Processing experimental protocols against IDS

Tommy Adams
Processing experimental protocols against IDS

GIAC GCIA Gold Certification

Author: Tommy Adams,  tommy@adamsfam.org
Advisor: Chris Walker
Accepted: 8/7/2018

Abstract

Experimental protocols such as TCP Fastopen, QUIC, and Multipath TCP are not uncommon on Internet-connected networks. If a network has modern operating systems and browsers it is a near certainty that experimental protocols are traversing the network. This paper will examine potential consequences of experimental protocols to current network security monitoring practices and the potential for intrusion detection evasion. This paper will provide a roadmap by which an analyst may process any new, odd, or experimental traffic against their open-source intrusion detection system.
1. Introduction

The purpose of this paper is to help an analyst determine how their open-source intrusion detection system (IDS) will process experimental protocols, and what to do if they find anomalous behavior. Experimental protocols are not so abnormal as to be unworthy of our attention. TCP Fastopen is an experimental protocol, and yet Citrix’s Netscaler, an Enterprise-centric platform has a support article on how to enable TCP Fastopen. TCP Fastopen is supported by current version of Linux, macOS, Windows, and BSD; as well as most current browsers. The Quick UDP Internet Connection (QUIC) protocol is another example of an experimental protocol with broad support; Google states that they have used QUIC for years to support such popular platforms as YouTube. Multipath TCP is a final example of an experimental protocol that is used on mobile devices to provide a behind-the-scenes transition from Wi-Fi to cellular networks. iOS version 7+ with Siri began using Multipath TCP, and in iOS 11+ the API for developers using Multipath TCP has been made publicly available. These examples show that experimental protocols are not limited to the fringes of the Internet, they can be mainstream and are therefore worthy of further analysis by those who are involved in network security monitoring and who manage intrusion detection systems.

Before we analyze experimental traffic we’ll review an example of a typical, established, protocol, such as TCP. A Request for Comments document (RFC) is how a protocol is defined. RFCs are typically processed and archived by The Internet Engineering Task Force (IETF). An example of an established RFC is RFC 793, which defines typical behavior for the TCP protocol.

Tommy Adams, tommy@adamsfam.org
RFC 793 includes the typical behavior of a three-way handshake by which two
hosts establish a TCP connection and begin to send and receive data after the completion
of the three-way handshake. Figure 1 represents a visualization of the three-way
handshake (3WHS), as shown in Wireshark:

1. The first packet uses the TCP SYN Flag (pn 51)
2. The second packet the SYN and ACK flags (pn 53)
3. The third packet the ACK flag (pn 54)
   a. In each case of pn 51, 53, and 54, the Len=0 value indicates there is no
data in each of those packets.
4. The fourth packet, pn 77, shows the first data packet of the newly established
session over tcp/80; the data is evidenced by the HTTP GET call.

While RFC 793 does not prohibit data on the initial SYN packet, it is not typical.

If there is data on the SYN, this data is not passed up to the application layer until the
3WHS is complete. TCP Fastopen operates differently in this regard, and there are
security implications for the way that TCP Fastopen works.

Experimental protocols are those that operate outside the confines of existing,
established RFCs. For example, TCP Fastopen begins the transference of data between
two hosts before the 3WHS is complete, rather than the typical approach of transferring
data once the TCP session has completed each step of the 3WHS. TCP Fastopen, and other experimental protocols do not merely ignore the historical and current value of RFCs, in fact, TCP Fastopen has an ‘Experimental’ RFC.

RFC 7413, published in December of 2014 explains the concept of TCP Fastopen (TFO) and it is clear that the primary factor driving this experimental use of TCP is performance. A TFO session breaks the tradition found in RFC 793 and begins transmitting data as soon as possible. The following diagram represents a TCP connection making use of TFO:

![TCP Fast Open Diagram](image)

Figure 2 shows the first packet, originating from the Sender host, contains the TCP SYN flag, which is the first packet in a 3WHS, as well as data. The second packet is also standard 3WHS, the Receiver host responds with the SYN-ACK flags set. After the second packet we see the atypical traffic, as represented by the third and fourth packets; which show the Receiver host sending data to the Sender host before the handshake is complete. Another unique factor of TFO traffic is that data on the SYN is passed up to the application layer right away before the 3WHS is complete. This traffic has no
indication of being malicious, but it is not RFC 793 traffic, it is experimental. When an intrusion detection signatures or rule is written within the context of expecting TCP traffic to adhere to RFC 793, the rules or signatures may fail to report actual malicious activity, or they may give false-positive alerts.

Finally, let us consider network traffic that does not strictly follow RFCs. RFCs describe the standard approach and implementation of a protocol, but an application developer, for example, may ignore convention through casual laziness, ignorance, or willful intent, which may or may not be malicious. An example of intentionality is the S7comm protocol. S7comm is a proprietary protocol used by some Siemens programmable logic controllers. There are some things about this protocol that are well-understood, such as it uses TCP port 102, and consists of several other protocols (e.g., COTP, TPKT, TCP). However, S7comm uses a proprietary data payload that cannot be dissected by publicly available tools such as Wireshark or tcpdump. The S7comm protocol is found in Industrial Control Systems which are used in verticals such as manufacturing, automation, and critical infrastructure. S7comm is one example of network traffic which cannot be wholly defined via an RFC, or even by multiple RFCs. The following diagram, Figure 3, reveals which part of the S7comm protocol makes use of RFCs and which do not.

Tommy Adams, tommy@adamsfam.org
Analyzing protocols for which there is no RFC that is either well-established or experimental is out of the scope of this paper. However, they are a reality and the analytical approach that follows can be applied to learn more about network traffic for which there is no RFC.

RFCs define the standards for network protocols and equipment, and these RFCs are used by operating systems, and applications to interpret network traffic and process data. Intrusion detection systems, such as Snort and Bro, are applications which process network data and these start by using a Packet Decoder. Snort documentation states that decoding is the first process that takes place. The decoder determines the underlying protocols and then inspects for anomalies. For example, when the decoder identifies the packet as IPv4, but the packet is less than 20 bytes (minimum requirement for an IPv4 packet), the decoder will alert and stop decoding that specific packet – there is no reason to continue to process the packet against a preprocessor(s) or rule(s) because it has already shown itself to be anomalous. Bro also detects behavior like this and calls it, appropriately, “weird.” Further analysis on the data can be performed by parsing Bro’s anomaly output file, weird.bro.

Tommy Adams, tommy@adamsfam.org
2. **Snort – a primer to a longstanding IDS**

Can an attacker perform IDS evasion or insertion using TFO? Let’s remind ourselves of a few key Snort factors before we dig into these types of questions.

Snort initially looked only at individual packets. If a Snort rule was looking for content of “/etc/passwd” and this data was in a single packet, Snort could detect this. If that same content is spread over two packets, where packet-1 includes “/et” and packet-2 includes “c/passwd” Snort would miss this content in context; meanwhile the destination host would reassemble the packets and see the full payload. The above scenario is an example of IDS evasion – the “malicious” traffic that a host will reassemble evades the IDS. Another impact of the packet-by-packet approach is the potential for false positives. If Snort has a rule looking for that same content and receives a single packet with the ACK flag set and the content includes “/etc/passwd,” Snort will alert. However, a destination host that receives a single packet with the ACK flag set, where there was no existing TCP connection, will drop that packet out of its buffer once it expires or the buffer is full. So, in this example, the Snort alert is a false-positive. The above scenario is an example of IDS insertion – the IDS is alerting on traffic that the host will either not see or will not process. Snort’s stream5 preprocessor is a solution for each of the previous examples. Stream5 builds a stream of data, based on an established TCP-connection or a group of UDP packets where the source and destination IP addresses, as well as the ports, all match. In other words, the Stream5 preprocessor allows Snort contextual insight into a group of packets and allows Snort to assemble and inspect packets in the same way that the destination host will – at least, that is the goal! Snort
rules that contain the keyword “flow” make use of the Stream5 preprocessor, we’ll dig a little more into this when we are inspecting traffic later in this document. For the moment, think about the impact of TCP Fastopen’s use of sending data before the 3WHs is complete, and how that could impact rules that rely on flow, and therefore the Stream4 preprocessor, to find content. We’ll do more discovery on this concept, but first let’s take a packet capture file (pcap) that includes content “/etc/passwd,” where the data occurs after the TCP 3WHs is complete and run that data against Snort.

Here is a simple Snort rule that is intended to alert where there is a match for the string “passwd”:

```
alert tcp any any -> any 80 (msg: "Found passwd"; flow: to_server,established; content: "passwd"; sid: 10000003 7;)
```

How to write Snort rules is out of scope for this document, but a brief description of the above rule is warranted. The rule will alert on traffic from any source IP address and any source port, to any destination IP address where the destination port is 80, and there is an established TCP connection from a client to a server, and there is plaintext content of “passwd” in the TCP stream.

The following is tcpdump command and output of a packet which uses TFO and which has content that includes “passwd”, we’ll look only at the specific packet with the malicious content, rather than all packets, and the output has been edited for brevity:

```
$ tcpdump -nnttX -r tcp-fastopen-2nd-stream.pcap
IP 192.168.0.212.37745 > 192.168.0.33.80: Flags [S], seq 3976103259:3976103345, win 29200, options [mss 1460,sackOK,TS val 14248101 ecr 0,nop,wscale 7,exp-tfo cookie 261fb060cecab690], length 86
  0x0000:  4500 009e edfe 4000 4006 7167 0a63 6301 E.....@.@.qg.cc.
  0x0010:  0a63 632d d8ee 0050 ecfe 855b 0000 0000  .cc -......P...
```

Tommy Adams, tommy@adamsfam.org
In the above output, we see in the first gray-highlighted section, that this packet includes the TFO cookie and this is the initial SYN packet. In the second gray-highlighted section, we see the content includes data on the SYN; the keyword “passwd”, and as we look at the entire payload we see this is a GET request where the client is attempting to obtain the contents of the server’s /etc/passwd file. The method shows is an attempt to exploit a vulnerability called local file inclusion (LFI), and if the server is vulnerable to this, it will dump the contents of that file in its response back to the client.

An attacker can use the contents of the /etc/passwd file from the server to learn more about the users and groups on the server, and now how some of the necessary data to perform an offline password cracking attack. So, we’ve seen data about the attack, and the content includes the string “passwd;” let us run that same packet against the simple rule we saw above and see if the rule will detect the malicious data.

```
$ snort -q -K none -A console -c local.rules -r tcp-fastopen-2nd-stream.pcap
```

Excellent! We see that the snort rule can detect this malicious traffic.

Tommy Adams, tommy@adamsfam.org
3. Analysis Methodology - Introduction

Before we attempt manipulating TFO packet data to see if we can bypass Snort, or create false positives; let’s look at the methodology to analyze packet data using the Security Onion distribution. This methodology is intended to be high-level; as there are many available resources for how a person may analyze data with the tools included with Security Onion. For this paper, the open-source IDS will use the tools and resources available in the Security Onion 16.04 distribution.

To analyze experimental protocols, we’ll focus primarily with Scapy, Wireshark, and Snort. This type of analysis will use stand-alone pcap files; it is, therefore, best done outside of a production network security monitoring (NSM) environment. A production NSM environment will be receiving and processing live data from a network TAP or mirror/SPAN port. What we’re doing is stand-alone analysis; even if it were possible to use a production sensor, it is likely to overcomplicate analysis with unnecessary data. An additional complication is that if a person were to replay a pcap file to a production environment using a tool like tcpreplay, the timestamps will change and this data will be appended to the broader NSM logs; this could introduce unnecessary confusion during incident response. To keep things simple, and to keep stand-alone analysis and data outside of a production environment, please consider the following approach which makes use of the following baseline host:

- Virtual Machine software: (e.g., VirtualBox or VMWare)
  - Security Onion 16.04
    - Single NIC
    - Evaluation mode

Tommy Adams, tommy@adamsfam.org
Our analysis methodology is straightforward:

- Obtain data (e.g., pcap files)
- Process against Security Onion
- Adjust data for evil (e.g., via Scapy)
- Re-process against Security Onion
- Community contribution (when applicable)

4. Analysis Methodology – Obtaining Data

Before one can analyze data using IDS tools, the traffic must be discovered and captured. And building one’s lab, with a TFO-enabled web-server and client are out of the scope of this paper. TFO is experimental, so “firing up Wireshark and filtering for TFO” is not likely an efficient approach. If a person can invest the time to hunt for local TFO traffic, they may need to configure their host or browser first. For example, configure Firefox to permit TFO via the about:config settings (Figure 4) and then hunt for TFO enabled web content while capturing traffic using tcpdump, tshark, Wireshark, etc.

However, there is another method to obtain pcap files. Hit up the community!

There are various sources such as the Dshield Slack channel or the Snort and SecurityOnion mail lists (the following pcap, showing TFO via Wireshark, came from the

Tommy Adams, tommy@adamsfam.org
Dshield community). We know that a key aspect of TFO is to begin data transfer before the three-way handshake (3WHS) is complete. Let’s look at a TFO session in Wireshark:

In the above image packet numbers 1, 2, and 3 show the TCP 3WHS take place. The first packet uses the TCP SYN Flag (seen at the bottom of the image, by ‘Flags,’) the second packet the SYN and ACK flags, and the third packet the ACK flag. But this is different than the earlier view we took at a 3WHS. In this image, we see a ‘Client Hello’ in the first packet and find that the packet is 52 bytes; this very first packet contains data. While the TCP Protocol (RFC 793) does not prohibit data on the initial SYN packet, it is unusual, and in this case, the data is a TFO cookie (not shown in Wireshark). With typical TCP, if there is data on the SYN, the receiving host does not send the data to the application layer until the 3WHS is complete. If we had looked at the packets preceding the 3WHS, we would see additional packets containing data until the session is complete. Once we have data, we need to analyze it as-is, and then adjust the data with an intent to make it malicious in some way. We’ll start by inspecting pcap data as-is with Security Onion.

Tommy Adams, tommy@adamsfam.org
5. Analysis Methodology – Process Data against Security Onion

We’ll use a relatively new (at the time of this edit) script by Doug Burks called so-import-pcap which he describes as “… a quick and dirty EXPERIMENTAL script that will import one or more pcaps into Security Onion and preserve original timestamps.” In fact, the author recommends that a snapshot of Security Onion is taken before running the script; I took his advice and did not have any issues. The following command makes use of this script and imports a pcap file for analysis:

```
trfflnhr@so-eval-16:~$ sudo so-import-pcap pcaps/filename.pcap
```

The resulting output is lengthy but includes important facts. The output results were edited for brevity and clarity:

```
so-import-pcap
This is a quick and dirty EXPERIMENTAL script that will import one or more pcaps into Security Onion and preserve original timestamps.

It will do the following:
- generate IDS alerts using Snort or Suricata
- generate Bro logs
- store IDS alerts and Bro logs with original timestamps

Warnings:
- Do NOT run this on a production deployment. It is designed for standalone systems designated for so-import-pcap.

Press Enter to continue or Ctrl-c to cancel.

Please wait while...
...creating temp pcap for processing.
...analyzing traffic with Snort.
...analyzing traffic with Bro.
...writing /nsm/sensor_data/so-eval-16-ens33/dailylogs/2018-05-02/Snort.log.1525219200

Import complete!

You can use this hyperlink to view data in the time range of your import: 
<link redacted>
```

Tommy Adams, tommy@adamsfam.org
or you can manually set your Time Range to be:
From: 2018-05-02    To: 2018-05-03

In addition to the important warning and informational data, the output results confirm that the pcap data is analyzed by core IDS tools (e.g., Snort, Bro) and prepared for human analysis using these tools directly or any other appropriate tool provided by Security Onion (e.g., Sguil, Squert, Kibana). The output even provides a link that will launch Kibana and auto-filter for the time range of the import, which makes “finding the data” a non-issue for the analyst. Now that the pcap is processed by the stand-alone sensor let’s look to see if any Snort alerts are generated.

There are a variety of tools that could be used to view Snort alerts. We’ll use Sguil as the quick insight it provides to any correlating Snort rule, and ability to pivot
from any alerts to other tools is helpful.

![Image of SGUIL interface](image.png)

Figure 5

In the above image, we see… absolutely nothing! The snort signatures are current, and yet no Snort rules alerted on the pcap data. Why is this? Because TFO traffic is not inheritably evil, it is simply experimental, and this particular packet has nothing malicious in it. So what happens if take an existing TFO packet capture and attempt to it evil – make it bypass a snort rule that would catch the same payload or content that used a typical 3WHS or make the data appear malicious to Snort, causing (false positive)

Tommy Adams, tommy@adamsfam.org
alerts, but crafting the data that will be discarded by hosts? To accomplish this we’ll use Scapy.

6. Analysis Methodology – Adjust Data… for evil

Detailed information about, or instructions on the use of Scapy is out of the scope of this paper; there are several related links in the Further Reading section below for additional study. Let’s focus narrowly on how (bypassing much of the why) to modify the TFO packet to insert malicious content into the stream. Rather than build packet data layer by layer, we’ll take the following actions:

1. Import a pcap file into Scapy:

>>> tfo=rdpcap("/home/pcaps/tfo.pcap")

2. Display the raw data from the specific packet that we want to modify (in this case the third packet (remember to start counting at 0), the Ethernet, IP, and TCP layers have been snipped from the output):

>>> tfo[2]
<Raw load='GET /index.php HTTP/1.0\r\nUser-Agent: Mozilla/5.0\r\nAccept: */*

3. Add (or replace) the raw data in the specific packet that we want to modify:

>>> tfo[2][Raw]=load='GET /index.php?page=../../../etc/passwd HTTP/1.0\r\nUser-Agent: Mozilla/5.0\r\nAccept: */*

4. Fix checksum issues (the following is a simple example, typically the checksums must be removed at each layer, for each packet; so that Scapy will recalculate them. For more information on this see the Appendix.):

>>> del tfo[2][TCP].chksum

5. (Optional) Show the new packet to verify changes (the Ethernet and IP layers are snipped from the output for brevity and clarity):

>>> tfo[2]
<Raw load='GET /index.php?page=../../../etc/passwd HTTP/1.0\r\nUser-Agent: Mozilla/5.0\r\nAccept: */*

6. Export the modified data into a new pcap file:

>>> wrpcap("/home/pcaps/tfo_evil.pcap", tfo)

In the above example, we took an existing packet and re-crafted it to introduce malicious content. We can use the resulting pcap file and inspect it with our NSM tools, using the Security Onion script, so-import-pcap, as shown earlier, or use

Tommy Adams, tommy@adamsfam.org
tcp_replay. Additionally, we can analyze the new pcap file with Wireshark, tcpdump, etc.

But let’s reconsider the unique attributes of TCP Fastopen and wonder how it could be used to create false positives. This critical thinking approach is a requirement of testing experimental protocols against IDS. We turn again to RFC 7413 to better understand this experimental protocol.

We see in the RFC that if a server receives the initial SYN with a TFO cookie request, and the server does not support TFO, the session will be downgraded to regular TCP. This downgrading is also intended to take place when the \texttt{PendingFastOpenRequests} is exceeded on a server, this prevents resource exhaustion via an SYN flood (with valid TFO cookies) attack. Similarly, if a server receives a TFO request SYN packet, but the cookie is invalid, the server is to drop the data from the SYN and reply with a SYN-ACK acknowledging the SYN sequence number only. Another scenario where the connection starts out attempting TFO and is then downgraded to “regular TCP” is when the SYN exceeds the TCP option space for the TFO option.

Finally, the RFC makes it clear that the data on the SYN of a TFO session is delivered to the receiving host application layer “right away”, before the completion of the 3WHS. This is different that TCP traffic following the RFC 793 protocol, which does not deliver the SYN data to the application layer until the 3WHS is completed. Each of these are of interest, in that they are unique to TFO and could perhaps be used for an IDS-related attack. Let’s look at one of these factors and see if there is the possibility of crafting a malicious packet.

Tommy Adams, tommy@adamsfam.org
We will again import a non-malicious TFO packet capture into Scapy and modify it. In this case, rather than adding malicious content to the packet that we hope will both bypass the IDS (cause a true negative, no alert) and deliver malicious data to the destination host, we will craft a packet that we expect the destination host will discard, but which will cause the IDS to alert (creating a false positive).

1. **Import a pcap file into Scapy:**

   ```
   >>> tfo2=rdpcap("/home/pcaps/tfo.pcap")
   ```

2. **Replace the SYN packet data with malicious data:**

   ```
   >>> tfo2[0][Raw]=load='GET /index.php?page=../../../etc/passwd
   HTTP/1.0\r\nUser-Agent: Mozilla/5.0\r\nAccept: */*
   '
   ```

3. **Re-craft the SYN packet to exceed option space** (the default is 536 bytes, or it could be up to the cached server MSS, so experiment with the number of bytes):

   ```
   >>> tfo2[0][TCP].options
   [(\'MSS\', 1460), (\'SAckOK\', \'\'\'), (\'Timestamp\', (14248099, 0)), (\'NOP\', None), (\'WScale\', 7), (254, '\xf9\x89')]  
   ```

4. **Fix checksum issues** (the following is a simple example, typically the checksums must be removed at each layer, for each packet; so that Scapy will recalculate them. For more information on this see the Appendix.):

   ```
   >>> del tfo2[TCP].chksum
   ```

5. **Export the modified data into a new pcap file:**

   ```
   >>> wrpcap("/home/pcaps/tfo_evil_2.pcap", tfo2)
   ```

To summarize the above actions, we crafted a malicious packet that includes malicious data on the SYN, and which is likely to exceed the server MSS for this packet, causing the server to downgrade the connection from TFO to TCP. Of critical importance is that the receiving host will not pass up this data to the application layer, but Snort may see that a connection is established and thus alert on the data in the SYN. Will Snort give us a false-positive on the newly crafted packet?

Tommy Adams, tommy@adamsfam.org
7. Analysis Methodology – Reprocess the “evil” data

Generally, we would export the crafted data from Scapy and then use so-import-pcap to import the pcap into Security Onion. Once the data is imported, we can then look at it via Sguil, look at snort alerts, nsm data (e.g., /nsm/sensor_data/so-eval-16-ens33/dailylogs/2004-12-23/snort.log.1103760000), etc. Let’s look at the second packet, to see if it causes a Snort alert. Specifically, if the initial SYN packet, which has the malicious payload that will not be passed up to the application layer by the receiving host, will cause a false-positive Snort alert. Let’s run this new packet against the simple Snort rule we used previously:

```
$ snort -q -K none -A console -c local.rules -r tcp-fastopen-2nd-stream.pcap
```

```
```

Snort does alert on this packet, which we expect to be a false positive – as we expect the destination host to drop the data from the SYN when the connection is downgraded from TFO to TCP. False positives are a serious problem for analysts as a person could detune alerts to avoid these false positives and then potentially miss actual issues where an experimental protocol such as TFO is used for malicious purposes.

8. Analysis Methodology – Adjust Data… for evil – Part 2

We’ve looked at re-crafting an experimental protocol for IDS insertion, let’s see if we can make something happen with IDS evasion. Can we take advantage of the fact that TFO places data on the SYN to bypass the simple Snort rule we’ve been testing? The following pcap file, crafted by Judy Novak, uses the TFO protocol and includes the

Tommy Adams, tommy@adamsfam.org
content “passwd.” However, this time, the content of “passwd” is split between two packets. The content of “pass” is found at data on the SYN which is normal behavior for TFO, but atypical for regular TCP, and the remaining content, “wd,” is located in a subsequent PSH/ACK packet. Remember that our Snort rule uses the stream5 preprocessor to assemble a TCP stream so that the rule can process content in the same way that the receiving host will see the data. Let’s inspect this new packet, the one intended to evade Snort, first with tcpdump, and we’ll look at only those two packets with the malicious payload.

```
$ tcpdump -nntX -r new-tcp-fastopen.pcap
IP 192.168.0.212.37745 > 192.168.0.33.80: Flags [S], seq 3976103259:3976103297, win 29200, options [mss 1460,sackOK,TS val 14248101 ecr 0,nop,wscale 7,exp-tfo cookie 261fb060cecab690], length 38: HTTP: GET //index.php?page=../../../etc/passwd
```

```
0x0000: 4500 006e edfe 4000 4006 7197 0a63 6301
  .n..@.@.q..cc.
0x0010: 0a63 632d d8ee 0050 ecfe 855b 0000 0000 .cc-
  ....P....[
0x0020: d002 7210 55d8 0000 0204 05b4 0402 080a
  ..r.U.........
0x0030: 00d9 68a5 0000 0000 0103 0307 fe0c f989
  ..h...........
0x0040: 261f b060 cece b690 4745 5420 2f2f 696465782e7068703f706167653d2e2e2f70617373
  &..`...GET.//index.php?page=../../..etc/passwd
```

```
IP 192.168.0.212.37745 > 192.168.0.33.80: Flags [P.], seq 3976103298:3976103346, ack 0, win 29200, options [TS val 14248101 ecr 0,nop,nop], length 48: HTTP
```

Tommy Adams, tommy@adamsfam.org
We see the content of “passwd” split between the two packets, highlighted in gray above. Now let’s look the same pcap with Wireshark – and we’ll follow the TCP stream to see how the receiving host will process the data.
We see in the red highlight that the sent packets will be assembled to deliver the malicious payload – the local file inclusion attack that uses GET to attempt to capture the contents of the destination hosts /etc/passwd file. The response, in blue highlight shows the destination host is responding to the attacker with the contents of its /etc/passwd file. Now the attacker can potentially crack passwords and access the host. So we’ve confirmed this data is malicious, let’s process it against our IDS tools.

Tommy Adams, tommy@adamsfam.org
9. Analysis Methodology – Reprocess the “evil” data –

Part 2

Will our IDS tools alert to the new pcap, once which we know is efficiently malicious to the end host? Rather than show you an empty (spoiler-alert!) Sguil screen, let’s simple glance back at the same Snort rule we used previously, and then process this new pcap through Snort.

```bash
$ cat local.rules
alert tcp any any -> any 80 (msg: "Found passwd"); flow: to_server,established; content: "passwd"; sid: 100000037;)

$ snort -q -K none -A console -c local.rules -r new-tcp-fastopen.pcap
```

We see above that this same Snort rule will not detect this attack. This is classic IDS evasion. The crafted packet does contain the content of password but this specific Snort rule was not able to find the match and therefore did not alert.

10. Community #Contribute

The process of testing any protocols can be tedious, especially experimental protocols where some behavior by either the sending host, receiving host, or middleboxes may be unexpected or unplanned. A person should consider hitting up various Defender and NSM communities to obtain pcap files (or other log data) for analysis. Another recommendation is that research is shared. For example, ask peers to confirm what you are observing, to request help deciphering or processing data, or to ask folks to test out a Snort rule that you have created in response to your research.

Tommy Adams, tommy@adamsfam.org
Experimental RFCs are not new, and there will always be new ones as applications and networks change over time. We can and should process new protocols, or adjustments to old protocols, against our current IDS and share our findings with the broader community.
References


KeyCDN website. TCP Fastopen Image retrieved April 23, 2018, from https://www.keycdn.com/support/tcp-fast-open/


Tommy Adams, tommy@adamsfam.org
Processing experimental protocols against IDS

Snort IDS website. *README.decode* Retrieved June 19, 2018, from

https://Snort.org/faq/readme-decode

Testing for Local File Inclusion, OWASP. Retrieved August 2, 2018, from

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion


S7 Communication (S7comm), a Siemens proprietary protocol. Retrieved June 20, 2018, from https://wiki.wireshark.org/S7comm


Bro 2.5.4 documentation, *weird.bro*. Retrieved June 20, 2018, from


Virtual Box Download. Retrieved June 22, 2018, from

https://www.virtualbox.org/wiki/Downloads

Virtual Box License and Use information, Licensing FAQ. Retrieved June 22, 2018, from

https://www.virtualbox.org/wiki/Licensing_FAQ


Placeholder – if I need to Ref SANS Courseware then Day 2: 89-94, 112, 150-152 &&

Day 3: 181-188, 205

Tommy Adams, tommy@adamsfam.org
Appendix

Security Onion is a “free and open source Linux distribution for intrusion detection, enterprise security monitoring, and log management. It includes Elasticsearch, Logstash, Kibana, Snort, Suricata, Bro, OSSEC, Sguil, Squert, NetworkMiner, and many other security tools. The easy-to-use Setup wizard allows you to build an army of distributed sensors for your enterprise in minutes!” --- https://securityonion.net/

Dshield Slack channel: dshieldusers.slack.com

SecurityOnion Google group: https://groups.google.com/forum/#!forum/security-onion

Snort mail list: https://www.snort.org/community

Figure Table

Figure 1: A screenshot of Wireshark, data by the author

Figure 2: Image retrieved from KeyCDN

Figure 3: Table created using as inspiration the S7comm table found at the WiresharkWiki

Figure 4: Firefox about:config – enabling TFO

Figure 5: Sguil with no alerts

Figure 6: Wireshark – Follow TCP Stream

Further Reading

TCP Fastopen:
Shaving your RTT with TCP Fastopen, by Bradly Falzon
https://bradleyf.id.au/nix/shaving-your-rtt-wth-tfo/

Scapy:
What I learned at Camp [editing pcap files with Scapy], by Judy Novak

Scapy Cheat Sheet, SANS Institute
https://blogs.sans.org/pen-testing/files/2016/04/ScapyCheatSheet_v0.2.pdf

The Very Unofficial Dummies Guide To Scapy, by @catalyst256

Tommy Adams, tommy@adamsfam.org
Appreciation

Judy Novak – suggesting TCP Fastopen as a topic and providing several pcap files

Johannes Ulrich – providing a pcap file for TCP Fastopen (via the DShield slack group)

D’Arcy Davis - encouragement

Jeff Lake – threats + encouragement + threats

Avalon Biscuit Bar and The Commons – public wifi (with a VPN, of course!) + coffee + snacks

Kristin Adams – unending beauty + creating space for this project

Tommy Adams, tommy@adamsfam.org
<table>
<thead>
<tr>
<th>Country/Region</th>
<th>City</th>
<th>Start Date</th>
<th>End Date</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Northern VA</td>
<td>Reston</td>
<td>Mar 02, 2020</td>
<td>Mar 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich</td>
<td>Munich</td>
<td>Mar 02, 2020</td>
<td>Mar 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Secure Japan</td>
<td>Tokyo</td>
<td>Mar 02, 2020</td>
<td>Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Jeddah March</td>
<td>Jeddah</td>
<td>Mar 07, 2020</td>
<td>Mar 12, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS St. Louis</td>
<td>St. Louis</td>
<td>Mar 08, 2020</td>
<td>Mar 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dallas</td>
<td>Dallas</td>
<td>Mar 09, 2020</td>
<td>Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Prague March</td>
<td>Prague</td>
<td>Mar 09, 2020</td>
<td>Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Wild West Hackin</td>
<td>San Diego</td>
<td>Mar 10, 2020</td>
<td>Mar 11, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Doha March</td>
<td>Doha</td>
<td>Mar 14, 2020</td>
<td>Mar 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Francisco Spring</td>
<td>San Francisco</td>
<td>Mar 16, 2020</td>
<td>Mar 27, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Norfolk 2020</td>
<td>Norfolk</td>
<td>Mar 16, 2020</td>
<td>Mar 21, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Kuwait March</td>
<td>Salmiya</td>
<td>Mar 21, 2020</td>
<td>Mar 26, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Abu Dhabi March</td>
<td>Abu Dhabi</td>
<td>Mar 28, 2020</td>
<td>Apr 02, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Frankfurt March</td>
<td>Frankfurt</td>
<td>Mar 30, 2020</td>
<td>Apr 04, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS 2020</td>
<td>Orlando</td>
<td>Apr 03, 2020</td>
<td>Apr 10, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Muscat April</td>
<td>Muscat</td>
<td>Apr 04, 2020</td>
<td>Apr 09, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Riyadh April</td>
<td>Riyadh</td>
<td>Apr 04, 2020</td>
<td>Apr 16, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Bethesda</td>
<td>Bethesda</td>
<td>Apr 14, 2020</td>
<td>Apr 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Minneapolis</td>
<td>Minneapolis</td>
<td>Apr 14, 2020</td>
<td>Apr 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels April</td>
<td>Brussels</td>
<td>Apr 20, 2020</td>
<td>Apr 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Blue Team Summit &amp; Training 2020</td>
<td>Online</td>
<td>Mar 02, 2020</td>
<td>Mar 09, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s Only</td>
<td>Anytime</td>
<td></td>
<td>Self Paced</td>
</tr>
</tbody>
</table>