad P.O, Kollam, Kerala,India

Abstract-Protocols used with different OSI layers plays an important role in quality data communication. Ambiguities in protocol specification leads to different interpretations by developers. These ambiguities should be minimized to avoid bugs and interoperability problems. There are several approaches dealing with protocol implementation testing which include a binary level verifications and code level analysis of a protocol implementations. In this article we present a formal review on verifications of protocol implementations done through static and dynamic techniques. Recent developments include a Symbolic Execution together with a Rule-Based specification.

Keywords- Static verification, Dynamic Verification

I. INTRODUCTION

Network protocols are always prone to implementations flaws, security vulnerabilities and interoperability issues either caused by developer mistakes or ambiguities in protocol specifications like RFC. These problems are very difficult to detect because many bugs manifest only after prolonged operations of protocol implementations and reasoning about semantic errors requires a machine readable specification[1]. There are several approaches dealing with protocol implementation testing which includes both static and dynamic analysis of source code of implementation. Static analysis provides a static algorithm for analysis of source code while Dynamic testing includes dynamic test case generation by symbolically executing the source code of implementation. In this paper we present a literature review on the details of various types of methods and techniques used for protocol implementation testing and their performance towards different protocols.

II. STATIC CHECKING OF PROTOCOL IMPLEMENTATIONS

The static (compile) time source code analysis of Network protocols implementation is done by using a verification tool Pistachio that checks source code of implementation against rule based specifications. This was applied to implementations of SSH and RCP protocols and detected many bugs including security vulnerabilities. Pistachio was explained with an alternating bit protocol implementation.

•Alternating bit protocol

1. Start by sending n = 1
2. If n is received, send n + 1
3. Otherwise resend

```
int main(void)
int socket, value=1, recvalue;
while(1)
send(socket,&value,sizeof(int));
{ recv(socket,&recvalue,sizeof(int));
if (recvalue == value)
value += 2;
send(socket,&value,sizeof(int));
```

Starting with an empty hypothesis successively setting the values, and when we reached the conclusion a rule is validated.

```
ie Ø (program entry)
=> send(_, out, _)
out[0..3] = 1
 n := 1
recv(_, in, _)
 in[0..3] = n
=> send(_, out, _)
out[0..3] = in[0..3] + 1
n := out[0..3]
```

All the communications are in terms of two communication primitives send and receive. We will bind the value of val in two parameters in and out. The rule specification corresponding to this is shown above, setting the value of n to 1.If n values are received, send n+1, otherwise resend n. Program execution is simulated by symbolic execution, which uses an automatic Theorem Prover that tracks program variables, path conditions and checks whether rules are satisfied and also branch conditions hold or not. Found Pistachio is a very fast tool that detects many security related errors with low false positive and negative rates.

Darwin theorem prover returns an “yes” or “no” corresponding to each theorem. Pistachio generates a warning if a valid conclusion is not created from a hypothesis. It generate a skeleton rule based specifications for library functions, but some functions like getrrno() is not modeled. Core rule set detected were functionality, message format, compatibility, library call errors ,and buffer overflow. Conclusion was not provable because of out of bound exceptions due to buffer overflows. To detect all these Pistachio can write regression rules. But Pistachio couldn’t realize the depth of coverage of protocol implementation source code. So Pistachio is extended with a symbolic execution technique for detecting generic bugs

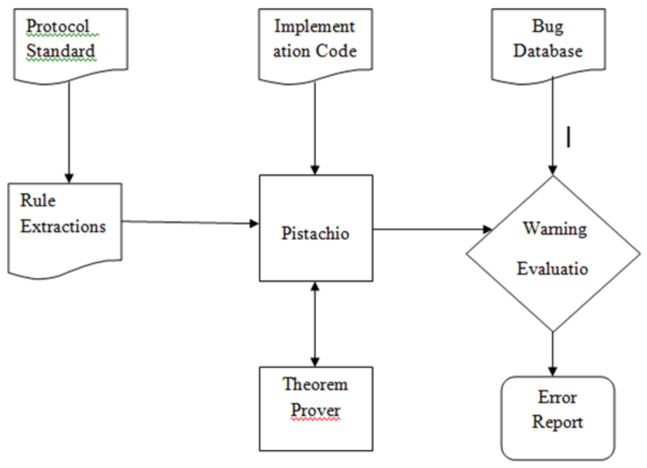

Fig 1. Fig.1.Pistachio Design

III. Dynamic Verification Of Protocol Implenetation

Recent developments in verifications of network protocols combine symbolic execution, a program analysis technique that can generate inputs that explore multiple paths in a program with rule based specifications to discover various types of errors, which would be hard to detect manually 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. Using symbolic execution, the tool can generate an exhaustive set of input packets that achieve a broad and deep exploration of the program state space. The practical verification tool used is SYMBEXNET. It is found that the former tool SYMNV was a good tool for verifying network protocol implementation flaws but the only limitation was it uses only single packet symbolic execution and couldn't check whether different implementations of same network protocols interoperate or not. This limitations were throne out in SYMBEXNET. Here DHCP protocols were analyzed for implementation bugs.

A. *Rule Derivation*

Rules are extracted from the RFC or IETF standard of protocol, keywords such as SHALL, SHOULD, MUST etc contained in the RFC documents are good candidates for this.

Example rule for discovering inconsistent Query IDs in DNS packets

```
1.query{source ip != 224.0.0.251
 2 AND flag.QR = 0x00
3 AND questions != 0x00}
4 ;
5 response {destination_ip = @query.source_ip
 6 AND flag.QR = 0x80
7 AND ANY data.answer(
8 name = @query.question.name)
9 AND id != @query.id}
```

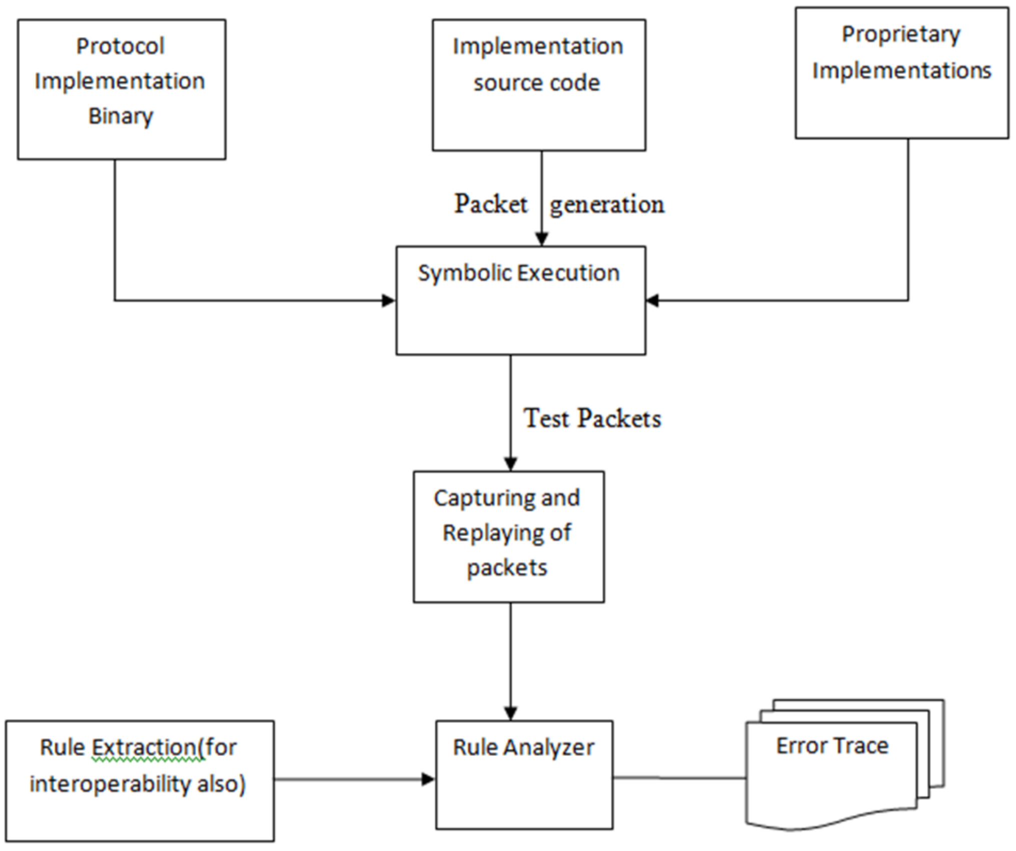
Protocol Implementation Binary
Implementation source code
Proprietary Implementations
Packet generation
Symbolic Execution
Test Packets
Capturing and Replaying of packets
Rule Extraction(for interoperability also)
Rule Analyzer
Error Trace

Fig.2.SYMBEXNET Design

Thus encoding the externally-visible behavior of a network protocol in terms of input and output packets. A rule basically contains a parameter and the value which it can acquire. The conditions may be logical or Boolean expressions.

B. *Validation of packet rules*

The rule validation happens when new packets are generated from binary implementation for the DHCP Server. The captured input and output packets from the previous step are validated against the rule-based specification. SYMBEXNET translates the specification rules into a set of non-deterministic finite automata (NFAs). Through analyzing all captured replay packets against each NFA, SYMBEXNET detects rule violations.

C. *Symbolic Execution*

Symbolic Execution is a program analysis technique used in dynamic software testing. It generates an exhaustive set of test input packets with high code coverage. The idea is to use symbolic variables instead of actual data, which generates an execution tree corresponding to each conditional branch statements and a path constraint is created which is solved by using a satisfiability checker. In SYMBEXNET symbolic execution is carried out in packet fields which considers combinations of fields symbolic in multiple rounds.

IV. EXPERIMENTAL RESULT OF STATIC VERSUS DYNAMIC VERIFICATIONS

The main idea behind static checking of protocol implementation is to detect maximum bugs with minimum false positives. Core rule set generated in Pistachio are message structure and data transfer(format, structure, data transfer in SSH2),compatibility rules(backward compatibility with SSH1),Functionality rule(what functionality should or should not support(none authentication is not supported).Most of the warnings are related to bug database which includes buffer overflow and authentication failure. False positives are due to incorrect specifications in library calls, also limitations of theorem prover and loop breaking.

Dynamic testing of DHCP protocol implementation detected a large no of bugs which are categorized into Generic Bugs(from Symbolic execution),Semantic Bugs(from rule based specifications), and interoperability bugs .

Example Bugs

Generic Bugs: Vulnerability caused by source port number

Semantic Bugs: Incorrect responses to unknown record class.

Interoperability bugs: Incorrect responses to broadcast address .

V. CONCLUSIONS

From our analysis of Static versus Dynamic testing of different network protocol implementation testing it is found that Dynamic testing can generates a large set of dynamic test suites that explores multiple program paths to detect a large set of bugs. Dynamic Testing strategy is a stateful execution apart from the stateless static testing.

ACKNOWLEDGMENT

This paper recounts research done with others. In particular we thank JaeSeung Song,Cristian Cadar,Peter Pietzuch, Member, IEEE. for their work SYMBEXNET and Udrea, C. Lumezanu, and J. S. Foster, "Rule-based static analysis of network protocol implementations helped us in making the review.

REFERENCES

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

[2] O. Udrea, C. Lumezanu, and J. S. Foster, "Rule-based static analysis of network protocol implementations," Inf. Comput., vol. 206, pp. 130–157, Feb. 2008.

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

[4] C. Cadar, P. Godefroid, S. Khurshid, C. S. P_as_areanu, K. Sen, N. Tillmann, and W. Visser, "Symbolic execution for software testing in practice: Preliminary assessment," in Proc. 33rd Int. Conf. Softw. Eng., 2011, pp. 1066–1071.

[5] J. C. King, "Symbolic execution and program testing," Commun. ACM, vol. 19, pp. 385–394, Jul. 1976

[6] R. Droms. (1997, Mar.). IETF RFC 2131: Dynamic host configuration protocol. [Online].Available: http://www.ietf.org/rfc/rfc2131.txt

[7] R. Hao, D. Lee, R. K. Sinha, and N. Griffeth, "Integrated system interoperability testing with applications to VoIP," IEEE/ACM Trans. Netw., vol. 12, no. 5, pp. 823–836, Oct. 2004.

[8] P. D. Marinescu and C. Cadar, "Make test-zesti: A symbolic execution solution for improving regression testing," in Proc. 34th Int. Conf. Softw. Eng., Jun. 2012, pp. 716–726.

[9] O. Kon and R. Castanet, "Test generation for interworking systems,"Comput. Commun., vol. 23, no. 7, pp. 642–652, 2000.

[10] R. Sasnauskas, O. Landsiedel, M. H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle, "Kleenet: Discovering insidious interaction bugs in wireless sensor networks before deployment," in Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sens. Netw.2010, pp. 186–196.

[11] A. Vallejo, J. Ruiz, J. Abella, A. Zaballos, and J. Selga, "State of the art of ipv6 conformance and interoperability testing," IEEE Commun.Mag., vol. 45, no. 10, pp. 140–146, Oct. 2007.