ROOK: Multi-Session based Network Security Event Detector

Masayoshi Mizutani, Shin Shirahata, Masaki Minami
Keio University
Graduate School of Media and Governance
Endo 5322, Fujisawa-shi
Kanagawa 252-8520, Japan
{mizutani,true,minami}@sfc.wide.ad.jp

Jun Murai
Keio University
Faculty of Environmental Information
Endo 5322, Fujisawa-shi
Kanagawa 252-8520, Japan
jun@wide.ad.jp

Abstract

We have implemented Multi-Session based Network Security Event Detector: **ROOK** *to detect botnet activity and P2P file sharing traffic and our results show that our method is less false positives than existing network security event detectors (e.g. IDS). We proposed a network security event detection method by analyzing correlation among multiple sessions. Our method can recognize hosts behaviors by rules that describe multi-session correlations: a rule includes the order of starting sessions and information exchange between sessions. By this method, ROOK detected DNS and IRC activities of bots in the experiment.*

1 Introduction

Network security operators need to take measures to respond to incidents by malware (e.g. computer viruses, spyware and bots) and need to detect policy violations (e.g. using file sharing programs). Network security event monitoring and detection is an effective method for early incident detection and response. Network-based Security Event Detectors (SED, e.g. IDSes) are deployed in many networks for incident response.

However, the competition between malware authors and system defenders is fierce and continuous; since SED have become widely deployed, malware authors have attempted to evade detection using two methods. The first method is disguising malware traffic as innocent traffic. This method is adopted by malwares. Many malware activities[1] are similar to those of non-malicious software, so huge numbers of false positives are generated when trying to detect the malware traffic. The second method is using encryption. The software encrypts not only the contents but also the protocol headers to avoid detection.

A new method of security events detection is needed because it is difficult to detect security events of the software by existing SED. Existing SED's methods mostly use features of a single session and a single packet as detection triggers. Some SED systems operate differently. However the methods have problems distinguishing between malwares and P2P file sharing programs. For example, anomaly detection[2, 3] is able to detect malware activities. However, this method is less accurate and needs hand tuning. Traffic classification[4, 5, 6] is an effective method for classification of software category because its original goal is traffic engineering. However it difficultly specifies software details (e.g. software name). Less-accurate SED's methods increase network security operators' cost by creating many false positives. Correlation analysis[7, 8] is able to detect security events by flexible and complex rules. Therefore most Security Events Manager(SEM)[9, 10] that is implementation of correlation analysis needs huge logs of security events. As a result, huge amounts of storage are needed to operate SEM.

The goal of our research is development of a system to detect network security events in disguised traffic and encrypted traffic. We propose a multi-session based SED approach[11] to resolve this problem. A *Session* is defined a communication for one purpose in this paper. (e.g. a HTTP session, a pair of DNS query/response, a pair of ICMP request/response) Existing methods difficulty figure out software's behavior because of focusing on only contents of a single packet and a single session. This approach focuses on correlation of the multiple sessions. We define correlation of a multi-session as the order of the session appearances and cross-reference of sessions' contents. We designed and implemented multi-session based SED: *ROOK* and did an experiment using this implementation on a live network. The experiment shows decreasing false positives by the multi-session based approach.

2 Multi-Session based Detection

We propose multi-session based SED's method for detecting security events by disguised traffic and encrypted traffic. A concept of this method is enhanced signature-based IDSes that focuses on correlations of multiple sessions. It is hoped that our methods can accurately detect variant and complex security events by focusing on correlations of multiple sessions,

2.1 Requirements

We evaluate our system based on now well it meets four requirements. Evaluation of our system is on going; performance has not yet been measured.

- **Accuracy**: Security events that are detected by our method should be kept at the minimum false positives and false negatives.
- **Flexibility**: Detection method must be enough flexible to detect various security events. Malwares keep raising and attack type is diversified into various kinds.
- **Scalability**: This means implementation performance. The implementation monitors live network traffic. Therefore high speed traffic decoding and detecting function and handling huge sessions are necessary.
- **Memory Usage**: This requirement is similar to scalability. The implementation must operate using minimal memory because computer resources are limited.

2.2 Approach

Our approach attempts to follow the network behavior of software that uses disguised traffic and encrypted traffic. Existing SED methods can recognize only a fraction of the illicit activity on the network because they handle only a single packet and a single session in their detection procedures. Both malware and P2P file sharing programs typically communicate with a number of different hosts using multiple sessions. The multiple sessions are often related. The software communicates according to pre-defined algorithms, based on results from prior communication (e.g. DNS requests). Each program exhibits a unique signature in the pattern of communication sessions it follows. Therefore, it is possible to identify which software is present by monitoring the features of multiple sessions appearing on the network.

We defined that *multi-session correlation* is two relations: order of session appearances and information exchange between sessions. Order of session appearances means sequences and repeats of multiple session appearances. Information exchange between sessions means using contents of other sessions for detection some sessions. Our method detects security events using multi-session correlations and existing IDS's functions: pattern matching and checking protocol header and so on. Multi-session means not only multiple sessions between a pair of hosts but also point-to-multipoint sessions. Section 2.3 describes two kinds of relations by effective cases at using our method.

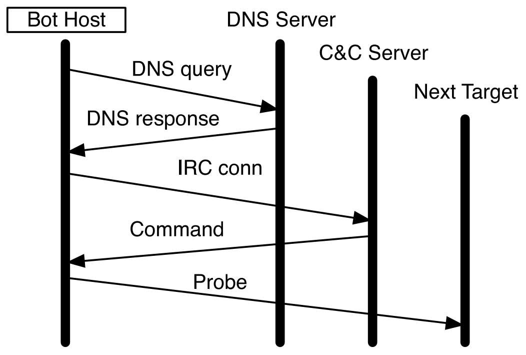

Figure 1. Detectable Event1: A Bot Activity Example

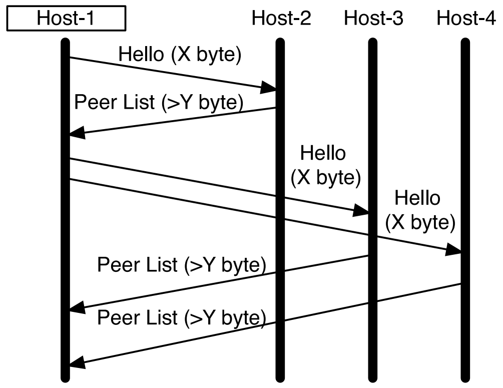

Figure 2. Detectable Event2: P2P File Sharing Program

2.3 Effective Cases

Figure 1 shows communications example of bot activities. In the figure, an infected host (Bot Host) learns Command and Control (C&C) server's IP address via DNS, connects IRC session to the C&C server, receives a command to probe an other hosts and starts to probe. It is difficult to determine bot activities when inspecting each individual-session, because there sessions are often seen in typical net-

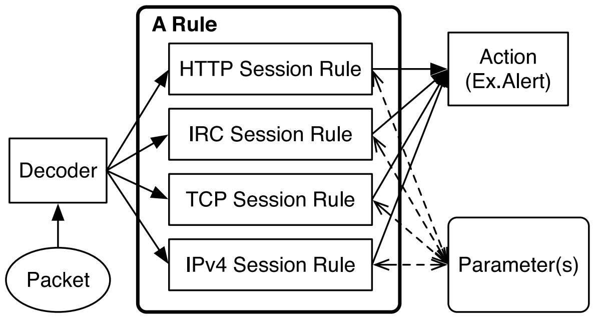

Figure 3. ROOK Design Overview

work traffic. However the order of sessions appearances and protocols (DNS → IRC → Probing) shows a behavior of bots. In addition, there are correlations in the contents of each session. The bot gets IP address of C&C IRC server from DNS response and a trigger of probing by IRC message. The features can be used for detection of bots.

Figure 2 shows an example of P2P file sharing program's communications. Most P2P file sharing program quickly establish many sessions. In figure 2, A P2P file sharing program on Host 1 connects to Host 2,3,4 in quick succession. This software sends *Hello* and receives *Peer List* in each session. It is often the case that protocol of session initiation (e.g. [12]). Therefore P2P file sharing program activities are detectable by counting sessions of the same protocol (e.g. order of packet size and packet direction) even if the traffic is encrypted.

3 Design

ROOK monitors network traffic on the fly and detects security events using its rule base. Figure 3 shows ROOK design overview. First, ROOK decodes packets from live network traffic or a file to determine whether the packet is TCP or UDP, reassembles fragments, sorts by hosts, and maps to sessions. reassemble fragments and segments. Decoding results are compared with predefined rules. Figure 3 shows one example rule at the center of the figure. Rules describe traffic features of each sessions, multi-session correlations and actions when detected security events.

Rules of ROOK are divided by each session. Multiple sessions that are used for security events detection are not synchronized with each other. Therefore, rules of ROOK consists of some session rules that are independent of other session rules for keeping asynchronous. In the figure, session rules for HTTP, IRC, TCP and IPv4 are defined. When a packet that consists of IPv6, TCP and IRC is incoming, ROOK test the packet with TCP and IRC rules.

Synchronizing session and cross-reference of other sessions' contents are achieved by *Parameters*. Multiple parameters can be used in each rule. Figure 3 shows parameters that are used by multiple session rules. Parameters are very flexible and can be used for varied purposes. For example, they can be used as a flag for recognizing order of sessions, a counter for the appearance of sessions, memory for saving strings, IP address and so on. Session rules and parameters achieve expression of session appearance order and information exchange between sessions.

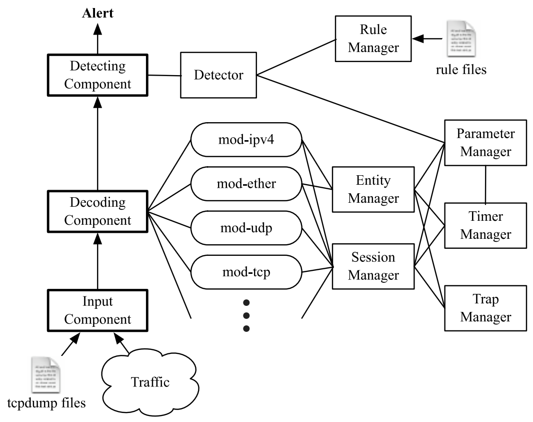

Figure 4. ROOK Implementation

4 Implementation

ROOK is implemented in C with libpcap[13] (ver. 0.9.5, rev. 1), libxml[14] (ver. 2.6.30) and libpcre[15] (ver. 7.4) on Linux 2.6.18 and MacOSX 10.4. The ROOK implementation philosophy is high expand-ability and flexibility. Therefore, components and modules of ROOK have high independence. Figure 4 shows overview of ROOK implementation. ROOK consists of three components:

- **Input Component**: ROOK can capture live network traffic and read tcpdump[16] files by libpcap. This component supports packet queuing for sudden high processing load of Decoding/Detection Component. It can read multiple tcpdump files and capture multiple network interfaces.
- **Decoding Component**: This component includes several protocol modules. Protocols that the current ROOK implementation supports are Ethernet, 802.1Q, ARP, IPv4, IPv6, TCP, UDP, ICMP, HTTP, DNS, TFTP and IRC. These modules manage sessions and entities. In order to determine the presentation layer protocol without TCP/UDP port numbers, this component is implemented using payload-based classification[6]. Since most bot's communications to C&C servers are IRC with a non-standard port[17], a function to determine the protocol without relying on the TCP/UDP ports is necessary.
- **Detection Component**: This component reads rules that are written in XML and detect security events. The

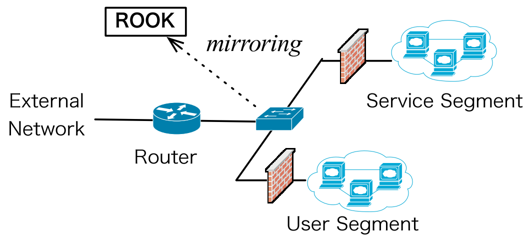

Figure 5. ROOK Running Environment Example

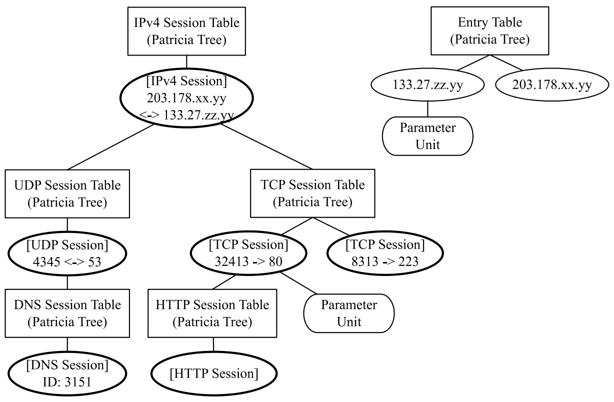
IPv4 Session Table (Patricia Tree)
[IPv4 Session] 203.178.xx.yy <-> 133.27.zz.yy
Entry Table (Patricia Tree)
133.27.zz.yy
203.178.xx.yy
Parameter Unit
UDP Session Table (Patricia Tree)
TCP Session Table (Patricia Tree)
[UDP Session] 4345 <-> 53
[TCP Session] 32413 -> 80
[TCP Session] 8313 -> 223
DNS Session Table (Patricia Tree)
HTTP Session Table (Patricia Tree)
Parameter Unit
[DNS Session] ID: 3151
[HTTP Session]

Figure 6. Session and Entity Management

rule format is described in Section 5. This component compares decoding results of traffic and conditions of triggers. Conditions of triggers are included predefined values and parameter values. Management of parameters is described in Section 4.2.

ROOK monitors network traffic similar to the way major SED implementations. Figure 5 shows an example of a network environment that consists of an external network, an user segment and a service segment. The user segment includes end-users' hosts and the service segments includes servers of Web, Mail, DNS and so on. In the figure, ROOK monitors mirroring traffic among the external network and the user segment and the service segment. In the case of divided internal segments, ROOK can detect more security events by monitoring traffic between internal segments.

4.1 Session and Entity Management

ROOK manages information and status of sessions and hosts for storing parameters and keeping each protocol information. In the implementation, *Session* means unit of communication between two hosts by each protocol: a pair of source and destination IP address on IPv4/IPv6, a pair of source and destination port numbers on UDP/TCP, a pair of request and response on ICMP, one transaction ID on DNS and so on. *Entity* means one identity of hosts on each protocol: IP address on IPv4/IPv6 and MAC Address on ethernet. Each protocol information is used by decoding traffic. Sequence numbers, request statuses, transaction IDs, buffering data and so on are stored like existing IDSes. Parameters can be stored to arbitrary sessions and entities for multi-session correlations. Targets of storing parameters are described in rules. Section 5 describes its rules formats and details.

ROOK is implemented multiple session tables and entity tables as patricia tree by each protocol for isolation of modules. Session tables are implemented as an optional function for modules because that some modules don't have sessions. For example, it is difficulty at ARP module to handle a pair of identities because that ARP request is broadcast and reply is unicast on ethernet. In addition, it is not always true that one IPv4 session is over one Ethernet session (a pair of src/dst MAC address) because of asymmetric IP routing. Therefore, session tables are divided by each protocol for that modules use sessions. Entity tables are implemented on only Ethernet, IPv4 and IPv6 modules and they are devided by each protocol as session tables. Because, there is a possibility of key collision by adding new modules at future.

Figure 6 shows a structure of sessions and entities. The figure shows session tables of IPv4, UDP, TCP, DNS and HTTP and an entity table of IPv4. A patricia tree key of session table is different by each protocol. Foe example, the session table of IPv4 use a byte sequence that is combined source and destination IPv4 address. The key length is 64 bit (32bit + 32bit). In the figure, there are one IPv4 sessions (203.178.xx.yy $\leftrightarrow$ 133.27.zz.yy), two TCP sessions (port numbers are 32413 and 80; 8313 and 223), one UDP session (port numbers are 4345 and 53), one HTTP session and one DNS session. ROOK figure out 2 hosts 203.178.xx.yy and 133.27.zz.yy and there are two hosts entry in the entity table of IPv4. Patricia tree keys of entity tables are IPv4 address (32bit) on IPv4 module, IPv6 address (128bit) on IPv6 module and MAC addresses (48bit) on Ethernet module.

Sessions and Entities create *Parameter Unit* for storing parameters on demand. In the figure, a TCP session (Key: 32413 $\leftrightarrow$ 80) and an entity (Key: 133.27.zz.yy) have parameter units. Functions of parameter units are descirbed in Section 4.2.

4.2 Parameter Management

Parameter management is a very important function for multi-session based detection. Parameters work as counters, flags and so on. The number of parameters and the situation are defined by rules that are written by network security operators. Therefore, parameter implementation must be scalable and high-speed in which they are used. A parameter can be accessed to read or write at high-speed even if huge

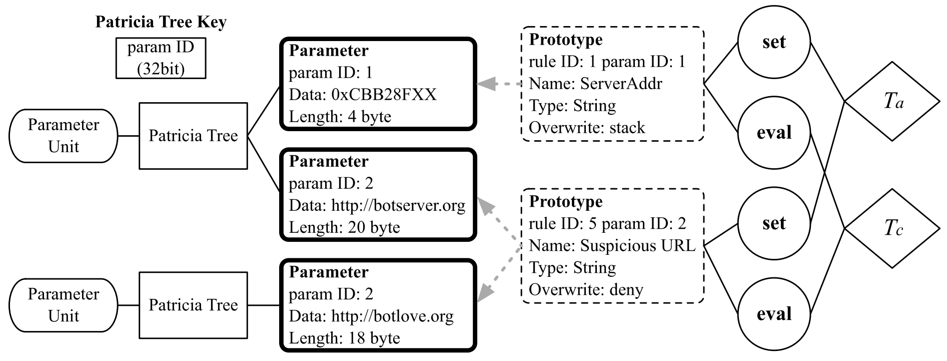

Figure 7. Parameter Management

parameters are saved.

In ROOK, parameter table is implemented as a patricia tree for memory usage and high-speed access. Parameter table is a part of parameter unit. Figure 7 shows parameter management structure. All parameters have parameter ID (param ID). Param ID are sequentially assigned unique numbers at start-up ROOK. A key of patricia tree is a param ID (32bit). Size of a patirica tree node is 24 byte and size of a parameter structure is 28 byte. Therefore, memory usage is increased about $48 + 28 + L$ byte by adding one parameter. (L is saved data length) For example, a parameter of IPv4 address wastes about 84 byte.

All parameters are based a *Prototype* that are written in rules and bound *Operations*. Right side of Figure 7 shows relations of parameter, prototypes, operations, triggers. There are three operations: *Set*, *Eval*, *Unset* that are written with triggers. When conditions of a trigger are matched, operations in the trigger is executed. Parameters are generated from a prototype when the *Set* operation is executed. The parameter type (Int, Bool, Addr, String, and Raw) is defined in its prototype. If the same parameter (same rule ID and param ID) is existing in a parameter unit, ROOK refers to *Mode* that means overwriting modes: *allow*, *deny* and *stack*. Parameter can be overwritten in *allow* mode. In *deny* mode. parameter can't be set other value until it is unset or timeout. In *stack* mode, all values that are set are saved.

5 Experiment

We experimented monitoring by ROOK on two data set with one rule (Figure 8) for showing effectiveness of our method. The first data set (N_1) is traffic from our research network. This network consists of some user segments and a server segment. Both segments are filtered most incoming connection by firewall. The second data set (N_2) is network traffic for a conference. This network is a temporary during four days. User's PCs and servers are in same segment that is not filtered. Table 1 shows the dimensions of our data set. The experimental results are described in following sections.

```
<rule name="Bot IRC Activities ">
 <param name="irc_server" type="addr" mode="stack"
       label="P1"/>

 <session proto="dns" label="G1">
  <trigger label="T1">
   <sig dns.answer_query="botcc1.example.com:A" />
   <sig dns.answer_query="botcc2.example.com:A" />
   <!-- ...snip... -->
   <sig dns.answer_query="botcc.example.ua:A" />
   <set scope="dst" name="irc_server"
        filter="dns.answer_res_data"
         key=":A" timeout="120" />
  </trigger>
 </session>

 <session proto="irc" label="G2">
  <trigger label="T2">
   <sig tcp.dir="to_server">
    <eval scope="src" filter="ipv4.dst_addr"
          name="irc_server" stat="eq" />
   </sig>
   <unset scope="src" name="irc_server" />
   <act type="msg" arg="Detect Bot Node Activity"/>
  </trigger>
 </session>
</rule>
```

Figure 8. Bot Activities Rule

Figure 8 shows a rule that is applied in this experiment. This rule describes bot activities as a multi-session rule.

- P_1: The parameter that is named "irc_server" is saved some value as IP address and a stackable parameter. It is defined in `<param>` tag. This parameter means an IP address of C&C IRC server.

- G_1: The upper session rule (G_1) is described a trigger (T_1) that means DNS response of suspicious domain names based on [18] and traffic data from our honeypot. A `<trigger>` consists of four type tags: `<sig>`, `<set>`, `<unset>` and `<act>`. T_1 has multiple `<sig>` tags and one `<set>` tag. Multiple `<sig>` tags in a trigger means concurrent conditions. `dns.answer_query="botcc-ex1.com:A"` of `<sig>`'s attribute means that answer records of a DNS response packet included *botcc-ex1.com* and the an-

Table 1. Dimensions of our Data Set

Set	Start	Duration	Packets	Bytes	IP Sessions	Ave. bps	Ave. pps
N_1	2007-11-13 21:00	48 hour	76.77M	38.06G	811.37K	1.76M	444.32
N_2	2007-09-12 10:00	12 hour	163.86M	77.15G	692.94K	14.28M	3.79K

Table 2. Results of the Experiment

Trigger	Condition	N_1	N_2
T_1	$S_1 \cup S_2 \cup ... \cup S_{21}$	756	419
T_2	$S_{22} \cap E_1$	0	0

Table 3. Details of the Experiment

Trig.		Sig./Eval.	N_1	N_2
T_1	S_1	dns.answer_query=botcc1.example.com	1	17
	S_2	dns.answer_query=botcc2.example.com	6	3
	S_3	dns.answer_query=botcc3.example.com	2	20
	S_4	dns.answer_query=botcc4.example.com	0	6
	S_5	dns.answer_query=botcc5.example.com	0	5
	S_6	dns.answer_query=botcc6.example.com	0	2
	S_7	dns.answer_query=botcc7.example.com	0	3
	S_8	dns.answer_query=botcc8.example.com	6	6
	S_9	dns.answer_query=botcc9.example.com	2	3
	S_{10}	dns.answer_query=botcc10.example.com	69	173
	S_{11}	dns.answer_query=botcc11.example.com	0	3
	S_{12}	dns.answer_query=botcc12.example.com	141	67
	S_{13}	dns.answer_query=botcc13.example.com	0	6
	S_{14}	dns.answer_query=botcc14.example.com	0	7
	S_{15}	dns.answer_query=botcc1.example.net	54	10
	S_{16}	dns.answer_query=botcc2.example.net	0	19
	S_{17}	dns.answer_query=botcc3.example.net	2	9
	S_{18}	dns.answer_query=botcc1.example.org	5	8
	S_{19}	dns.answer_query=botcc1.example.ru	302	35
	S_{20}	dns.answer_query=botcc2.example.ru	81	13
	S_{21}	dns.answer_query=botcc.example.ua	85	4
T_2	S_{22}	tcp.dir=to_server & tcp.seg_id=1	5	23
	E_1	*irc_server*=ipv4.dst_addr	0	0

swer record is an *A Record*. When monitor traffic matches the conditions of any one `<sig>`, `<set>`, `<unset>` and `<act>` in same `<trigger>` tag are executed. In T_1, a `<set>` tag is executed when matching conditions. This `<set>` tag store A record of the DNS packet in a parameter unit of destination host as a parameter that is named *irc_server*. Timeout of this parameter is after 120 seconds from storing parameter.

- G_2: The lower session rule means an IRC connection. There are `<sig>` and `<eval>` tags in T_2. `tcp.dir=to_server` shows a condition of a packet direction: to server and `tcp.seg_id=1` shows a condition of TCP segment data packet ID. They mean first packet with data segment after established TCP connection. `<eval>` tag is for comparing destination IP address (`filter="ipv4.dst_addr"`) and parameter "irc_server" instead of static conditions. If destination IP address is matched with "irc_server", parameter is unset and detection message is output as alert by `<act>` tag.

Table 2 shows the result of this experiment and Table 3 shows detection count of each condition. In Table 3, the first column of the table shows labels of triggers ($T_{1,2}$). The second column shows name of each condition and the third column shows conditions of `<sig>` and `<eval>`. S_n means a `<sig>` tag and E_n means a `<eval>` tag. The fourth and fifth column show detection count in data sets: $N_{1,2}$. in Table 2, the first column shows triggers and the second column shows conditions of triggers. The third and fourth column show detection count of each trigger in data sets.

In this experiment, no bot activities are detected by our method. In results, there are many DNS queries of suspicious domain name for purposes other than IRC connection. Reason that $T_{1,2}$ count of N_1 is grater than N_2 is thought of as more internal hosts on N_1 than N_2. Our method monitors not only DNS response packets but also IRC communication. T_2 compares both of signatures: `tcp.dir=to_server & tcp.seg_id = 1` and a parameter: `irc_server=ipv4.dst_addr`. When destination IP address of IRC communication to server match with the *irc_server* parameter, ROOK determines that monitoring communication is bot's IRC connection. Some IRC connections are observed in data set. However they are not bot's activities.

6 Discussion and Future Work

This experiment shows an accuracy of our method is higher than existing methods. Existing methods get false positives trying to detect suspicious domain names of DNS query and response from these data sets. However our method gets no false positives in the experiment. On the other hands, we confirmed that ROOK with Figure 8 is able to detect 27 different kinds of bot:

- 9 "Trojan horse IRC/BackDoor.SdBot" variant
- 7 "Trojan horse SHeur" variant

- 5 “Virus identified Wind32/Virut” variant
- 3 “Trojan horse BackDoor.Generic” variant
- 3 others

Their names are checking results by AVG Free Edtion[19]. The 27 samples were selected at random from 1,550 bots that was captured by our honeypot between Apr. 2007 and Nov. 2007. The confirmation was using network traffic that is generated by running bot samples. Bots were running on Windows XP SP2 over Parallels Desktop 3.0 for Mac[20] and the bots communications traffic with DNS server, C&C server and so on were captured by tcpdump on gateway host. ROOK determined that 27 sample traffic are communications of bot activity. Therefore our method is effective to detect bot activities.

Experiments by rules to detect other objects (e.g. P2P file sharing program) and study other effective cases are needed. We did experiments only using one rule for bot IRC activities on the two data sets record in this paper. In order to detect most bots, not only IRC communications but also other activities of bots should be supported by our method. On the other hand, our method has possibilities of accuracy increase for existing IDSes' signatures. Therefore we need try to detect various security events by ROOK.

We have to experiment in various environments too. false positives/false negatives rate of SED's methods depends strongly on network traffic trends and environment. In order to demonstrate an effectively of our method, results of many experiment by various network traffic are needed.

We have to evaluate performance of this implementation. ROOK may be heavy more than existing IDS because ROOK has new functions: *parameter* in addition to existing IDS functions: packet decoding, session management, pattern matching and so on. Memory usage is increased by storing parameter and number of steps is increased by comparing parameters and network traffic. We should evaluate performances of ROOK to know effect of performance decrement on detection procedure.

We designed and implemented multi-session based SED: ROOK to detect security events by encrypted traffic and faked traffic. In addition, we experiment ROOK with rule of detection bot DNS and IRC activities by using two data sets from live network traffic. The results of experiments shows effectively of multi-session based detection method for a part of bot activities. In the future, we have to work more experiments and evaluations about ROOK.

7 Acknowledgments

We thank Rodney Van Meter for writing and technical advice.

References

[1] Paul Bacher,Thorsten Holz,Markus Kotter,Georg Wicherski. Know your enemy: Tracking botnets, May 2005. `http://www.honeynet.org/papers/bots/`.

[2] James R. Binkley. An algorithm for anomaly-based botnet detection. *SRUTI '06*, pages 43–48.

[3] Musashi Y.,Ludena R.,Dennis A.,Nagatomi H.,Matsuba R.,Sugitani K. A DNS-based Countermeasure Technology for Bot Worm-infected PC terminals in the Campus Network. *Journal for Academic Computing and Networking*, 10(1):39–46, 2006.

[4] Andrew W. Moore, Denis Zuevy. Internet Traffic Classification Using Bayesian Analysis Techniques. *SIGMETRICS0́5*, 2005.

[5] P-CUBE. Approaches To Controlling Peer-to-Peer Traffic: A Technical Analysis. `http://www.p-cube.com/doc_root/products/Engage/WP_Approaches_Controlling_P2P_Traffic_31403.pdf`.

[6] T. Karagiannis, D. Papagiannaki, M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. Technical report, `http://research.microsoft.com/~thomkar/papers/BLINC_TR.pdf`, 2005.

[7] Risto Vaarandi. SEC - a Lightweight Event Correlation Tool. *Proceedings of the 2002 IEEE Workshop on IP Operations and Management*, pages 111–115, 2002.

[8] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[9] OSSIM Open Source Security Information Management. `http://www.ossim.net/`.

[10] ArcSight. `http://www.arcsight.com/`.

[11] Masayoshi Mizutani, Shin Shirahata, Masaki Minami, Jun Murai. A proposal of a method to detect security events by using correlation of multi-session. *Computer Security Symposium 2007, IPSJ Symposium Series*, Nov 2007.

[12] Salman A. Baset and Henning G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *IEEE Infocom*, 2006.

[13] LIBPCAP. *`http://www.tcpdump.org/`*.

[14] Daniel Veillard. libxml. Sep 1999. `http://xmlsoft.org/`.

[15] Philip Hazel. PCRE - Perl Compatible Regular Expressions. sep 1997. `http://www.pcre.org/`.

[16] LBNL's Network Research Group. TCPDUMP. `http://www.tcpdump.org/`.

[17] Cyber-TA Research and Development Project. SRI Honeynet and BotHunter Malware Analysis Automatic Summary Analysis Table, 2005.

[18] Bleeding edge threats: blackhole.conf, Nov 2007. `http://doc.bleedingthreats.net/pub/Main/SnortConfSamples/blackhole.conf`.

[19] Grisoft. AVG Free Edition. `http://free.grisoft.com/`.

[20] Parallels. Parallels desktop for mac. `http://www.parallels.com/`.