TOWARDS RUN-TIME PROTOCOL ANOMALY DETECTION AND VERIFICATION

InSeon Yoo and Ulrich Ultes-Nitsche
Department of Computer Science, University of Fribourg,
Chemin du Musee 3, Fribourg, CH1700, Switzerland.

Keywords: Run-time Protocol Verification, Protocol Anomaly Detection, SDL, EFSM/CEFSM

Abstract: 'How to verify incoming packets whether they follow standards or not?' and 'How to detect protocol anomalies in real-time?', we seek to answer these questions. In order to solve these questions, we have designed a packet verifier with packet inspection and sanity check. In this work, we specify TCP transaction behaviours declaratively in a high-level language called Specification and Description Language (SDL). This specification will be then compiled into an inspection engine program for oberving packets. In addition, the SanityChecker covers protocol header anomalies.

1 INTRODUCTION

Protocols are created with specifications, known as RFCs, to dictate proper use and communication. An anomaly is defined as something different, abnormal, peculiar, or not easily classified. Protocol anomaly refers to all exceptions related to protocol format and protocol behaviour with respect to common practice on the Internet and standard specifications. This includes network and transport layer protocol anomalies in layer 3 and layer 4 and application layer protocol anomalies in layer 6 and layer 7.

Protocol anomaly detection is essential for understanding new attacks. Without names and prior documentation, new attacks can only be defended against by understanding their intrusion methods and their effects. However, important to note is that not all threats or attacks exhibit themselves as protocol anomalies. Some types of application logic attacks, denial of service (DoS) attacks, viruses, and reconnaissance methods all appear as perfectly legitimate network traffic. In this paper, we present our packet verifier model based on a specification of security protocols given in a high-level language, called Specification and Description Language (SDL). Then, we address how to verify TCP protocol transition.

2 EXAMPLE CASES OF PROTOCOL ANOMALIES

2.1 Ping of Death

Attackers sends a fragmented PING request that exceeds the maximum IP packet size(64KB), causing vulnerable systems to crash. The idea behind the Ping of Death and similar attacks is that the user sends a packet that is malformed in such a way that the target system will not know how to handle the packet. The Ping of Death attack (Fyodor, 1996) sent IP packets of a size greater than 65,535 bytes to the target computer. IP packets of this size are abnormal, but applications can be built that are capable of creating them. Carefully programmed operating systems could detect and safely handle abnormal IP packets, but some failed to do this. ICMP ping utilities often included large-packet capability and became the namesake of the problem, although UDP and other IP-based protocols also could transport Ping of Death.

2.2 Land Attack

The land attacks (CISCO, 1997) are also known as IP DOS (Denial of Service) (Fyodor, 1997). The land attack involves the perpetrator sending a stream of TCP SYN packets that have the source IP address and TCP port number set to the same value as the destination

Yoo I. and Ultes-nitsche U. (2004).
TOWARDS RUN-TIME PROTOCOL ANOMALY DETECTION AND VERIFICATION.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 299-304
DOI: 10.5220/0001395802990304

address and port number, i.e., that of the attacked host. Some implementations of TCP/IP cannot handle this theoretically impossible condition, causing the operating system to go into a loop as it tries to resolve repeated connections to itself.

2.3 SYN flood attack

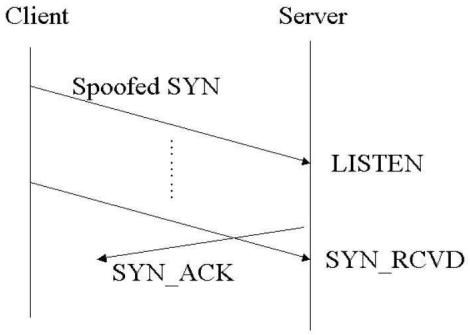

Figure 1: SYN Flooding

The client system begins by sending a SYN message to the server (CERT/CA-1996-21, 2000). The server then acknowledges the SYN message by sending SYN-ACK message to the client like in Figure1. The client then finishes establishing the connection by responding with an ACK message. The connection between the client and the server is then open, and the service-specific data can be exchanged between the client and the server.

The TCP SYN attack exploits this design by having an attacking source host send SYN packets with random source addresses to a victim host. The victim destination host sends a SYN ACK back to the random source address and adds an entry to the connection queue. Since the SYN ACK is destined for an incorrect or nonexistent host, the last part of the three-way handshake is never completed, and the entry remains in the connection queue until a timer expires, typically within about one minute. By generating phony SYN packets from random IP addresses at a rapid rate, it's possible to fill up the connection queue and deny TCP services to legitimate users.

2.4 Teardrop attack

Teardrop attack (Hoggan, 2000) targets a vulnerability in the way fragmented IP packets are reassembled. Fragmentation is necessary when IP datagrams are larger than the maximum transmission unit (MTU) of a network segment across which the datagrams must traverse. In order to successfully reassemble packets at the receiving end, the IP header for each fragment includes an offset to identify the fragment's position in the original unfragmented packet. In a Teardrop attack, packet fragments are deliberately fabricated with overlapping offset fields causing the host to hang or crash when it tries to reassemble them. Under normal conditions packet fragments will yield a positive integer value as can be derived from the diagram below.

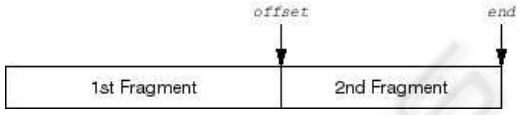

Figure 2: IP TearDrop Attack - Correct reassemble

However, the teardrop attack sends a fragment that deliberatley forces the calculated value for the end pointer to be less than the value for the offset pointer. This can be achieved by ensuring that the second fragment specifies a FRAGMENT OFFSET that resides within the data portion of the first fragment and has a length such that the end of the data carried by the second fragment is short enough to fit within the length specified by the first fragment. Diagramatically this can be shown as follows:

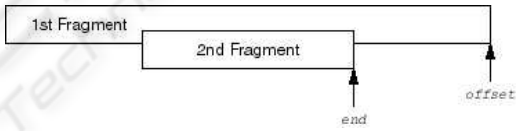

Figure 3: IP TearDrop Attack - Incorrect reassemble

When the IP module performing the reassembly attempts a memory copy of the fragment data into the buffer assigned to the complete datagram, the calculated length of data to be copied (that is the end pointer minus the offset pointer) yields a negative value. The memory copy function expects an unsigned integer value and so the negative value is viewed as a very large positive integer value. The results of such an action depends upon the IP implementation, but typically cause stack corruption, failure of the IP module or a system hang.

3 PACKET VERIFIER MODEL

The purposes of the packet verifier are validating compliance to standards, and validating expected usage of protocols e.g. protocol anomaly detection. The packet verifier checks the protocol header of packets, verifies packet size, checks TCP/UDP header

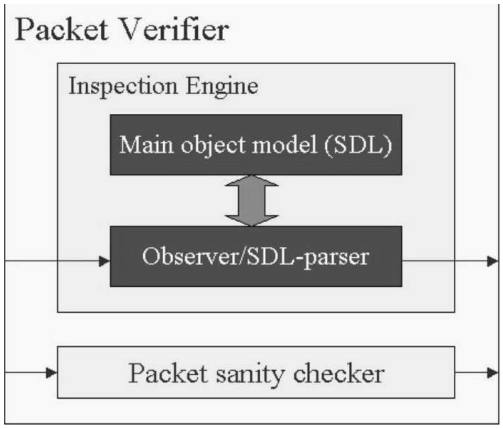

Figure 4: Components of the Packet Verifier

length, verifies TCP flags and all packet parameters, does TCP protocol type verification, and analyses TCP Protocol header and TCP protocol flags. In addition, the packet verifier contains an inspection engine like in Fig.4, including an observer and an main object model, to validate expected usage of protocols with SDL (the Specification and Description Language) (ITU-T, 1992) specifications. SDL specification defines the possible behaviours of protocols. While investigating the encoded packets, the observer/SDL-parser validates whether or not each sequence of packets follows what is required by the SDL specifications.

3.1 Protocol Specification

In our work, we have abstracted from our state machine specification (see Figure5) to capture only the essential details of TCP protocols. Using a more abstract specification, where the state machines accept a superset of what is permitted by the standards, and is still sufficient to deal with incomplete protocol runs meeting the standards (such as in the case of the SYN flood attack). We present a specification of the TCP state machine in this section. In order to achieve the goal of identifying TCP transition invariants for a set of reachable states, our method first searches through all the reachable states from the initial state to find the invariants. It then reduces the number of the searched states. As the number of the searched states reduced, the number of invariants increases.

A TCP connection is always initiated with the three-way handshake, which establishes and negotiates the actual connection over which data will be sent. The whole session begins with a SYN packet, then a SYN/ACK packet and finally and an ACK packet to acknowledge the whole session establishment. Our TCP specification is depicted pictorially in Figure 5. A new session starts in the LISTEN state. Data transfer takes place in the connection ESTABLISHED state. If the TCP connection is initiated from an external site, then the state machine goes through SYN_RCVD and ACK_WAIT states to reach the ESTABLISHED state. If the connection is initiated from an internal machine, then the ESTABLISHED state is reached through the SYN_SENT state. In order to tear down the connection, either side can send a TCP segment with the FIN bit set. If the FIN packet is sent by an internal host, the state machine waits for an ACK of FIN to come in from the outside. Data may continue to be received till this ACK to the FIN is received. It is also possible that the external site may initiate a closing of the TCP connection. In this case we may receive a FIN, or a FIN + ACK from the external site. This scenario is represented by the states FIN_WAIT_1, FIN_WAIT_2, CLOSING, TIME_WAIT_1, and TIME_WAIT_2 states. Our state machine characterizes receive and send events altogether to understand and to check properly. If the connection termination is initiated by an external host, note that the TCP RFCs do not have the states CLOSE_WAIT, LAST_ACK_WAIT, and LAST_ACK since they deal with packets observed at one of the ends of the connection. In that case, it is reasonable to assume that no packets will be sent by a TCP stack implementation after it receives a FIN from the other end. In our case, we are observing traffic at an intermediate node, e.g. firewall, so the tear down process is similar regardless of which end initiated the tear down. To reduce clutter, the following classes of abnormal transitions are not shown: conditions where an abnormal packet is discarded without a state transition, e.g., packets received without correct sequence numbers after connection establishment and packets with incorrect flag settings. Beause these parts will be checked by the SanityChecker.

3.2 Protocol SanityChecker

To cover other protocol aspects apart from TCP state specification, we are building a sanity checker. This performs layer 3 and layer 4 sanity checks. These include verifying packet size, checking UDP and TCP header lengths, dropping IP options and verifying the TCP flags to ensure that packets have not been manually crafted by a malicious user, and that all packet parameters are correct. In the IP protocol, according to the Internet Protocol Standard (RFC791, 1981), an IP header length should always be greater than or equal to the minimal Internet header length (20 octets) and a packet's total length should always be greater than its header length. IP address checks will also be important since land attacks use the same IP address for source and destination. According to the TCP standard (RFC793, 1981), neither the source nor the des-

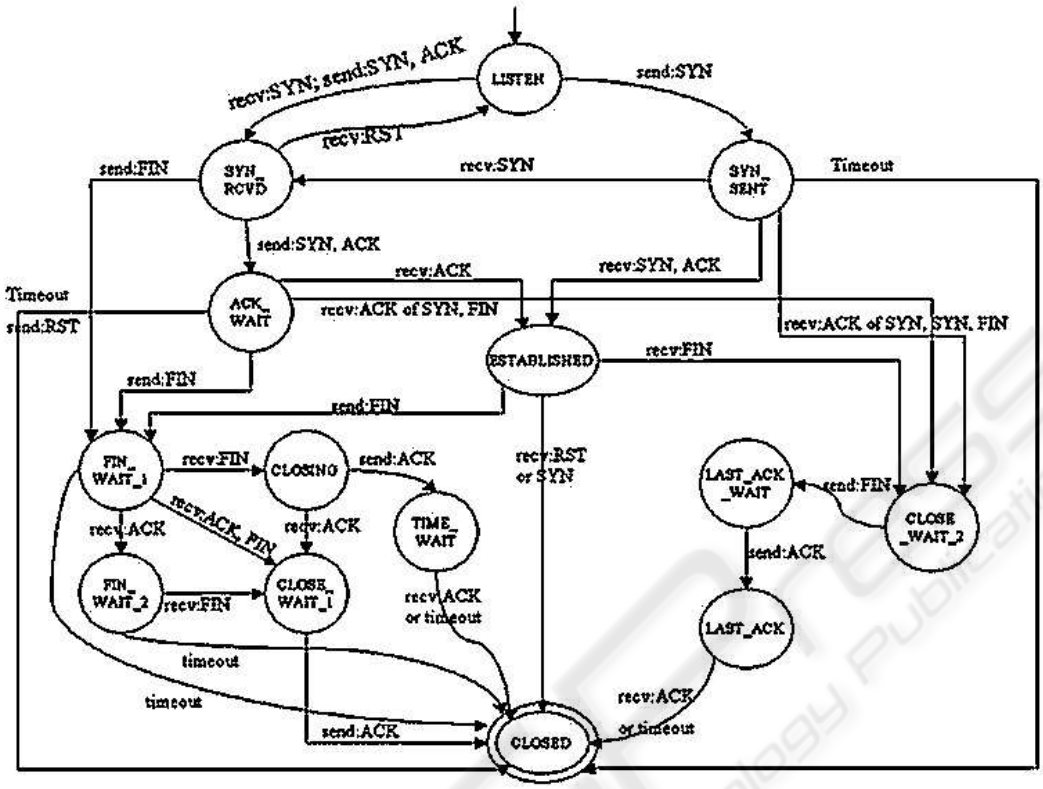

Figure 5: TCP Protocol State Machine

tination TCP port number can be zero, and TCP flags, e.g. URG and PSH flags, can be used only when a packet carries data. Thus, for instance, combinations of SYN and URG or SYN and PSH become invalid. In addition, any combination of more than one of the SYN, RST, and FIN flags is also invalid.

SanityChecker examines every packet within a 10 second window, and at the end of each window it will record any malicious activity it sees using syslog. SanityChecker currently detects some attacks; all TCP scans, all UDP scans, SYN flood attacks, Land attacks, and ping of death attacks. SanityChecker assumes any TCP packet other than a RST may be used to scan for services. If packets of any type are received by more than 7 different ports within the window, an event is logged. The same criteria are used for UDP scans. If SanityChecker sees more than 8 SYN packets to the same port with no ACK's or FIN's associated with the SYN's, a SYN flood event is logged. Any TCP SYN packets with source and destination address and ports the same is a identified as a land attack. If more than 5 ICMP ECHO REPLIES are seen within the window, SanityChecker assumes it may be a Smurf attack (CERT, 1998). Note that this is not a certainty. SanityChecker also assumes that any fragmented ICMP packet is bad, so this catches attacks such as the ping of death. To make the certatinty higher, this SanityChecker cooperateswith the protocol inspection engine with SDL. Furthermore, when the SanityChecker cooperates with the protocol inspection engine, this can work as a sequence verifier to matches the current TCP packet's sequence numbers against a state kept for that TCP connection. For example, in teardrop attack case, fragmented packets can be dealt with packet's IP id, and sequence. The inspection engine with SDL examines all packet transitions and remember IP id, and sequences, so the SanityChecker can detect reassemble problems with this engine.

4 GENERATING THE STATE MACHINE SPECIFICATION

We made the specification by hand, our next step is applying SDL to accomplish this specification. SDL is an International Telecommunication Union (ITU) standard, based on the concept of a system of Communicating Extended Finite State Machine (CEFSM)

Model (E.Hopcroft and D.Ullman, 1979). To understand how SDL can work based on the CEFSM, we address the dynamic semantics of the finite state machine, and SDL underlying model in this section.

4.1 Specification Development

We present how we specify TCP state transition with CEFSM in this section. A CEFSM is defined as a 6-tuple $<$ S, s_0, E, f, V, X $>$, as we mentioned above. Where, S is a set of states, s_0 is an initial state, E is a set of events with their parameter lists, f is a state transition relation. V is a set of local variables along with their types and initial values, if any. X is a set of signals. For a state, an input event, and a predicate composed of a subset of V, the state transition relation f has a next state, a set of output events and their parameters, and an action list describing how the local variables are updated. The purpose of SDL in our project to verify whether the TCP transition follows the standards. To do this, we made very simple TCP transition using SDL based on Figure 5. For TCP state transition, our CEFSM is like following:

- S = { listen, syn_rcvd, syn_sent, ack_wait, established, fin_wait_1, fin_wait_2, closing, close_wait_1, close_wait_2, time_wait, last_ack_wait, last_ack, closed }
- s_0 = listen
- E = { send(ip_id, flags), recv(ip_id, flags)}
- f : { f(listen, recv(ip_id, SYN), ip_seq_per_id = 0) -> (syn_rcvd, ip_seq_per_id = ip_seq_per_id + ip_seq, SYN, ACK), f(listen, send(ip_id, SYN), SYN, ip_seq_per_id != 0) -> (syn_sent, ,), ... }
- V = ip_seq_per_id, ip_seq
- X = ACK, SYN, FIN

In this SDL specification, Timeout, and other flags e.g., RST, PSH, URG are not included. Timeout and RST, PSH, URG flags can be dealt with low-level implementation part. To detect packet fragmentation, the SDL specification part can tell the packet sequence and proper flag, and low-level implementation part cooperate with this SDL, other flag combination, and timeout part. Figure 6. shows the StateTransition process which we built in SDL. Like SDL-GR(Figure 6.), sdl-PR is also in [Table 1].

Table 1: Part of SDL-PR Source in the process StateTransition

```
PROCESS StateTransition ;
NEWTYPE PacketInfo
STRUCT
ip Integer;
seq Integer;
flag TCPFlags;
OPERATORS
Unexpected: Integer, TCPFlags -> PacketInfo;
SanityCheck: Integer, Integer -> PacketInfo;
ENDNEWTYPE;
DCL pkt PacketInfo;
DCL tcp_id, tcp_seq, tcp_seq_per_id Integer := 0;
DCL tcp_flag TCPFlags;
DCL cur_process PId; /* current process */
Timer t;

START;
NEXTSTATE idle ;
STATE syn_sent ;
INPUT Packet(tcp_id, tcp_seq, tcp_flag) ;
TASK pkt := SanityCheck(tcp_id, tcp_seq) ;
DECISION tcp_flag ;
( ACKFIN ):NEXTSTATE close_wait_2 ;
( ACKSYN ):NEXTSTATE established ;
( SYN ): NEXTSTATE syn_rcvd ;
ELSE: TASK pkt := Unexpected(tcp_id, tcp_flag) ;
NEXTSTATE - ;
ENDDECISION;
ENDSTATE;
STATE syn_rcvd ;
INPUT NONE;
OUTPUT ROP(tcp_id, ACKSYN) to SENDER ;
NEXTSTATE ack_WAIT ;
ENDSTATE;
```

5 CONCLUSION

We have discussed protocol anomalies and address a packet verifier model. The purposes of the packet verifier are to validate compliance to standards, and to validate expected usage of protocols, especially protocol anomaly detection. Considering performance in real-system, we are implementing a SanityChecker to cover protocol header anomalies, and using SDL, we specify TCP transition behaviours, the specification will then be compiled into an inspection engine program for observing packets. At the moment, we specified TCP protocol transition. Through this state machine, we will implement the inspection engine. This will be our future work. We believe that this protocol anomaly analysis and the packet verifier model will be useful to detect protocol anomalies and verify proper usage of protocols.

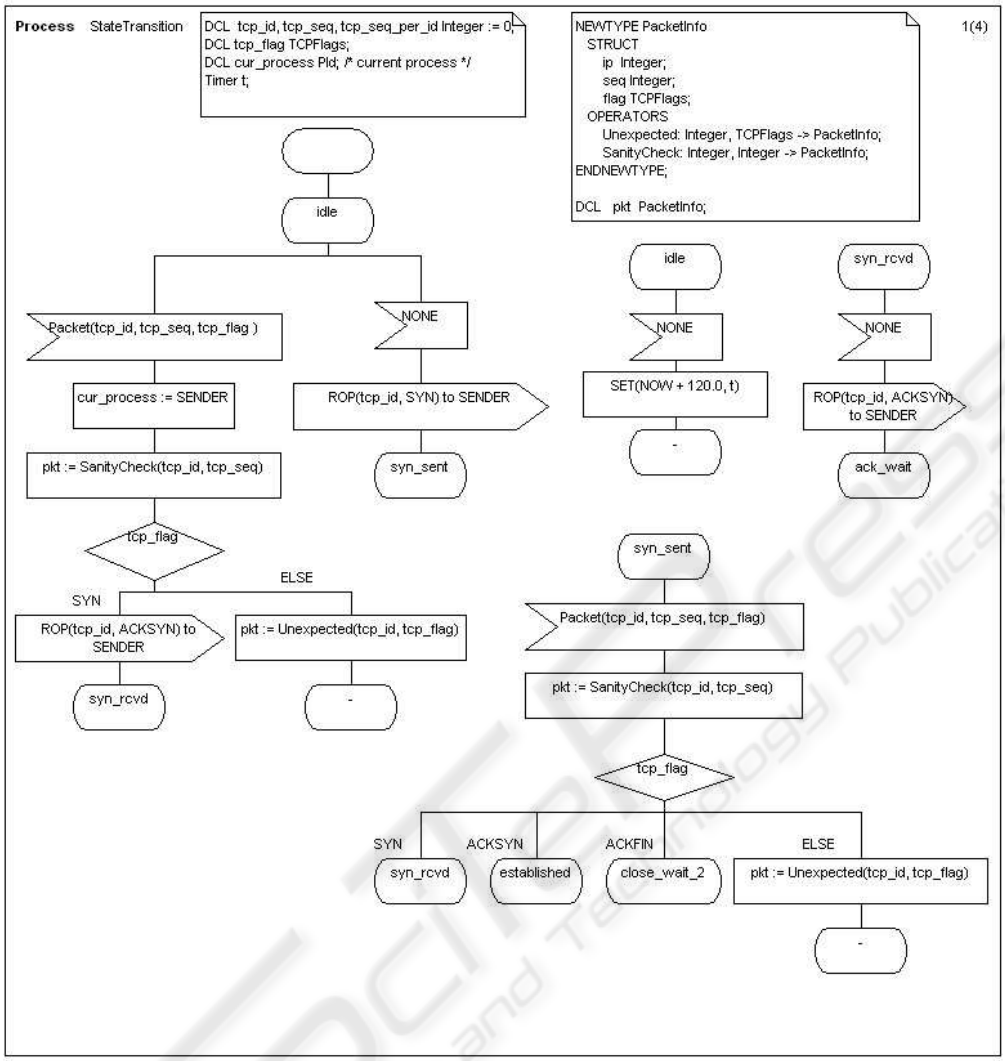

Figure 6: Process StateTransition of the TCP Protocol State Machine in SDL

REFERENCES

CERT (1998). Advisory ca-1998-01 smurf ip denial-of-service attacks. In *Online Publication*.

CERT/CA-1996-21 (2000). Advisory ca-1996-21 tcp syn flooding and ip spoofing attacks. In *Online publication*.

CISCO (1997). Security advisory: Tcp loopback dos attack (land.c) and cisco devices.

E.Hopcroft, J. and D.Ullman, J. (1979). *Introduction to Automata Theory, languages, and computation*. Addison Wesley.

Fyodor (1996). Ping of death attack. In *INSECURE.ORG*.

Fyodor (1997). The land attack(ip dos). In *INSECURE.ORG*.

Hoggan, D. (1994-2000). Teardrop attack. In *The Internet Book: Introduction and Reference*.

ITU-T, C. (1992). *Recommendation Z.100: Specification and Description Language (SDL)*. General Secretariat, Geneve, Switzerland.

RFC791 (1981). Internet protocol. In *DARPA Internet Program Protocol Specification*.

RFC793 (1981). Transmission control protocol. In *DARPA Internet Program Protocol Specification*.