Taking over malicious connection in half way by migrating protocol state to a user-level TCP stack

Qi Tang, Chao Zheng, Qiuwen Lu, Wei Yang
National Engineering Laboratory for Information Security Technologies
Institute of Information Engineering, CAS
Beijing, China
Email: {tangqi, zhengchao}@iie.ac.cn

Qingsheng Yuan, Xunxun Chen
National Computer Network Emergency Response Technical Team/Coordination Center of China
Beijing, China
Email: qingshengcn@163.com

***Abstract*—Network intrusion detection system (NIDS) takes necessary measures when detecting threats. Since most of the malicious contents like phishing sites and advanced persistent threats are transmitted on transmission control protocol (TCP), existing measures are usually injection-based, such as injecting a reset (RST) packet to terminate the connection or a HTTP 302 response to redirect users' requests. Injection is a feasible measure but is unable to scrub traffic like removing malicious contents. Therefore, taking over malicious TCP connections instead of injection is a more effective solution for NIDS. In this paper, we propose an efficient and flexible solution to take over malicious connections selectively at any period of the connections combining with two typical deployments of NIDS. The NIDS usually works as a passive protocol analyzer to gain high performance, when malicious contents are detected, it will migrate TCP states to a user-level TCP stack and work as a transparent proxy. The migration to user-level TCP stack is flexible and graceful due to bypassing the complexity and overhead of OS TCP stack. To evaluate our approach, we elaborate an experiment to compare with the migration to OS TCP stack. The result shows that the response speed of our approach is 8x faster than the OS stack, and more stable.**

Keywords—Network intrusion detection system; Take over; TCP state Migration; Finite state machine; User-level TCP stack;

I. INTRODUCTION

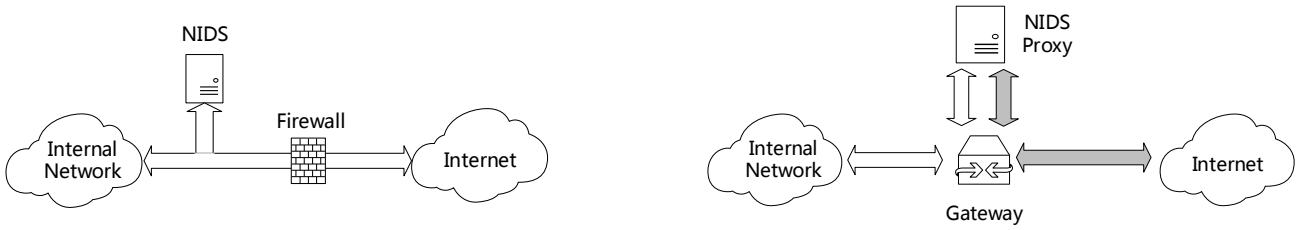
Companies and organizations have deployed increasingly network intrusion detection system (NIDS) to protect their digital property and infrastructure from malicious content [1], [2]. When a threat is detected in network traffic, NIDS will take the necessary measures to interrupt the connections. The measures are varied based on the deployment method of NIDS. The typical deployment is with an optical splitter or a firewall, and capture and analyze network traffic in real-time. Since most malicious content like phishing sites and advanced persistent threats are transmitted on TCP, NIDS could adopt different approaches to deal with different kinds of threat, these approaches including removing malicious content, warning user or just terminating the connection.

Terminating the connection is usually the simplest approach to avoid further damage. It can be achieved by dynamically configuring the firewall to discard packets or sending RST packets. But blocking is rude because it is unable to notify the user, users could not distinguish blocking from a network failure. Another approach is sending a HTTP 302 response, then the phishing site was redirected to a warning page. This is a feasible solution but it is not suitable for content based detection. Also, it will not work on other application layer protocols, e.g. file transfer protocol (FTP). Therefore, taking over malicious TCP connection instead of injection becomes a much better solution for NIDS.

Generally speaking, NIDS has two typical deployment methods, as a (1) passive protocol analyzer or a (2) transparent proxy. Passive protocol analyzer [5], [6], [7], [8], [13] is widely used for its high performance without decreasing reliability of current network. It traces and TCP states passively by a simple finite state machine. Limiting by lack of the complete TCP stack, existing NIDS in this mode can only inject a few packets. Transparent proxy [3], [4] is working as a middlebox in the physical network. It has a complete TCP/IP stack to communicate with both sides separately, and it could take over bidirectional connections transparently. Transparent proxy's performance is far lower than the passive protocol analyzer, due to the extra cost of taking over every TCP connection [13]. The cost includes timers, data buffers, complicate finite state machine and so on. For a transparent proxy working on the basis of OS TCP/IP stack, select the connections to take over based on the contents is challenge. Because malicious content is transferred in an established connection and migrating an established connection to OS stack will introduce considerable overhead.

The goal of our work is to find an efficient and flexible solution to take over malicious connections in NIDS. We explore an alternative approach that only takes over malicious connections to protect users. Combining with previous constrains, our approach is designed for two explicit goals: 1) High efficiency: taking over connection selectively instead of indiscriminately, 2) High flexibility: taking over connection at any period, no matter the connection is establishing or established, the content is a request or a response.

Our key approach is to let the NIDS working as a passive protocol analyzer to gain high performance, once malicious contents are detected, for example a phishing URL, it'll take over the malicious connection by migrating TCP states to a user-level TCP stack. A TCP connection is two sets of sending and



(a) Passive Protocol Analyzer

(b) Transparent proxy

Fig. 1. Different types of NIDS based on deployment: (a) Passive protocol analyzer monitors traffic passively and detects threats. (b) Transparent proxy replaces users to request and detects application data.

receiving states, which contains SEQ, ACK etc. With the help of user-level TCP stack, we bypassed the OS stack's complexity and overhead, NIDS obtained a great deal of flexibility to manipulate the TCP states. The takeover procedure is graceful and reliable. For the malicious connection, the NIDS is a transparent proxy, and now it can take the necessary measures such as responding a warning page to the user or removing malicious JavaScript from the web page through revising sequence numbers.

To evaluate our approach's correctness and performance we also elaborate an experiment of migrating to OS TCP stack. The result shows that the response speed of our approach is 8x faster than the OS TCP stack, and is more stable.

The key contributions of this paper including:

- We propose a flexible approach of migrating TCP states to a user-level TCP stack straightly by reverting a complete TCP control block (TCB) with only a few states. This approach can bypass the complexity and overhead of migration to OS stack.
- We design and implement an NIDS which can take over malicious connections selectively at any period and take appropriate measures to block threats. The NIDS is efficient and flexible by combining the advantages of passive protocol analyzer and transparent proxy.

The remainder of this paper is organized as follows. Section II describes background information on two different NIDSs. In section III we discuss the challenges to achieve our goals. Section IV explained the design and implementation of our NIDS. In section V, we construct a contrast experiment to evaluate our approach. Section VI introduces the related works about existing approaches of preventing malicious connections. In section VII, we make a conclusion.

II. BACKGROUND

Based on deployment methods NIDS can be classified into two categories namely: Passive protocol analyzer and Transparent proxy.

A. Passive protocol analyzer

Passive protocol analyzer which is shown in Fig. 1(a) is widely used for its high performance without decreasing reliability of network [5], [6], [7], [8], [13]. Passive protocol analyzer gets network traffic by a network splitter or a sensor. It does not involve in or interfere normal communication. In contrast, it only monitors network traffic passively and inspects malicious contents. Since many attacks escape detection by fragmenting IP packets and segmenting TCP flows [13], [15], passive protocol analyzer associates sessions instead of inspecting packet singly in order to improve detection ability. This NIDS will reassemble IP fragments and reconstruct TCP flows by tracing contexts of sessions and analyzing states of TCP.

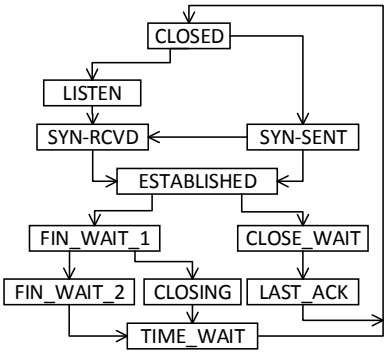
Finite state machine (FSM) is often used for describing protocols. It can easily express design criteria of protocols in terms of desirable and undesirable protocol states and state transitions. According to RFC 793 [11], there are eleven states during a TCP connection's lifetime. These states are LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. The state transition of TCP is illustrated with Fig. 2(a).

Likewise, FSM is also used in passive protocol analyzer to trace and analyze states of TCP [14]. Considering that the main purpose of NIDS is to detect contents in transit. Thus, states after active close and passive close will not be traced by passive protocol analyzer. A typical FSM adopted by passive protocol analyzer such as [7] is shown in Fig. 2(b). There are only four states in it. Benefiting from incomplete implementation of TCP stack and simple finite state machine, passive protocol analyzer is usually high-performance and dose not decrease reliability of network.

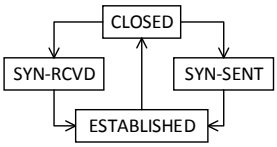
B. Transparent proxy

Transparent proxy is a typical middlebox in physical network [3], [4]. It has a complete TCP/IP stack to communicate with both sides separately, and it could take over bidirectional connections transparently. All requests from users will be serviced by it. The major process is shown in Fig. 1(b). When a user makes a request to a target server, the transparent proxy acts as the target server and establishes a TCP connection with the user. On the other side, transparent proxy will make a new TCP connection with target server itself. Upon receiving responses from target server, it will inspect the contents and transmit it back to user on the previous TCP connection.

Differing from passive protocol analyzer, transparent proxy will not reassemble packets on itself which are below the application layer for detection. This is completed by TCP/IP stack who bases on. Therefore, we consider that the FSM of transparent proxy is the same as TCP's which is shown in Fig. 2(a).



(a) TCP finite state machine



(b) Libnids's finite state machine

Fig. 2. Finite state machine for different types of NIDSs

TCP is a full-featured, connection-oriented and reliable transport protocol for TCP/IP applications. Many mechanisms are utilized to ensure the efficiency and reliability including sliding window, data buffers, timers, acknowledge mechanism, congestion control, etc. These complex functions introduce additional cost to transparent proxy [13]. Along with intricate FSM, the overhead of taking over each connection is huge. Therefore, performance of transparent proxy based on complete TCP/IP stack is far lower from passive protocol analyzer.

III. Challenge

The challenges of taking over malicious connections arise from two explicit goals:

High efficiency: Efficiency is an important indicator for NIDS especially in high speed network environment. Malicious connections may only account for a small part of all connections. It would be unbearable to taking over whole TCP connections for NIDS especially in backbone network which enjoys enormous traffic. Therefore, selective taking over mechanism should be applied.

High flexibility: NIDS not only recognizes a malicious connection by detecting user's request such as URL but also detects contents of server's responses. Malicious contents may appear in half way of a connection. Thus, NIDS must be able to recognize malicious contents before taking over this connection. In other words, NIDS could take over malicious connection at any period, no matter the connection is establishing or established, the content is a request or a response.

Although passive protocol analyzer has high performance, it could only terminate connections and redirect malicious requests by injection rather than taking over them due to lacking of the full features of TCP. Therefore, the takeover procedure must be completed with transparent proxy mode which has a complete TCP/IP stack. For a transparent proxy working on the basis of OS TCP/IP stack, selecting the connections to take over based on the contents is a great challenge. This is because malicious content is transferred in an established connection and migrating an established connection to OS stack will introduce considerable overhead.

IV. Design And Implementment

On the basis of the previous discussion, we now describe the design and implementation of NIDS with ability of taking over malicious connections selectively in half way.

A. Overview

The key approach is to let the NIDS working as a passive protocol analyzer to gain high performance, and taking over the malicious connection as a transparent proxy. Fig. 3 shows the overview of the NIDS. Extra modules and methods are utilized to achieve the migration. There are five major modules in the NIDS:

Capture Module: Capture module is used to collect network traffic. It is the entrance to the entire system. We use DPDK [9] to implement the capture of network traffic in consideration of its high performance.

Protocol Analysis Module: As mentioned previously, passive protocol analyzer will reassemble IP fragments and reconstruct TCP flows for deep inspection. That's what protocol analysis module does. Protocol analysis module adopts the FSM with four protocol states which is shown in Fig. 2(b) and traces the state transitions of sessions. In order to help to take over malicious connections later, protocol analysis module will trace extra set of states for each connection. The details will be discussed later.

Detection Module: Detection module detects the contents of connections reassembled by protocol analysis module. Some detection algorithms are utilized along with user-defined rules.

Alarm Module: When malicious contents or threats are detected, alerts need to be sent to the network administrator and monitoring center by many different ways. Alarm module can show the malicious events in different forms.

Proxy Module: Proxy module is an extra module which is a biggest difference from normal passive protocol analyzer. For malicious connections, the NIDS is a transparent proxy. The malicious connections will be taken over by proxy module so that NIDS could take appropriate measures to protect users. We implement a user-level TCP stack and restore the contexts of the malicious connections with it. By migrating from FSM of passive protocol analyzer to FSM of TCP stack, proxy module can take over malicious connection selectively in half way.

In this paper, we focus on protocol analysis module, proxy module and the migration between them. Other modules are beyond the scope.

B. State Migration

In essence, a TCP connection is two sets of sending and receiving states. TCP control block(TCB) [11] is the commonly used data structure to describe the states. It holds almost all the information about a connection' current contexts including TCP state, four-tuples (source IP address and port as well as destination IP address and port), variables of sending sequence

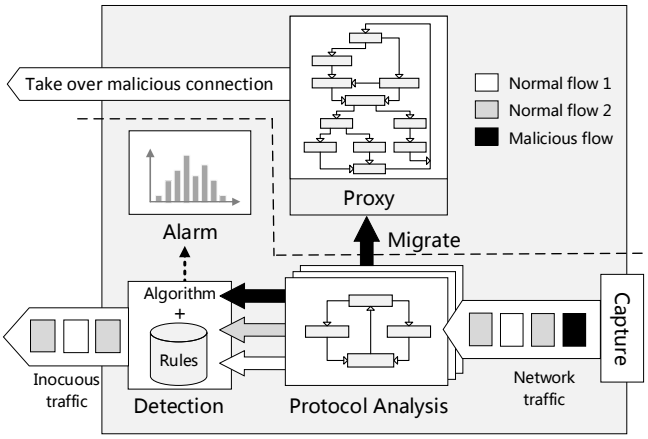


Fig. 3. Workflow of the NIDS with TCP state migration

and receiving sequence, TCP timers, variables of congestion control, variables of TCP options including max segment size, window scaling, timestamp, SACK, etc. In some ways, a TCB represents a certain connection.

It is a challenge to migrate an established TCP states to the OS TCP/IP stack. Because TCB could only be created by procedure of three-way handshake in OS stack. This will introduce considerable overhead. Therefore, method of state migration to a user-level TCP stack is utilized. We bypassed OS stack's complexity and revert a TCB with a concise set of states straightly. There are two steps to achieve the migration including:

Tracing states: When working as a passive protocol analyzer, protocol analysis module will trace a concise set of TCP states for each connection. Since protocol analysis module need to reassemble TCP flows, the contexts could be easily achieved. For each connection, the states shown in Table I are traced. Among these states, *client_seq*, *client_ack*, *server_ts* and *server_echo_ts* need to be updated in real time. The other states will remain their value constantly after the procedure of three-way handshake. The detailed meanings of these states are also explained in Table I.

These traced states occupy only little memory space for each connection without introducing overhead of complex calculations. Therefore, impact of tracing states in protocol analysis module on performance can be ignored.

Reverting TCB: In order to bypass the procedure of three-way handshake which is inevitable for OS TCP/IP stack, the approach of reverting an established connection straightly with a user-level TCP stack is proposed. This approach makes use of the states traced in previous step, reverts the structure of TCB and then adds it into the TCP flow table which is used to manage all TCP connections. The revert of TCB includes:

1) Four-tuples: They are reverted to the traced TCP states previously. Noting that the source IP address and port must be the same with the malicious server's, and the destination IP address and port are the same with client's.

2) Variables of sending sequence: According to RFC 793 [11], some pointers are used to describe sending window:

TABLE I. STATES TRACED BY PROTOCOL ANALYSIS MODULE

States	Meaning
four-tuples	Source IP address and port as well as destination IP address and port(Source means malicious server end)
client_seq	Sequence number of latest packets from client
client_ack	Acknowledge number of latest packets from client
eff_mss	Effective MSS after consultation of client and server.
SACK_support	Whether both side open SACK option
server_wscal	Window scaling option of server
client_wscal	Window scaling option of client
server_ts	Latest timestamp from server
server_echo_ts	Server echo client's timestamp

SND.UNA represents the first byte which has not been acknowledged by peer, SND.NXT represents the first byte of next sending and SND.WND represents the size of sending window. In this scenario, we use traced state *client_ack* which indicates the next byte client wants to receive to revert SND.UNA and SND.NXT. SND.WND can be simply reverted to default value of the user-level TCP stack. It has no effect on recovery of connection benefiting from flow control and congestion control later.

3) Variables of receiving sequence: Exactly as sending sequence, there are also two pointers to describe receiving window: RCV.NXT represents the next byte TCP stack wants to receive and RCV.WND represents the size of receiving window. Similarly, RCV.NXT can be reverted to traced state *client_seq* representing the latest sequence number sending by client. RCV.WND is also reverted to default value.

4) TCP options negotiated in three-way handshake: Some TCP options have been negotiated during three-way handshake. MSS of connection which represents the maximum segment size to send is the minimal value of client's and server's advertised MSS. It can be reverted to the value *eff_mss* traced previously. Window scaling options of both side can be reverted to *server_wscal* and *client_wscal* separately as well. There are also another variable in TCB indicates whether SACK option can be utilized in this connection. Only if both side open SACK permit option can this connection use. This variable has already been calculated and saved in states *SACK_support*, too.

5) TCP options after negotiation: Other TCP options appear in packets after connection has already been established. Timestamp option is a common option with many purposes. It can help measure round trip time of a connection and protect against wrapped sequence numbers. Thus, revert of timestamp is necessary. There are two fields in timestamp option: One is used to echo peer's timestamp. It is saved by variable TS.RECENT in TCB which means the latest timestamp of peer. It should be equal to *server_echo_ts* in Table I. Another filed is timestamp of current end. Usually, this is a global value for a TCP stack and different implementations of TCP stack share different values. However, our user-level TCP stack must be

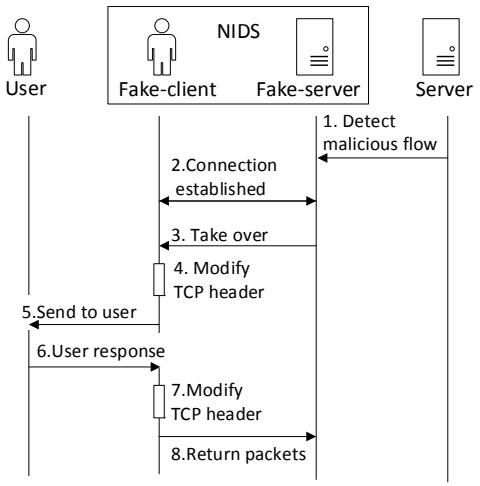


Fig. 4. Approach and sequence of migration to OS TCP stack

able to simulate behaviors of different TCP stacks. So another variable is added to save simulated server's timestamp in TCB. This variable is equal to *server_ts* in Table I and will be increased every 10msec according to RFC 1323 [12]. Another option is SACK. It indicates discontinuous packets receiving from peer. In this scenario, receiving buffer of the connection can be considered empty. Thus, SACK option will be calculated during following transmission of the connection without revert.

6) Other variables: Structure of TCB is huge and there are many other variables to manage the behaviors of connections. These variables can be reverted to default value when TCP connection is establihed by three-way handshake normally such as congestion window, slow start threshold, TCP timers, variables for calculating RTT, etc. They shall have no effects on the normal communication of the connection after migration.

By assigning appropriate values with a concise set of TCP states, huge structure of TCB could be reverted straightly in a user-level TCP stack. This approach is quick without OS stack's complexity and overhead.

In addition, since different versions of OS stack offer varying degrees of TCP feature support, e.g. SACK, timestamp, etc. It is an undefined behavior to take over connections enabling SACK by migrating to OS stack which does not support it instead. On the contrary, our user-level TCP stack can revert the actual behaviors of varying TCP stacks, it is more graceful and reliable.

V. Experiment

In this section, we will construct experiments to evaluate the correctness and performance of our approach.

The experiment is designed as follows. Test client visits a given phishing site through a browser and it generates a HTTP GET request. The NIDS then recognizes the URL in the header of the HTTP GET request and considers it is malicious according to the configurations. At last, the NIDS takes over the malicious connection and send warning webpage back to test client. The NIDS is deployed serially between a test client connected via a direct cable and a web server to be visited. The NIDS machine employs Linux 2.6.32 with 8G RAM, Intel Corporation 82574L Gigabit NIC and a quad-core Intel(R) Xeon(R) E31225 @ 3.10GHz CPU.

A. Contrast experiment

We construct an elaborate approach of taking over malicious connections by OS stack for purpose of comparing to migrating to user-level TCP stack. Limiting to the low scalability of OS TCP stack, third party libraries like DPDK [9] and Libnet [10] are utilized.

To migrate to OS TCP stack, the fake server and fake client are constructed. The fake server is created by Socket and listens to a certain port. Then the fake client forged by DPDK [9] sends an SYN packet to the fake server to start three-way handshake and establish a new TCP connection. When the connection is established, the migration is done. All packets from fake server to fake client will be modified to valid IP address and sequence number by fake client and then transmitted to users. All packets from users will be modified to valid values by fake client and sent back to fake server. The modification of IP address and sequence numbers are also base on the states shown in Table I traced by passive protocol analyzer. In this way, the NIDS could work as a transparent proxy and take over the connection. The detailed steps of this method are explained as follows, which is also shown in Fig. 4.

(1) NIDS detects malicious contents from the server as a passive protocol analyzer and is ready to take over this connection.

(2) Fake client forged by DPDK [9] establishes a new TCP connection with fake server through simulating three-way handshake.

(3) Fake sever takes over malicious connection, it takes appropriate measures to protect user such as sending warning page.

(4) Fake client captures the packet from fake server by DPDK [9] and modifies the IP addresses in IP headers as well as sequence number, acknowledge number and ports in TCP headers.

(5) Fake client transmits these modified packets to user by Libnet [10].

(6) Packets replied from users are captured by fake client through DPDK [9], too.

(7) Fake client modified TCP and IP headers of captured packets contrarily to step (4).

(8) Fake client feeds these modified packets back to fake server.

The following communications between NIDS and user cycle through step (3) and step (8).

B. Response delay

To measure the overheads of migration and takeover, we focus on the delays of response. The response delay is defined as the time between malicious contents are found and concrete measures are taken.

In the experiment scenario, the transmission delay and detection delay could be ignored. Thus, response delay can be

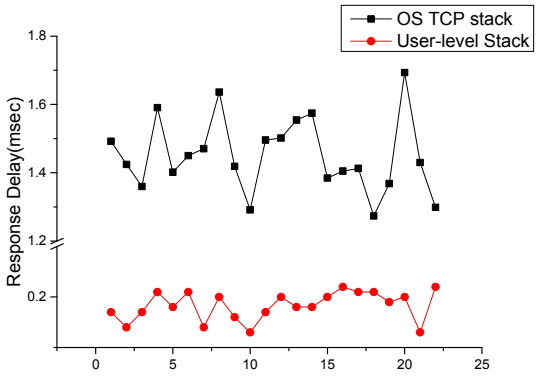


Fig. 5 Distributions of response delays for 22 requests

TABLE II. ANALYSIS ON RESPONSE DELAY

Methods	Response Delay(msec)			
	Min	***Max***	***Mean***	***Std. Dev***
OS TCP stack	1.27	1.69	1.45	0.11
User-level stack	0.12	0.22	0.18	0.02

calculated by the time of requesting and the time of receiving first warning packets by test client.

Both migrating to user-level TCP stack and migrating to OS TCP stack are under test separately. The result is shown in Table II. The mean response delay of migrating to OS TCP stack is 1.45msec while the delay of migrating to user-level TCP stack is only 0.18msec and is 8x faster than OS stack. The standard deviation of migrating to user-level TCP stack's delay is less than OS stack's. The distribution of the response delay is also shown in Fig. 5. Response delays of migrating to user-level TCP stack has less jitter and is more stable. The reason of this phenomenon is that migration to user-level TCP stack reverts an established TCP connection straightly and saves the process of three-way handshake, recapture and modification of packets.

Above all, migrating to user-level TCP stack is a more flexible and stable approach. It could bypass the complexity and overhead of OS stack, manipulate TCP states flexibly. The entire takeover is graceful and reliable.

VI. RELATED WORK

Countless NIDSs have been developed and deployed in the past few decades. Some NIDSs including [5], [6], [7], [8], [13], etc. are passive protocol analyzers. These NIDSs get network traffic by network splitters or sensors. Since attackers can fragment the attack signature into several TCP packets in order to evade detection of NIDS [15], these NIDSs trace the TCP states and reassemble TCP flows with a simple FSM. For example, [5] uses a component named preprocessor to reassemble TCP flows for later detection [2]. [6] will order raw packets into streams of data for application-layer protocol analysis [2]. [8] completes this process through stream reassemblers. Passive protocol analyzers are often high-performance without complex TCP FSM, timer events, etc. [13], but it is difficult to take over connections with them.

Other NIDSs including [3], [4], etc. are transparent proxies. [3] is an intrusion protection system based on P2P and reverse proxy architecture for web security. [4] is a centralized TCP architecture in which an organization's edge router can transparently proxies every TCP connection to protect network. Both of them only concern about application-layer data reassembled by OS stack. Transparent proxies take over entire traffic at the beginning and take appropriate measures such as blocking, injection, and scrubbing traffic to stop the threats.

VII. CONCLUSION

We implement an NIDS which can protect user by takeover malicious TCP connections when threats are detected, and the approach of takeover is efficient and flexible. For the most harmless connection, NIDS works as a passive protocol analyzer to gain high performance, once malicious contents are detected, it takes over the malicious connection by migrating TCP states to a user-level TCP stack. NIDS traces each TCP connection with a concise set of states under passive protocol analyzer mode, and revert to a TCB in user-level TCP stack. In this way, NIDS can take over malicious connections selectively. In addition, the migration is straight without the three-way handshakes. To evaluate our approach's correctness and performance we also elaborate an experiment of migrating to OS TCP stack. The result shows that the response speed of our approach is 8x faster than the OS TCP stack. And the response delay of state migration to user-level TCP stack is more stable.

REFERENCE

[1] Liao, Hung-Jen, et al. "Intrusion detection system: A comprehensive review." Journal of Network and Computer Applications 36.1 (2013): 16-24.

[2] Rai, Kajal, and M. Shyamala Devi. "Intrusion detection systems: A review." Journal of Network and Information Security Volume 1.2 (2013).

[3] He, Qian, et al. "P2PRPIPS: A P2P and Reverse Proxy Based Web Intrusion Protection System." (2013).

[4] Hsu, F-H., and Tzi-cker Chiueh. "CTCP: A transparent centralized TCP/IP architecture for network security." Computer Security Applications Conference, 2004. 20th Annual. IEEE, 2004.

[5] Roesch, Martin. "Snort: Lightweight intrusion detection for networks." Lisa. Vol. 99. No. 1. 1999.

[6] Paxson, Vern. "Bro: a system for detecting network intruders in real-time." Computer networks 31.23 (1999): 2435-2463.

[7] Libnids: http://libnids.sourceforge.net

[8] Kruegel, Christopher, et al. "Stateful intrusion detection for high-speed network's." Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on. IEEE, 2002.

[9] DPDK: http://dpdk.org

[10] Libnet: http://libnet.sourceforge.net

[11] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, 1981

[12] Jacobson, V., Braden, R., and D.Borman, "TCP Extensions for High Performance", RFC 1323, 1992

[13] Watson, David, et al. "Protocol scrubbing: network security through transparent flow modification." IEEE/ACM Transactions on Networking (TON) 12.2 (2004): 261-273.

[14] Barry, Bazara IA, and H. Anthony Chan. "A Cross-protocol approach to detect TCP Hijacking attacks." Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on. IEEE, 2007.

[15] Ptacek, Thomas H., and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. SECURE NETWORKS INC CALGARY ALBERTA, 1998.