White Paper

Reza Saadat, Sr. Technical Marketing Engineer,
Spirent Security & Applications

Transport Layer Security v1.3 Fuzz Testing

Improving Security and Robustness of Cloud, IoT, and Internet Services

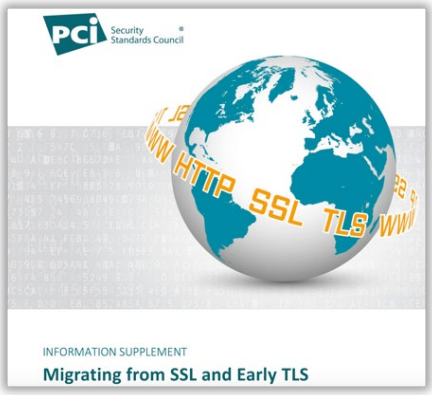
Abstract/Executive Summary

For over two decades, Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) have been responsible for securing the communication in the network and between smart devices. The history of these technologies has been marked by significant cryptographic breaks and implementation flaws with exploits wreaking havoc upon enterprises and the public. TLS v1.3 is a huge leap forward for web-based encrypted communication with improved security, and performance compared to its predecessors. In these early stages of TLS v1.3 deployment and with the rise of encrypted traffic volume, it is prudent to gain realistic, preemptive and actionable intelligence through performance/scalability as well as negative testing of the services that leverage this latest version of transport layer security. Fuzz testing delivers randomness as part of testing strategies that can reveal undetected bugs and compromises that would otherwise get exploited to breach your defenses.

PCI Security Standard Council®:

"It is critically important that entities upgrade to a secure alternative as soon as possible, and disable any fallback to both SSL and early TLS. SSL/early TLS was removed as an example of strong cryptography in PCI DSS v3.1 (April 2015)."

https://www.pcisecuritystandards.org/documents/Migrating-from-SSL-Early-TLS-Info-Supp-v1_1.pdf



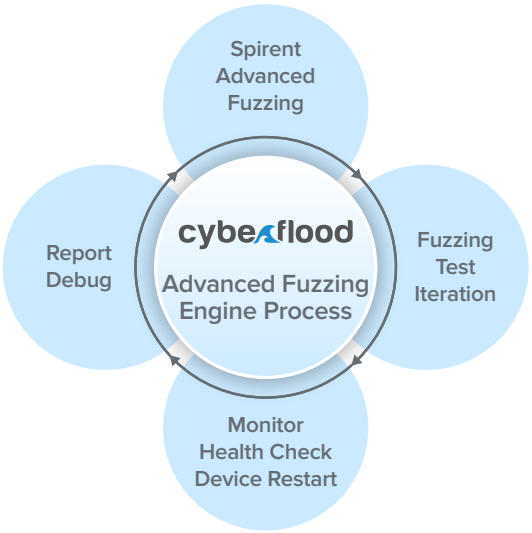
Introduction

With the arrival of TLS v1.3, the wait for new and improved transport layer encryption protocol is over. It has been over eight years since the last update (TLS v1.2) which was defined in RFC 5246 (https://tools.ietf.org/html/rfc5246). The goal of TLS is essentially to prevent eavesdropping, tempering and message forgery while client/server applications are in communication over the internet. TLS v1.3 promises to be faster and more secure than its predecessors. Given the importance and momentum that encrypted traffic has in relation to enterprise web traffic landscape, TLS v1.3 will play an important role for consumers as well as providers of services that are based on this encrypted transport layer technology. Gartner predicts that by 2019 more than 80% of enterprises' web traffic will be encrypted.[1] In order to make emerging encryption deployments effective in terms of enabling improved privacy and security, it is imperative that organizations proactively investigate viability, and robustness of encrypted transport layer technologies as well as solutions that may be based on such services. Vulnerabilities in the implementation of transport layer protocols have been exploited for malicious attacks as we have seen in recent years. For example, 'Heartbleed' is one of the most significant security bugs of all time that was exploited in the code for TLS extension of OpenSSL library—what issues may lurk in TLS v1.3 implementations are not known.

One of the more promising approaches to uncovering bugs missed in manual audits of code implementation while providing an overview of the target software's robustness is fuzz testing. Fuzz testing or fuzzing delivers invalid, unexpected, or random data to the inputs of a computer program, operating system, or hardware system while monitoring for application or program. Spirent's CyberFlood Advanced Fuzzing provides a unique approach to fuzz testing called SmartMutation™ based fuzzing that would essentially offer unlimited number of test inputs to test the resiliency of the target. Furthermore, CyberFlood is a hyper-realistic L4-L7 traffic generator that can do further stressing of the device with extreme load of legitimate and malicious attack traffic as well as fuzz testing. It is important to recognize at these initial phases of TLS v1.3 deployment rollouts, leveraging tools and strategies that would provide visibility for further hardening of emerging diverse and unproven TLS v1.3 implementations are essential. CyberFlood advanced fuzzing can provide visibility to developers as well as network and cyber security specialists for implementation areas that are vulnerable to invalid or random data inputs.

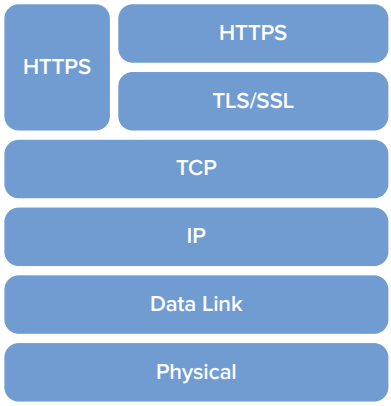
CyberFlood advanced fuzzing engine is the heart of Spirent's fuzz testing approach and it offers framework driven testing solutions, with SmartMutation-based test cases that provide coverage to a wide range of recent protocol implementations. Test repeatability is guaranteed through supported fuzzing seed values that are used to easily alter the negative inputs and allow for generating the same mutation over time. Other significant features of CyberFlood advanced fuzzing are: Reports, Monitors and Restarters. Reports provide detailed information on hard faults that are reproducible, in addition to warning faults that happen once but are not reproduced. Monitors are an essential component of CyberFlood advanced fuzzing that offer multiple ways of instrumenting and measuring service activities to confirm the service being tested is still operational. Lastly, Restarters provide means of bringing back up the test target after a non-recoverable fault while providing flexibility of different methods of getting them restarted.

[1] Gartner Predicts 2017: Network and Gateway Security–Published, 13 December 2016



Evolution of Securing Transport Layer

TLS and its predecessor, SSL have become an important and integral part of securing communication of client/server applications for HTTP (HTTPS), SMTP, POP3, FTP and many more IP-based protocols. These transport layer security protocols are implemented on top of transport layers such as TCP. For example, following illustrates such HTTP(S) – TLS/SSL protocol layers at a high level.



The initial version of SSL v2 was drafted in 1995 by Netscape Communications to extend HTTP and it was intended to establish secure connections on unsecured networks using a number of techniques including cryptography and Public Key Infrastructure. SSL v2 had all the main pieces in place however it did have a few fatal flaws that resulted in getting SSL v2 deprecated. Examples of SSL v2 underlying weaknesses are:

- **Man-in-the-middle attack vulnerability during initial handshake:** In SSL v2, initial packets (i.e., Client Hello and Server Hello) are not protected against man-in-the-middle attack. The connection can be downgraded by an attacker inserting himself between client and server during initial handshake and eliminating all encryption options except weakest/easiest to crack ciphers.
- **Truncation attack vulnerability:** In SSL v2, there is no specific shutdown sequence as there is in TLS. Either side can simply send a regular TCP FIN when either side is done using a connection and therefore the peer will not be able to determine if it is a legitimate end of data or not if attacker forges a TCP FIN.
- **Same cryptographic key used for encryption and authentication:** A negotiated weak encryption scheme would result in the message authentication code using the same weak key.

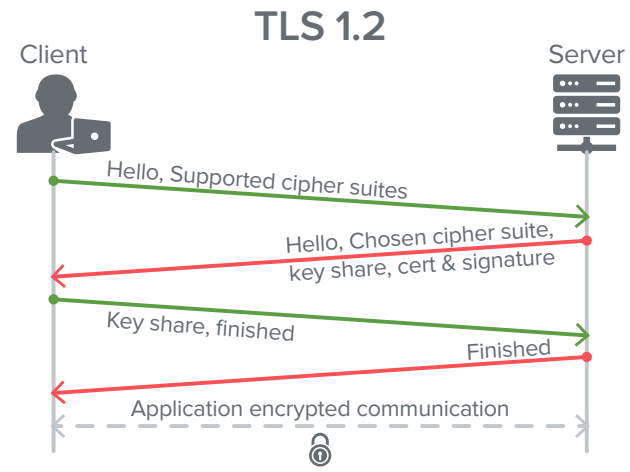
Above are a few of issues found in SSL v2 after it was submitted for review and consequently SSL v2 was withdrawn and followed up with SSL v3 by Netscape. During ratification of SSL v3 by IETF (Internet Engineering Task Force)[2], a few minor changes were made of which a superficial one was to change the name from SSL to TLS and therefore SSL v3.0 became TLS v1.0 and approved (https://tools.ietf.org/html/rfc2246) in January 1999. It is worthwhile noting that SSL v3 was never ratified and it is not interoperable with TLS v1.0.

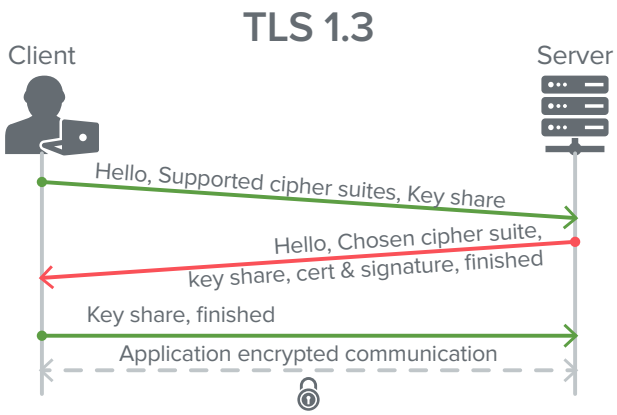
It took over 5 years—since 2006, before a new version of TLS protocol is released as TLS v1.1 (https://tools.ietf.org/html/rfc4346) with small security improvements as well as clarification and editorial improvements over TLS v1.0. In 2008, TLS v1.2 (https://tools.ietf.org/html/rfc5246) with major improvements, particularly for negotiation of cryptographic algorithms and addition of support for authenticated encryption was approved. And finally in 2013, the work on TLS v1.3 officially began and its readiness was assessed in 2016 (https://www.internetsociety.org/events/ndss-symposium-2016/tls-13-ready-or-not-tron-workshop-programme).

[2] Current TLS v13 draft: https://tools.ietf.org/html/draft-ietf-tls-tls13-21

TLS v1.3 vs. TLS v1.2—Improved Performance, Security, and Robustness

Encrypted protocols such as TLS have added overhead similar to other transport protocols. In this case, TLS v1.3 optimizes the initial handshake and reduces the latency between client and server as illustrated in the example below.





TLS v1.3 can optimize the initial handshake by sending a key share at the initial message based on making a guess as to which key agreement the server would choose. This exchange requires only one round-trip before the client can send the request and that can save time.

There are further improvements in terms of load time compared to TLS v1.2. The notion of TLS v1.2 "Resumption" is enhanced so in TLS v1.3 the client remembers that it has connected to a (TLS v1.3) server before and it can send data on the first message and therefore can start sending encrypted data without prior round-trip exchanges.

Besides above improvements in the handshake, TLS v1.3 also improves robustness of crypto support by removing obsolete and insecure features from TLS v1.2 such as:

- Static RSA handshake (offers no forward secrecy, DROWN vulnerability—Decrypting RSA with Obsolete and Weakened eNcryption)
- SHA-1, MD5 (SLOTH vulnerability—Security Losses from Obsolete and Truncated Transcript Hashes)
- Static Diffie-Hellman
- RC4 cipher suites
- Weak and lesser-used named elliptic curves
- Non-AEAD (Authenticated Encryption with Associated Data) ciphers
- Compression feature

By removing above, TLS v1.3 protocol is much more simplified compared to its predecessors and reduces the chance for exposure to the underlying vulnerabilities. Furthermore, simpler protocol means reducing chances of implementation flaws.

Despite TLS v1.3 leap forward in terms of improving performance and security which can benefit enterprises and consumers, there are actions that can be taken proactively by providers and consumers of TLS v1.3 services which will further assist in hardening the deployment or the offering. CyberFlood SmartMutation fuzz testing with its unique intelligence driven strategy can uncover more issues in more code paths by providing orders of magnitude higher test cases. Next, we will discuss the technical aspect of CyberFlood advanced fuzzing—TLS v1.3.

Cyberflood Advanced Fuzzing TLS v1.3

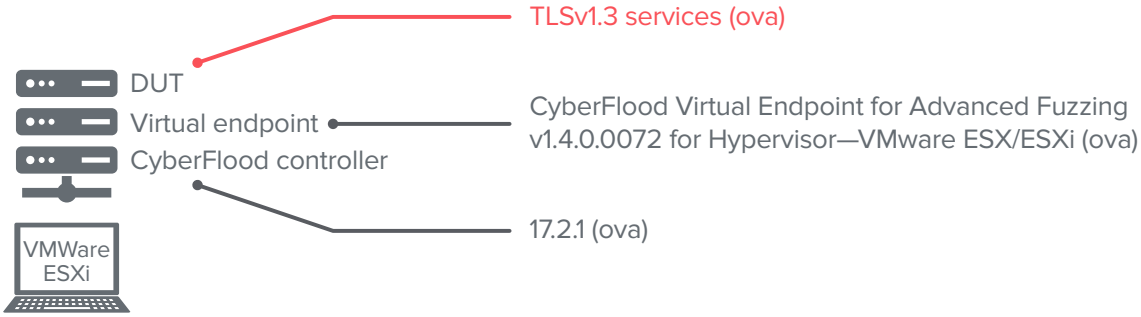
In this section, we will cover detail of using CyberFlood advanced fuzzing capabilities to perform fuzz testing against TLS services provided by a Device Under Test (DUT). Please note that familiarity with CyberFlood Advanced Fuzzing feature and TLS v1.3 is assumed.

CyberFlood 17.2.1 is the first security test product to support TLS v1.3. It supports TLS v1.3 draft-19, draft-20 and draft-21 fuzz testing in relation to the DUT. As stated before, there are benefits for providers as well as consumers of the service to perform this negative testing in order to expose possible underlying vulnerabilities before they are exposed or exploited by malicious attackers. It is worth noting that this type of negative testing is intended to complement a comprehensive test strategy and will not constitute a complete test plan by itself. The focus of this section is to describe an approach in performing fuzz testing with a DUT providing TLS v1.3 services and it is not intended to provide an exhaustive description of CyberFlood advanced fuzzing capabilities.

CyberFlood 17.2.1 supports advanced fuzzing on C100-S3 physical appliance traffic generator as well as virtual endpoints. In this section, we will leverage the virtual endpoint which provides a powerful solution for performing advanced fuzzing without a need for physical appliances. Following diagram illustrates the logical topology of a typical advanced fuzzing setup based on this virtualized solution.

In our case, we use an open virtual appliance file (ova) with Ubuntu 14.04.3 LTS and a commercially available TLS library with TLS v1.3 services as DUT. This approach reveals faults that the vendor uses to rectify issues in their implementation. Installing TLS v1.3 is platform and environment dependent and there may be many options to be considered in installing the service. A few of important options are listed here:

- Use TLS v1.3 flavor of implementation
- Allow multiple connections
- Use cipher suite TLS13-AES-128-GCM-SHA256
- Listen to specified port (6668 in this example)
- Use appropriate certificate file if needed
- Use appropriate key file if needed
- Bind services to data interface
- Echo server detected logging messages for further visibility



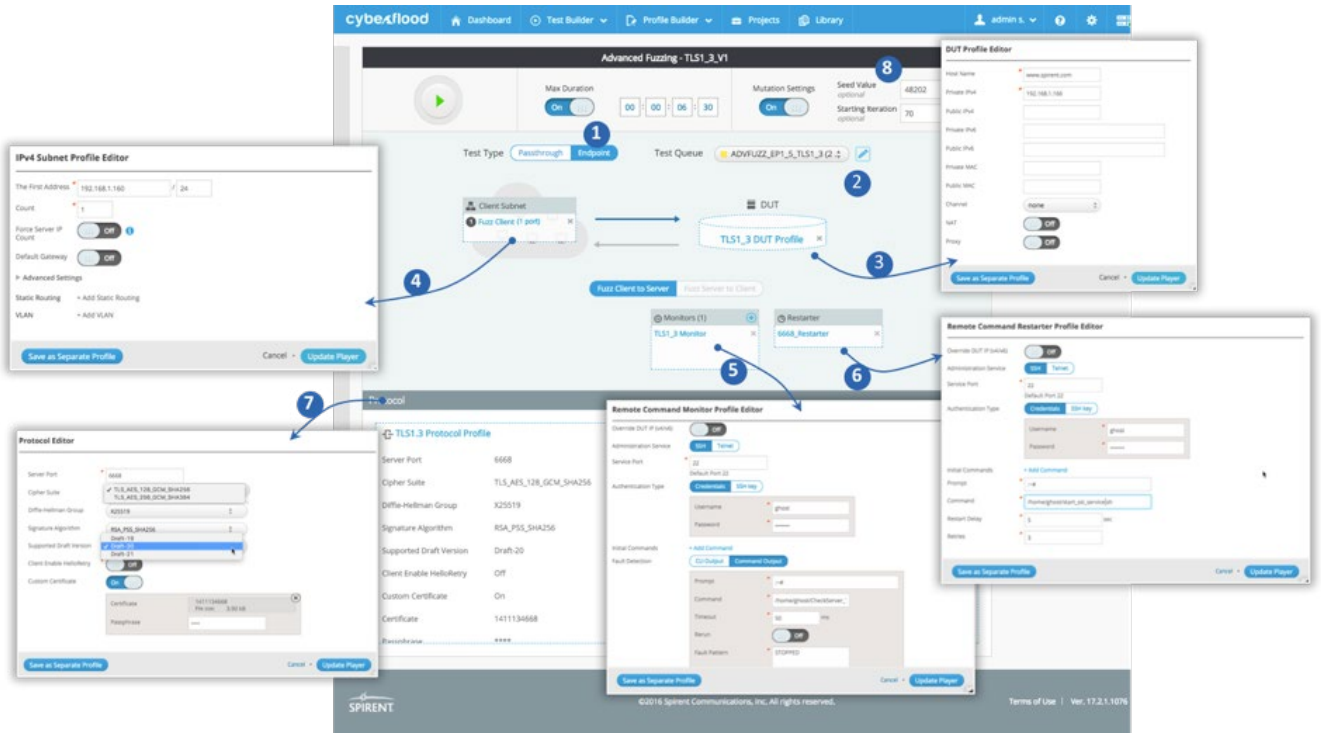
Given above server, we can use CyberFlood to perform TLS v1.3 advanced fuzzing by going to Test Builder > Advanced Fuzzing. The important steps and sub-steps for advanced fuzzing TLS v1.3 are illustrated below.

1. Select fuzz testing with an endpoint (server).
2. Select test queue.
3. Enter a hostname and the IP address as part of testing subnet for DUT.
4. Define client subnet as part of the testing subnet.
5. Configure remote command monitoring to check the health of the service. In this particular case, we will look for the port number of the service (6668) as part of the output for current network connections (linux netstat command). If the connection is no longer there that is an indication of the service termination and that would trigger the fault. Here is sample of a python script that was called from a shell script passing it the port number.
6. Restarter will simply start the sequence to bring up server in case a fault occurs. This sequence would be similar to steps taken for service initialization and bring up.

```
*** CheckServer _ TLS1.3.py
import subprocess
import sys
import time

def do _ cmd():
    cmd = "netstat -ntlpu|grep %s|awk
'{print $1}'" % sys.argv[1]
    out = subprocess.check _
output(['bash', '-c', cmd])
    return out

if _ _ name _ _ == '_ _ main _ _':
  while True:
    out = do _ cmd()
    if 'tcp' in out:
      print 'TLS1.3 PROCESS RUNNING'
      time.sleep(0.5)
    else:
      print 'TLS1.3 PROCESS STOPPED'
      while True:
        print 'CHECKING STATUS...'
        out = do _ cmd()
        if 'tcp' in out:
          print 'TLS1.3 PROCESS BACK UP'
```



7. Protocol configuration has detail of TLS v1.3 that will be fuzz tested. For example, one can pick service port number (6668), TLS_AES_128_GCM_SHA256 as cipher suite, Draft-20 as supported draft version, whether to have client hello retries, or use custom certificate. In this version of CyberFlood, X25519 is the option for Diffie_Hellman Group and RSA_PSS_SHA256 is the option for signature algorithm.

8. The optional seed and iteration values allow repeatability for code paths based on those two values. The CyberFlood advanced fuzzing reports will indicate the seed and the iteration number that faults occur in. This would allow user to repeat same faults using seed or iteration numbers indicated in the report.

At this point, fuzz testing TLS v1.3 can begin but do make sure that the DUT is reachable and TLS v1.3 service is running on the DUT. For this test, an actual TLS v1.3 implementation that is commercially available in the market was used and there were 4 faults that are captured in below report. The report consists of details regarding fuzzing for duration of test and the events that occurred. In this case, monitoring probe detected service termination 4 times, it restarted the service and was able to recreate the condition by backtracking the iteration number. In all 4 cases, fault was declared rather than a warning since they were reproducible. The main points of report generated at end of the run are explained and illustrated below.

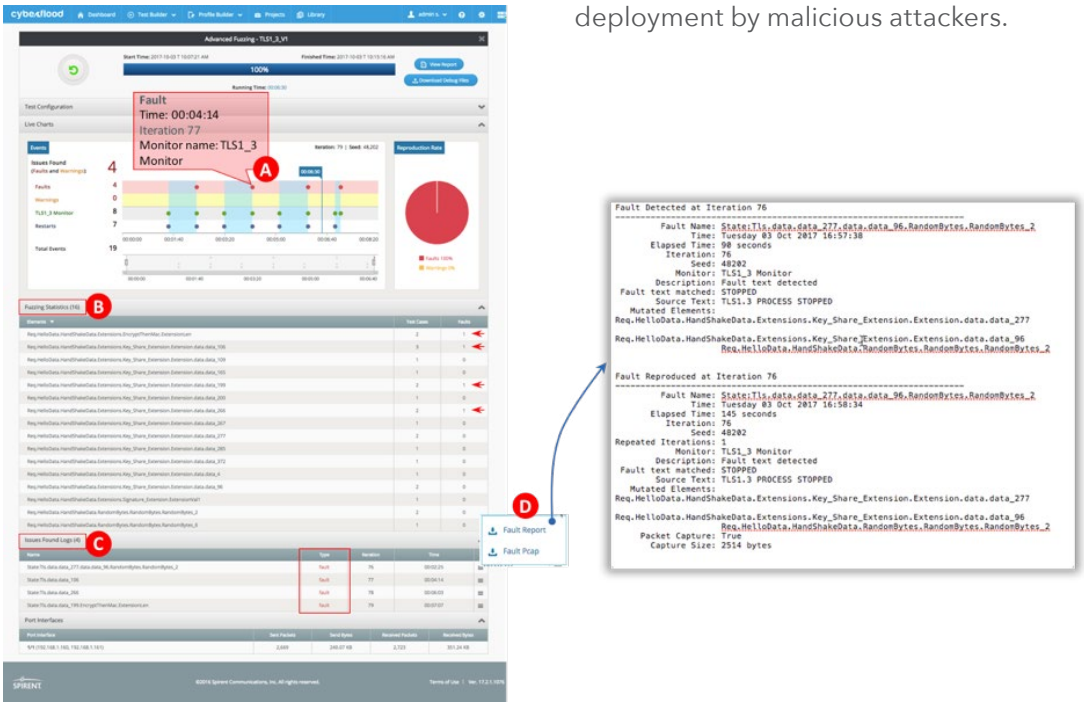
A. Each of the 4 faults are highlighted in the live chart and its timestamp and iteration number is indicated by performing mouse rollover.

B. Fuzzing Statistics will give statistical detail of test cases and where faults occurred. This is a detail accounting of the data modeling elements and they are rolled up into individual faults and matching names in the table with heading “Issues Found Logs”.

C. In this case, each of the four faults and the elements map in to one corresponding name but it is important to note that in some cases there will be multiple faulted elements that would be rolled into a single name entry as part of “Issues Found Logs” table.

D. Further information about each of the four faults can be found by clicking on multi-bar menu options. The “Fault Report” for the first found fault is shown above as well.

Information such as this and DUT logged messages during fuzz testing can indicate vulnerabilities in implementation that may get exploited during deployment by malicious attackers.



About Spirent Communications

Spirent Communications (LSE: SPT) is a global leader with deep expertise and decades of experience in testing, assurance, analytics and security, serving developers, service providers, and enterprise networks.

We help bring clarity to increasingly complex technological and business challenges.

Spirent's customers have made a promise to their customers to deliver superior performance. Spirent assures that those promises are fulfilled.

For more information, visit: www.spirent.com

Conclusion

All indications appear to point to increased importance of encrypted transport layer for interconnectivity of networks, internet and smart devices in the coming years. The case for moving to TLS v1.3 is very compelling due to increased efficiency and security that it brings compared to its predecessors, however it is a new and unproven technology. Malicious attackers have exploited SSL/TLS vulnerabilities to disrupt services and to their gains in recent years and this trend will most likely continue.

CyberFlood advanced fuzzing provides unique capabilities in TLS v1.3 negative testing that can help in gaining insight and improving robustness of the services proactively. Administrators who deploy TLS v1.3 can use this insight to seek patches or alternatives if faults are found in sensitive areas. As illustrated in the above section, developers and providers of open source as well as commercial TLS v1.3 services can leverage automated CyberFlood advanced fuzzing report to improve robustness of their offering by rectifying issues found in the implementation which would otherwise be impractical to uncover by manual code inspections or manual testing.

Additional Spirent Resources

- Spirent CyberFlood Advanced Fuzzing datasheet: https://www.spirent.com/-/media/Datasheets/Security/CyberFlood_Advanced-Fuzzing_datasheet.pdf?la=en
- Spirent CyberFlood Fuzz Testing On-demand Webinar: https://www.spirent.com/Assets/Video/Webinar_Fuzz-Testing
- Blog URL: https://www.spirent.com/Blogs/Security/2017/December/TLS v1-3-deployments

spirent™ Promise. Assured.

Contact Us

For more information, call your Spirent sales representative or visit us on the web at www.spirent.com/ContactSpirent.

www.spirent.com

Americas 1-800-SPIRENT
+1-800-774-7368 | sales@spirent.com

US Government & Defense
info@spirentfederal.com | spirentfederal.com

Europe and the Middle East
+44 (0) 1293 767979 | emeainfo@spirent.com

Asia and the Pacific
+86-10-8518-2539 | salesasia@spirent.com

Rev B | 08/18