Global Information Assurance Certification Paper

Michael Ryan
GSEC Option 1 version 1.4
25 July 2002

OpenBSDs PF: An Alternative to Commercial Firewalls

Introduction

With so many people and companies having access to the Internet, many do not have the money to spend for a commercial firewall. Fortunately, an alternative way to protect internal networks effectively is with a Stateful Inspection firewall running on an inexpensive and functional operating system: OpenBSD.

The Stateful firewall is PF (Packet Filter), which runs on OpenBSD. This paper goes over the basic operation of PF, examples of some attacks that PF can help prevent, and how to get a basic system up and running. The OpenBSD project is known for its extensive security audits performed on the code that makes up the OpenBSD system. The members that make up the OpenBSD team are constantly reviewing the code looking for already known security vulnerabilities and for potential ones that may not have been discovered. Due to the attention to details when it comes to security, OpenBSD is an ideal platform for a firewall. Some background knowledge is assumed in OpenBSD, basic networking concepts, and TCP/IP. The following information will help to better understand Stateful Inspection and OpenBSDs implementation of it.

Background

A basic understanding of the Open Systems Interconnect (OSI) protocol stack and the way data is encapsulated especially the structure of an IP packet, is important to understanding any implementation of Stateful Inspection. The OSI stack is comprised of seven layers. The Application layer (layer 7) is where the user programs operate. This layer provides FTP, SMTP, Telnet, and more. The Presentation layer (layer 6) takes care of formatting data so that the application layer can accept it and formats and encrypts data to be sent across a network. The Session layer (layer 5) establishes, manages and terminates connections between applications. Reliable communication from end-to-end is provided in the Transport layer (layer 4). The Network layer (layer 3) handles the movement of packets around the network, i.e. routing, error handling and packet sequencing. In the Data Link layer (layer 2), packets are encoded and decoded into bits. The Data Link layer also handles errors in the physical layer. This layer further divides into the Media Access Control layer (MAC) and Logic Link Control layer (LLC). The Physical layer (layer 1) actually puts the data on the physical medium. In addition, Layer 1 controls voltage, electrical pulses, and light or radio waves.

Application (layer 7)
Presentation (layer 6)
Session (layer 5)
Transport (layer 4)
Network (layer 3)
Data Link (layer 2)
Physical (layer 1)

Figure 1 OSI Protocol Stack *(Stevens)*

Another protocol stack one should be familiar with is the TCP/IP stack. In comparison to the OSI protocol stack, the TCP/IP stack is much simpler. The OSI stack was also designed with more granularity and other protocols in mind other than TCP/IP and the TCP/IP stack. The TCP/IP stack is made up of only four layers: the Application, Transport, Network or Internet, and the Link layers. The Application layer (layer 4) handles the details of an application. The Transport layer (layer 3) handles the flow of data between to hosts from the application layer above. The TCP/IP suite has two different protocols at the Transport layer, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides a reliable flow of data, doing such tasks as error checking and breaking the data into appropriately sized chunks for the network layer. On the other hand, UDP only sends the data out, which is commonly referred to as datagrams. There is no guarantee that the packets reach the other host, this reliability must be handled by the application. Routing and movement of packets around the network are handled in the Network layer (layer 2). The Link layer (layer 1) puts the data on the wire. It also includes the device driver in the operating system and the Network Interface Card (NIC).

Application (layer 4)
Transport (layer 3)
Network (layer 2)
Link (layer 1)

Figure 2 TCP/IP Protocol Stack *(Stevens)*

Encapsulation

As the data goes down through the layers, each layer adds a header to the previous layers header and data. This process is widely known as encapsulation. When the receiving system gets these packets, the opposite takes place. Each layer strips off its header and sends the rest up the stack to the next layer. This process is called demultiplexing.

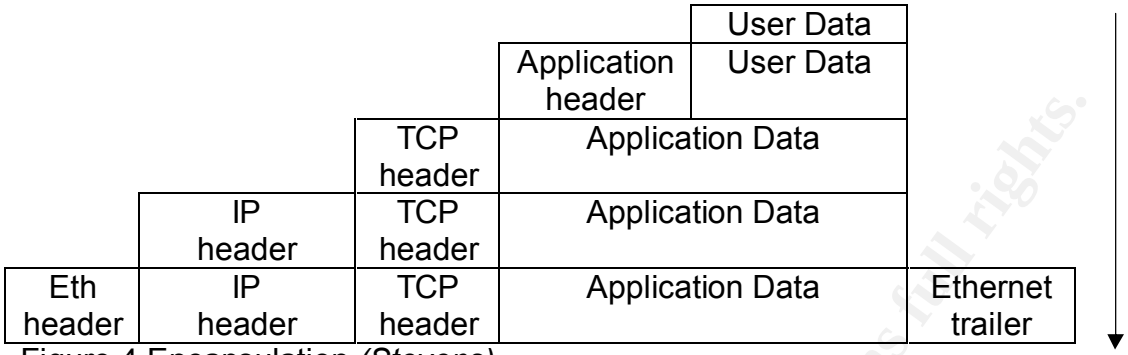

Figure 4 Encapsulation *(Stevens)*

A header that will be discussed is the IP version 4. The structure of the IP header is shown in Figure 5.

Version	**Header Len**	**TOS**	**Total Length**
IP Identifier		**Flags**	**Fragment Offset**
TTL	**Protocol**	**Header Checksum**	
Source Address			
Destination Address			
Options (Padding if needed)			
Data			

Figure 5 The structure of the IP header. *(Stevens)*

The Version field indicates the format of the IP header, in this case Ipv4. The Header Length specifies the length of the IP header in 32 bit words. Type of Service (TOS) specifies what type of service is requested by the application sending the data, e.g. minimize delay, maximize throughput, maximize reliability, and minimize monetary cost. The Total Length field is the length of the IP packet including header and data. The maximum packet size is 65535 bytes. IP Identifier contains a unique identification number that is given when the packet is created. When a packet is fragmented all of the fragments of the original packet contain the same IP Identifier to aid in reassembly. The Flags field contains three flags: Reserved, Don't Fragment (DF), and More Fragments (MF). The Reserved flag must be set to 0. In the second flag DF, if the flag is 0 the IP packet can be fragmented. If the flag is set to 1 the IP packet cannot be fragmented. In MF, if the flag is 0 there are no more fragments to follow. If the flag is 1, there are more fragments to follow. The Fragment Offset field contains a value to point to where the fragments data fit into the original packet; this is used when the receiving host reassembles the fragmented packets. Time to Live (TTL) contains a value, no greater than 255. When a packet passes through a router, the router decreases the TTL by one. If the TTL reaches 0 it is discarded. The reason for this functionality is to keep packets from wandering endlessly around a network. The Protocol field contains what protocol gave IP the data to

send. Header Checksum has a value referred to as a checksum. When an IP packet is created a Cyclic Redundancy Check (CRC) is performed on the IP header, it does not cover the data portion. When the receiving host gets the packet it also performs a CRC on the IP header. If the two Header Checksums do not match the packet, it is considered damaged and is discarded. The Source Address has the 32-bit IP address from the sending host. The Destination Address contains the address of the receiving host. The Options and Padding field is optional; it contains special handling instructions not covered in any other field in the IP header. Finally, the last field is the Data field; it contains the actual data and the previous layers header that is being sent from one host to another. *(Stevens, 34-37)*

Stateful Inspection

When a firewall is stateful, it keeps track of connections and the information pertaining to these connections in memory. An example of information stored in the state table is the destination address, port number, TCP sequencing numbers and flags. When a packet hits the firewall it is compared to entries in the state table. If an entry exists, it is passed; if not the firewall parses a user defined rule file to determine if the packet is allowed.

The advantage to Stateful Inspection is the information that the firewall has on each connection. The firewall can then use this information to detect and eliminate various types of spoofing attacks and other attacks that will be discussed later.

OpenBSDs PF

OpenBSDs PF was originally written by Daniel Hartmeier. It's now being improved and expanded by numerous other contributors. *(Hartmeier, OpenBSD Packet Filter)* PF resides in the kernel where it inspects every packet that enters or leaves the stack. PF is made of two components: the filter rules and the state table. All packets are compared against the rule set, which describe actions to be taken; for example, a pass or block. An example rule may look like:

```
block in all
pass in all
```

When the packet is being compared to the rule set, it is compared against all of the rules within the set. A packet may match more than one rule. If this happens, the last rule, which matches is applied to the packet. PF is also able to keep state. Any rule that passes a packet may also create an entry in the state table. A rule that passes a packet and creates and entry may look like this:

```
Pass in proto tcp from any to any port = 80 keep state
```

If PF receives a packet from a connection and it has an entry in the state table, it recognizes this connection as a pre-established valid connection and passes the received packet without further checking it against the rule set. PF handles TCP somewhat differently from UDP connections since UDP is stateless and TCP is stateful. When keeping state on TCP connections PF checks the sequence numbers of each packet against the state table. If they do not match they are dropped. Since UDP is stateless, PF keeps track of connections by host address and port number. PF considers additional UDP packets part of the same connection if the host address and port numbers match an entry in the state table. It may seem that keeping state on connections would be a performance burden on the firewall, by consuming system resources and therefore causing a bottleneck and slowing the connection down. However, it is much faster to perform state table lookups than to parse the rule set. "A typical rule set of 50 rules takes about 50 rule comparisons, whereas a state table of 50,000 entries, due to its binary tree structure, takes only about 16 comparisons." *(Coene,7)*

Another security advantage to PF is the capability to generate Initial Sequence Numbers (ISN). ISNs are randomly chosen at the beginning of a connection between two hosts. Then they are incremented by one for each byte sent. Some TCP stacks provide easy to guess ISNs, which makes them susceptible to ISN exploits, such as session hijacking. The ability for PF to provide random Initial Sequence Numbers (ISN) is quite beneficial for systems behind the firewall that do not have the ability to generate random ISNs, thus leaving them open to attacks such as IP spoofing and session hijacking. The generating of ISN numbers through PF is referred to as State Modulation and can be added to a rule in the rule set, with the directive "modulate state".

PF also provides IP normalization (defragmentation). To better understand IP normalization a brief explanation of IP fragmentation is needed. When packets traverse many networks, such as the Internet, certain network segments can only handle certain sized packets. The media that the network is made up of, e.g. FDDI, Ethernet and Token Ring causes this restriction. As the data reaches a new network segment, the IP layer of the protocol stack determines the Maximum Transmission Unit (MTU) of that particular segment. If the packet is larger than the MTU, then the IP layer performs fragmentation. When a packet is fragmented, each fragment becomes it's own packet, with its own IP header. *(Stevens, 149)* If the fragmented packet reaches a network that has a larger MTU compared to the network from which they just came from; then, the packets normally remain fragmented. As fragmented packets are received by PF, they are cached and reassembled. Reassembly of the packets takes place so if the vulnerability to various fragment attacks exists on any machines in the internal network; the threat to them is minimized. Here are a few different fragmentation attacks that PF can help eliminate through normalization.

Ping O' Death

In essence, the Ping O' Death is an illegal, oversized packet that may create havoc on an operating system. Many operating systems don't like to be pinged with a packet size greater than 65535 bytes (the default being 64 bytes). So, if 65536 byte packets are illegal, how are they made? Some versions of Ping will allow the user to send a Packet with a payload of 65536 or more. These oversized packets are then fragmented. When some receiving hosts reassemble them it causes a buffer overflow and corrupts the stack, resulting in a reboot, crash, panic, or hang.

Tiny Fragment Attack

The Tiny Fragment Attack uses small fragments. This attack is used to get through a firewall that checks the first packet. If the packet does not match any rules it's passed, as are subsequent packets. Ziemba, Reed and Traina clearly describe the Tiny Fragment attacks passage, "If the Fragment size is made small enough to force some of a TCP packets TCP header fields into the second fragment, filter rules that specify patterns for those fields will not match" *(RFC 1858).* Often this technique is employed to hide other malicious activities from a company's Intrusion Detection System (IDS), which use signatures to recognize various attacks.

Teardrop Attack

The Teardrop attack uses overlapping offset fields to cause problems on the receiving host, the Teardrop program crafts these custom packets. When the IP layer is reassembling a fragmented packet, it expects the incoming fragments to align so that the start of the data in the packet immediately follows the end of the data in the preceding packet.

Preventive Measures from Attacks

The previous attacks were added to give an idea of different types of fragmentation attacks. There are also many variations on these attacks. While a lot of these are old attacks and most vendors have released patches for their systems, a system behind the firewall may be unpatched, or a new type of fragmentation vulnerability may surface. OpenBSD can handle these attacks by caching the packets it receives and defragmenting them, then sending them on to the receiving host. This way if PF does encounter any anomalies in the packets either they are discarded or corrected and passed on.

It's helpful to have a firewall like OpenBSDs PF that can perform normalization on the packets that pass through it. While the firewall may prevent some attacks on unpatched systems, some other attacks may get through. The security should be layered to further ensure protection. Patching is another form, or layer, of prevention to add to the networks security. One should have a procedure in place for patching the systems on a regular basis and for monitoring

various sources, such as Bugtraq for new vulnerabilities. *(http://www.securityfocus.com/cgi-bin/vulns.pl)* In order for PF to normalize traffic, the scrub directive needs to be used. The following rule can be used to scrub (normalize) all incoming traffic:

```
scrub in all
```

Wouter Coene forewarns "Using the scrub directive uses quite an amount of server resources, so its use should be limited to protecting only the weak TCP/IP stack implementations" *(Coene, 16).*

Applying Concepts to a Working System

I will not discuss installation of OpenBSD. OpenBSDs site provides excellent documentation on installing a base system. However, I will go over some tips and advice for running PF.

We only want what is necessary for running our firewall, the more things you have running the more problems you can have, e.g. services to exploit. Fortunately, the OpenBSDs teams design goals where primarily that of security. Under /etc/rc.conf set the following:

```
portmap=NO
check_quotas=NO
ntpd=NO
sendmail_flags=NO
```

These are disabled due to the fact that they are not necessary on a firewall and are open to numerous vulnerabilities.

Next, make sure your system is up-to-date on all patches. Detailed instructions for doing this are on OpenBSDs site: www.openbsd.org. In /etc/sysctl.conf uncomment the line:

```
net.inet.ip.forwarding=1
```

This will allow your firewall to route packets between the two network cards.

The base install of OpenBSD provides PF. PF is disabled by default; one has to enable it. To enable it set the PF line to yes in /etc/rc.conf. This will also enable NAT (Network Address Translation).

```
pf=YES
```

The writing of PF rules should not be overlooked. These rules are located in /etc/pf.conf. Examining more closely the example of a rule previously used:

```
block in all
pass in all
```

"Block in all" is short for "block in from any to any". Similarly, "pass in all" is shortened from "pass in from any to any". As PF parses these rules, the first one tells it to block all incoming packets. When PF finds a match, it doesn't stop going through the rules, PF "remembers" that it found a rule that matched and continues. The second rule also matches. PF remembers this as well. At this point, it's reached the end of the rule file. PF goes back and sees that "pass in all" was the last rule that matched, and applies it.

The example above doesn't help too much, as it lets everything in to the network. All of our PF rules will start with "block in all" and a "block out all" (to cover both directions). The reason for starting with "block in all" and "block out all" is to explicitly tell PF if this packet is not allowed later on in the rule set, to block it. By default, PF will pass packets if they do not match a rule. Here are some useful rules that can be incorporated into a pf.conf. First, the objective is to block anyone trying to spoof non-routable addresses that would be coming from the Internet. In the rules below, ep0 is assumed to be the external interface on the firewall. The *quick* option tells PF to perform the rule without going through the rest of the ruleset. This can speed PF up if the rule explicitly directs the appropriate action to be taken next.

```
block in quick on ep0 inet from 127.0.0.1/8 to any
block in quick on ep0 inet from 192.168.0.0/16 to any
block in quick on ep0 inet from 172.16.0.0/12 to any
block in quick on ep0 inet from  10.0.0.0/8 to any
```

Next, add two rules to block everything going and leaving the network, so we can open explicitly what we want later.

```
block in all
block out all
```

To allow users on the internal network, access to the Internet. We can add a rule that passes any tcp traffic out of the internal network. This rule passes the initial packet (SYN packet) of each connection and keeps state on it. Keeping state on the SYN of incoming connections is another way to prevent people trying to scan a network using invalid combination of flags.

```
Pass out on ep0 proto tcp from any to any flags S/SA keep state
```

In a network that has a web server, which is available to the public, it is prone to exploits. Perhaps a preferable option for this web server is to keep state on the

initial SYN of the connections. This can be done with the following, where "xxx.xxx.xxx.xxx" is the web server address:

```
pass in on ep0 proto tcp from any to xxx.xxx.xxx.xxx/32 port = 80 \
                                                    flags S/SA keep state
```

The potential is present in the above example for someone to take advantage of the firewall. The state table entries are being created with the initial SYN of each connection. Someone could send a lot of packets with the SYN flag set to fill the connection tables and cause a denial of service. In this case, legitimate traffic could not reach the web server. To get around this possible denial of service (DoS) PF will clear connections from the state table after a preset period of time.

The utility to interact with PF is pfctl. Pfctl is run when OpenBSD is started up, but it can also run when the system is up. This is helpful for making changes to the rule set and not having to reboot to have those changes take affect. PF will be able to keep current connections open if a rule set is reloaded. The only time it wouldn't be able to do this is if a connection has a state table entry based on the initial SYN flag. To load the rules without rebooting the firewall:

```
pfctl –R /etc/pf.conf
```

Conclusion

OpenBSD allows the use of Stateful Inspection through PF to make an effective and inexpensive addition to an organization's security plan. In order to comprehend the implementation of OpenBSD and PF as a measure of security for a network, the given background knowledge is reviewed. This knowledge is vital in any area of an environment's security. Understanding how PF parses its rule sets and makes decisions are critical to writing a rule set. If incorrect rules are put in, packets that were blocked at one point may be allowed at another rule later on. Preventing attacks from the outside environment using a firewall is the first step to securing a network. Preventative measures, such as adding patches for systems, aides in the protection of a network, but OpenBSDs PF performs more extensive methods like normalization and the checking of sequence numbers.

To remodel and refine the security of a network, preliminary actions must be taken before applying OpenBSD with Stateful Inspection and PF. Applying these guidelines includes but is not limited to: shutting off unnecessary services, enabling IP forwarding, writing specific rules and taking precautions to prevent certain attacks such as the installment of updated patches.

For further reference or investigation visit following hyperlinks:

To access OpenBSD material:

http://www.openbsd.org

Home of PFs author and extensive information related to PF:
http://www.benzedrine.cx

The OpenBSD Packet Filter HOWTO:
http://www.inebriated.demon.nl/pf-howto/

References

"FAQ 6.0 Networks." URL: http://www.openbsd.org/faq/faq6.html (10 July 2002).

"OpenBSDs Programmers Manual PF.CONF." 2 July 2002. URL: http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html (15 July 2002).

Coene, Wouter. "The OpenBSD Packet Filter HOWTO." Version 20020405.2. 5 April 2002. URL: http://www.inebriated.demon.nl/pf-howto/ (10 July 2002).

Farrell, James. "IP Fragmentation Attacks on Checkpoint Firewalls." 3 April 2001. URL: http://rr.sans.org/firewall/frag_attacks.php (9 July 2002).

Hartmeier, Daniel. "Design and Performance of the OpenBSD Stateful Packet Filter (pf)." URL: http://www.benzedrine.cx/pf-paper.html (10 July 2002).

Hartmeier, Daniel. "OpenBSD Packet Filter." July 15, 2002. URL: http://www.benzedrine.cx/pf.html (July 21, 2002).

Handley, Mark, Vern Paxson, and Christian Kreibich. "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics." February 19, 1998. URL: http://www.icir.org/vern/papers/norm-usenix-sec-01-html/ (15 July 2002).

Hill, Greg. "Linux Comes of Age with Stateful Firewalling." 20 February 2001. URL: http://rr.sans.org/firewall/stateful.php (8 July 2002).

Senner, Lisa. "Anatomy of a Stateful Firewall." 9 May 2001. URL: http://rr.sans.org/firewall/anatomy.php (7 July 2002).

Stevens, W. Richard. TCP/IP Illustrated Volume 1 The Protocols New York: Addison-Wesley, 1994.

"The Ping o' Death Page." 25 November 1996. URL: http://www.pp.asu.edu/support/ping-o-death.html (11 July 2002).

Tran, Hoang Q. "OpenBSD firewall using pf." 10 July 2002. URL: http://www.muine.org/~hoang/openpf.html (11 July 2002).

Walden, Ralph E. “IP Header Fields.” URL: http://www.ee.siue.edu/~rwalden/networking/iphead.html (15 July 2002).

Ziemba, Paul G., Darren Reed, and Paul Traina. “Security Considerations for IP Filtering.” 5 October 1995. URL: http://www.ietf.org/rfc/rfc1858.txt?number=1858 (19 July 2002).

Last Updated: July 21st, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS