Journal of Computer and Communications, 2021, 9, 174-190
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Inmap-t: Leveraging TTCN-3 to Test the Security Impact of Intra Network Elements

Antonino Vitale[1], Marc Dacier[2*]

[1]Digital Security Department, EURECOM, Biot, France
[2]KAUST—Resilient Computing and Cybersecurity Center (RC3), King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia
Email: antonino.vitale@eurecom.fr, marc.dacier@kaust.edu.sa

How to cite this paper: Vitale, A. and Dacier, M. (2021) *Inmap-t*: Leveraging TTCN-3 to Test the Security Impact of Intra Network Elements. *Journal of Computer and Communications*, **9**, 174-190.
https://doi.org/10.4236/jcc.2021.96010

Received: May 24, 2021
Accepted: June 26, 2021
Published: June 29, 2021

Abstract

This paper rejuvenates the notion of conformance testing in order to assess the security of networks. It leverages the Testing and Test Control Notation Version 3 (TTCN-3) by applying it to a redefined notion of *System under Test* (*SUT*). Instead of testing, as it is classically done, a software/firmware/hardware element, an intangible object, namely the network, is tested in order to infer some of its security properties. After a brief introduction of TTCN-3 and Titan, its compilation and execution environment, a couple of use cases are provided to illustrate the feasibility of the approach. The pros and cons of using TTCN-3 to implement a scalable and flexible network testing environment are discussed.

Keywords

TTCN-3, Network Security, Conformance Testing, Deep Packet Inspection, Firewall

1. Introduction

This paper explores the usability of the Testing and Test Control Notation Version 3 (TTCN-3 [1]) to infer the presence of security related devices in a network. TTCN-3 is a standard maintained by ETSI that offers a modular testing language and an independent execution environment. Both are used and supported by a large community, very active in various industries. It is mostly used to assess the quality and conformance of a given implementation of a client or server for a specific protocol. Instead of testing a specific piece of software, the idea is to test the properties of an intangible object, namely the network connec-

*Work partly performed whilst this author was with EURECOM.

tion taking place between two communicating parties. More precisely, the proposal is made to use TTCN-3 to detect the existence of communication artefacts generated by security devices, such as an Intrusion Prevention System (IPS) [2] or a firewall [3], to name only two. The novel contribution is thus to redefine the classical notion of *System Under Test* (*SUT*) and to leverage TTCN-3 in a new way to assess some network security properties.

In order to present these ideas, this article is structured as follows. Section 2 motivates this work by explaining the role of a number of devices that could exist on a path between two communicating parties and the security implications of their existences. Section 3 reviews the state of the art related to TTCN-3 on one hand and, on the other hand, to the various approaches related to the testing of network security properties. Section 4 (resp. 5), in order for this document to be self-contained, provides a brief introduction to TTCN-3 (resp. to Titan, its compilation and execution environment). Section 6 shows, thanks to three distinct test cases how TTCN-3 can be used to detect the presence of the devices introduced earlier. Section 7, discusses the pros and cons of this approach based on the lessons learned and Section 8 concludes the paper.

2. Motivation

"*The Network is the Computer*" was Sun Microsystems' vision in the 80's as defined, in 1984, by John Gage, the 21st employee of that company. The fact that the Cloudflare company has recently registered the trademark for that very same phrase [4] is emblematic of the regained importance of the network in everyone's daily lives. There is not a single meaningful application which, at some point, does not need to connect to another machine, somewhere on the Internet. It is thus of the uttermost importance to ensure the quality, availability, reliability, performability and security of network connections between machines.

There is a large community of experts that have been very active and successful in monitoring, assessing and improving the four first of these properties. The achievements of the network measurement community are to numerous to be mentioned here.

On the security front, things are very different and, somehow, for good reasons. The security aspects of the network itself have barely been considered because, following the seminal Dolev-Yao model [5], the presence of so called "active observers" is always assumed within the network. Security protocols have to be designed to resist against them by ensuring *end-to-end* security properties. It is the sole responsibility of the two edge devices to do what is necessary to ensure a secure communication. That is a sound and reasonable design assumption but, in practice, very few real world protocols resist to a powerful "*active observer*" as it requires strong, unbreakable, mutual authentication of both end devices which is difficult to achieve in practice.

Furthermore, one observes a growing number of devices that interfere, at various layers, with some aspects of these connections. In the same way that traffic shaping devices [6] can, positively or negatively, influence the performances of a

network connection, several devices exist that can alter its security. Specifically, devices such as firewalls [3] [7], web proxies [8], deep packet inspection tools [9], intrusion prevention systems [2], to only name the most well known ones, are of concern here. All these devices, and many others, can, and usually do, break the *end-to-end* assumption made about a client-server communication, even encrypted ones sometimes. Their deployment is usually done to improve network security but they are double-edged swords since they can also be deployed surreptitiously by an attacker for malicious purposes anywhere in the Internet to observe or modify data in transit.

Last but not least, many protocols are vulnerable to man in the middle attacks at various layers of the protocol stack, as clearly laid out in the 2016 survey by Conti *et al.* [10] (ARP, DHCP, IP, TCP, DNS, HTTP, etc.).

Thus, in practice, the "*active observers*" of the Dolev-Yao model can exist under many forms but protocols that are vulnerable to them are still largely used without being even able to test for the presence of such adversary!

Clearly, what is lacking is a generic framework capable of testing the security properties of a network connection between two end points. On one hand, such a framework could enable the users to verify that known security mechanisms are present and properly configured (e.g. the firewall is up and running and implements NATing properly). On the other hand, it could also detect the presence of unknown entities interfering between two endpoints (e.g. the HTTP traffic received is different from the one sent by the server).

Proposing such a framework is the intent of this work. More specifically, it shows how to leverage the TTCN-3 environment to run test campaigns to detect either that known devices are configured properly or to detect the presence of malicious ones.

3. Review of the State of the Art

The origins of TTCN-3 are almost 40 years old, starting with the ISO/IEC standard 9646: OSI Conformance Testing Methodology and Framework (CTMF) [11]. The interested reader is referred to the historical review by Grawoski *et al.* [12] to discover the various iterations that have led to today's version of TTCN-3 [1]. TTCN-3 is presented in Section 4 but, for the sake of completeness, it is worth mentioning that some competitors exist to this framework such as [13] [14] [15] but reviewing them falls outside the scope of this paper.

TTCN-3 conformance test suites exist for a number of important protocols (eg. SIP [16], RIP [17] or WIMAX [18]). Many papers have been published on how to use TTCN-3, in particular during the first decade of the 21st century, mostly in networking venues, to introduce the environment and explain how it could be used to test correct implementations of protocols and distributed systems [19] [20] [21] [22]. It has also been considered by some authors outside the scope of conformance testing. For instance, Brezinski did propose to derive intrusion detection rules from the TTCN-3 conformance tests [23]. Others have shown how to use it as a fuzzing tool for various protocols [24] [25] or as a pe-

netration testing tool against web services [26]. To the best knowledge of the authors, no one has proposed to use TTCN-3 to test the network itself as we propose in this work.

On the security front, several initiatives have existed over the years to assess Internet security. The most famous ones are the various darknets, honeypots and honeynets that have been deployed over the years to assess the threats landscape, thanks to passive measurements [27] [28] [29] [30] [31]. A couple of other initiatives, have used active measurements techniques such as Shodan [32] and Censys [33]. All these systems focus on the devices on the edge of the network and ignore those within it.

A notable exception exists with the body of knowledge accumulated to detect proxies on the Internet [34] [35] [36]. These works have highlighted the existence of the "active observers" by using techniques specifically designed for this specific threat.

To conclude, it is important to mention the very interesting *Netalyzer* platform. Even if it initially aimed at performing end systems measurements [37] [38], in a second stage, that platform has shown its ability to detect specific elements within the network [35] [39]. Unfortunately, the platform has been shutdown in March 2019, preventing the authors from investigating the possibility to leverage it for their goal.

4. TTCN-3: Main Concepts

4.1. Introduction to TTCN-3

Simply speaking, TTCN-3 is a standardized framework defined to verify the conformance of a given implementation with respect to the specification of a communication protocol. In TTCN-3 parlance, a **System Under Test (SUT)** is the software to be tested. The test itself consists in exchanging messages with the SUT and in verifying that the replies are consistent with the specification. TTCN-3 provides all the communication, testing mechanisms and tools to carry out such test campaigns. TTCN-3 defines so called interfaces for the various components to communicate together, including with the SUT. Test campaigns are made of test cases written using the TTCN-3 language. A TTCN-3 program is referred to as an Abstract Test Suite (ATS). It expresses the configuration and behavior of an abstract test system, which is composed of a set of concurrently executing test components. A test case evaluates a single property of the protocol by means of an exchange of messages. Each test case produces a so called **verdict**. Writing the test cases is the responsibility of the test designer but TTCN-3 takes care of all the other operations such as the initial configuration, communication between modules, collection of the verdicts, etc. On top of that, TTCN-3 offers a number of functionalities to the test designer to easily handle issues such as concurrency, time management, event handling, etc.

TTCN-3 offers several modes, respectively called Single, Parallel and Distributed modes.

- In **Single** mode, there is only one component participating in the execution of the test campaign.
- In **Parallel** mode, several components run in an asynchronous way and communicate together.
- The **Distributed** mode corresponds to the *parallel* mode when the participating components do not run on the same machine.

4.2. TTCN-3 Core Language

The standardized testing language defined within TTCN-3 [1] has the look and feel of a regular programming language. It provides a well-defined syntax for the definition of tests independent of any application domain. A few important terms are defined here below. They will be used throughout this document, namely:

- A **component** is a user specified entity, which contains user-defined **ports**, via which the component can interact with other components and the SUT. Among the various components involved in a test campaign, some have a specific role, namely:
- The **Main Test Component (MTC)** is unique and mandatory for each test campaign. It is created at the beginning of the test campaign and when its execution is over, the test campaign also ends. The MTC coordinates the creation of the needed other components, possibly on remote devices.
- A **Parallel Test Component (PTC)** is created, as per the MTC instruction, to help it in carrying out the test cases.
- The **Test System Interface (TSI)** component (also referred to as the system component) is an abstract interface to the SUT. It is used by the other components in order to obtain some information from the SUT (e.g., its IP address, the content of a log file, etc.).
- **Port**: each component owns at least one port that is an interface towards the outer world. Two components can communicate with each other if they own the same type of port and if their ports are connected. The port definition includes the protocol supported to exchange data in a bidirectional flow. There are "Ethernet" ports but also "IP", "TCP", "HTTP" ones and several others.
- A **module** corresponds to a compilation unit in traditional programming languages. It is divided into a *definition* and a *control* part. The former contains top-level definitions, such as type, data, constant, port, component, function and test cases. The latter defines the execution sequence of the test cases defined in the first part.
- **Test case**: one module consists of at least one test case which is launched in its control part. A sequence of several test cases defines a **test campaign**.
- A **Verdict** is the output of a *test case*. It can only be one of the five built-in values, namely: pass, fail, inconc (stands for inconclusive), none (used as initial value) and error.

Each component generates a log file throughout the execution of the test campaign. A local verdict for each test case is saved into it. At the end of the test

campaign, all log files are combined to generate the global verdict file. It contains a global verdict for each test case. The global verdict of a given test case is defined as the value of the local verdict, for that same case, with the highest priority among all participating components (including the MTC). The priority order for the verdicts is defined as follows: none, pass, inconc, fail, error. Error is the highest priority and none the lowest.

TTCN-3 does not come with any compiler for the language. A compilation environment, such as Titan [40] is needed to use TTCN-3 in practice. Titan also provides an execution environment that will, e.g., provide mechanisms under the hood for the MTC to launch PTCs remotely whenever needed. For this paper to be self-contained, Titan is briefly introduced in the next Section.

5. Titan: Compilation and Execution Environment for TTCN-3

5.1. Introduction

Eclipse Titan is a TTCN-3 compilation and execution environment with an Eclipse-based IDE. The user of the tool can develop test cases, test execution logic and build the executable test suite for several platforms. Titan consists of a core part, executing in a Unix/Linux-like environment and a set of Eclipse plug-ins. The latest version of the project is 7.1.0 and dates back from July 2020. There are submissions as recent as May 2021 on their git website [41], indicating that this is an active project, supported by, among others, Ericsson, the Swedish company, one of the world leaders in communication technology.

It is worth noting that 1) not everything defined in the TTCN-3 standard is supported by Titan and that 2) Titan introduces a couple of novel concepts to support the execution environment.

5.2. Titan Compilation Environment

Titan includes a **TTCN-3 and ASN.1 Compiler** that transforms the TTCN-3 Core Language modules into C++ code. The intermediary C++ files are then compiled into executable files. During the compilation phase other C++ files will be included, such as:

- The **Base Library** which provides other C++ modules required for the correct compilation of the TTCN-3 Core Language structures and data types;
- The Titan implementation of the **Test Ports**. They are defined in C++ files. Titan supports a number of network protocols and provides also the so-called protocol modules that are APIs to encode/decode streams of bytes to be sent into test ports. For instance, when using a TCP test port, the HTTP protocol module simplifies the programmer's task if in need of sending/receiving HTTP messages.

The compilation environment can be installed from the Titan git repository [41]. The TTCN-3 modules compiled in this environment generate an executable file called the **executable test suite**. This file contains the test cases but also

the code required to facilitate the coordination of the execution of the various components. These other functions constitute the Titan execution environment.

5.3. Titan Execution Environment

To free the test designer from all the issues related to the communication, synchronisation and coordination aspects of a distributed system architecture, Titan introduces the notions of the **Main Controller** as well as of a **Host Controller**. These elements support the transparent execution environment to run the tests in the TTCN-3 *distributed mode*. They are not part of the TTCN-3 standard.

The **Main Controller (MC)** is a stand-alone program. It must be the first program to be launched in order to run a test campaign. Once running, it listens to a TCP port for incoming connections from the hosts that will participate to the test.

A host tries to establish a connection with the MC as soon as the *executable test suite* is launched on that host. This instantiates a **Host Controller (HC)** which dies if the connection with the MC fails. Similarly, the HC dies if the connection with the MC is interrupted.

It is clear that the MC must thus first be launched and, then, the various participating HCs. This is a manual process not supported by the Titan execution environment.

The MC keeps track of the connected HCs and determines when to start the test campaign itself. To do so, the MC chooses one of the connected HCs and instructs it to create an MTC. The HC accomplishes this by forking its own process. The child process becomes the MTC. Then, the MC communicates directly with the MTC to inform it about the test case(s) to be executed. Whenever a PTC is required for a given test case, the MC instructs the concerned HC to create one. As for the MTC, the HC forks its own process and the child becomes the needed PTC.

As can be seen, the *executable test suite* contains the code for the HC, MTC and PTC as well as all possible test case(s). Which one(s) to run within a given test campaign can be defined by means of a **Run-Time Configuration File** which contains a number of parameters such as which test case(s) to run, between which hosts, etc.

5.4. Titan Ports

As eluded to before, Titan supports a number of different protocols by means of so called *ports* which are the interfaces used by components to communicate together. In the context of this work, three of them have been used, namely the **LANL2**, the **IPL4** and the **PIPE** ports. Their characteristics are briefly given here below to ease the presentation of the test cases in the next Section.

- The **LANL2** port [42] is a layer 2, *i.e.* link layer, port. It enables the test designer to send and receive Ethernet II frames. The capture filtering is done by Libpcap. The headers and content of the upper layers (networking, transport

and application) must be provided as a sequence of bytes.

- The **IPL4** port [43] is a layer 4, *i.e.* transport layer, port. It is a general purpose, session independent port providing access to several Internet (IPV4 and IPV6) transport layers protocols such as TCP, UDP, SSL/TLS, etc. If the chosen protocol is connection-oriented then the test port will take care of the handshake messages.
- The **PIPE** port [44] is a special port that establishes a connection between the TTCN-3 test executor and the Unix/Linux shell. It offers the test designer a simple way to execute commands on a machine and retrieve the information available on stdin, stdout, and stderr.

Whenever a LANL2 test port is used, the component using that port must know the name of the sending interface and the MAC address of the recipient (final destination or next hop gateway). These two information are not necessarily known before run time but they can be obtained from the host thanks to the PIPE port.

Some caution must be taken to initiate a TCP connection thanks to the LANL2 port. Sending the initial SYN packet is easy. The remote device, assuming it is listening on the targeted port, will then generate a SYN/ACK packet, as per the usual TCP three ways handshake. The initiating device will not know what to do with this reply since the LANL2 port has bypassed TCP/IP when sending the initial frame. No port is listening for the SYN/ACK packet. As a result, the kernel is very likely to generate a RST packet as a response, which will kill the TCP connection that is being established. To prevent this from happening, a specific iptables [45] rule must be inserted during the whole test case to prevent the emission of the RST packet.

6. Testing the Impact of Intra Network Elements

6.1. Goal

Typically, when using the TTCN-3 framework, the test designer considers the SUT to be a specific hardware, firmware or software whose implementation compliance needs to be assessed with respect to a specific protocol specification.

In this work, TTCN-3 and the SUT are looked at from a different view point. The SUT is an intangible object; it is the connection established between a client and a server. The goal is to test whether its behavior is consistent with what is expected from an *end-to-end* connection. In other words, the goal is to test whether any element between the two end points is interfering with this connection and, if yes, how.

Today's networks contain a number of elements that routinely interfer with the *end-to-end* connections, such as:

- A firewall that implements Network Address Translation (NAT) modifies the IP addresses of packets traversing it;
- A traffic shaper might buffer packets before sending them at different rates, sometimes concatenating their payloads into bigger packets;

- An ssh jump host could redirect an incoming connection to a different IP and, or, destination port;
- A caching proxy could provide clients with responses to their web requests without talking to the queried server;
- etc.

In the general case, interfering with the expected end-to-end property of a connection is achieved with the best interest of the client in mind, e.g. to improve security, performance, latency, etc. However such practices are double edged swords and can also be used by malicious actors, anywhere in between the two end points in the network, to achieve some nefarious goal. Being able to test, and explain, such interferences are thus important for two reasons:

1) To verify that all the *good* elements are acting as they should.

2) To verify that no other element than the *good* ones is interfering with a connection.

The goal is thus to leverage the TTCN-3 environment to exchange especially crafted packets between a client and a server in order to determine whether or not any element is trying to attack the confidentiality, integrity or availability of the data in transit.

To achieve this goal, the *INMAP-T*[1] system has been built, named for **I**ntra **N**etwork **Map**ping with **T**tcn3. *INMAP-T* is made of a number of TTCN-3 test cases. The space allocated here does not enable a full description of all of them. Instead of an exhaustive presentation, three representative test cases have been chosen to exemplify the usage of TTCN-3.

The generic set up is first introduced, followed by a brief presentation of how to carry out tests thanks to TTCN-3. The discussion is limited to the atomic notion of a test case. Explaining how to combine them within a test campaign to precisely identify a specific interfering network element lies outside the scope of this paper.

6.2. Setup

The architecture used is very simple, made of three hosts: one for the MC and the MTC, one for the Client and one for the Server. The MC and MTC reside on the same host for the sake of simplicity. Both of them play a coordination role during the execution of the tests. The MC maintains a TCP connection not only with the HCs on the client and server machines but also with all the instantiated components (MTC and PTCs). These connections are used whenever the MTC commands the creation of a PTC: the MTC sends the request to the MC which forwards it to the target HC.

6.3. Test Cases

The following three test cases highlight how powerful TTCN-3 is to carry out the kind of tests of interest within this work.

1) The first test case uses *fragmented packets* to detect the presence of any

[1]This name is an indirect reference to the *nmap* tool [46] that focuses on the edges of the network.

element that inspects the payload before letting the packets go through.

2) The second compares *sequence numbers* and leverages the MTC component to detect a man in the middle at the transport layer.

3) The third one takes advantage of an *exporter* to verify that encrypted channels are not eavesdropped.

6.3.1. Fragmented Packets

Packets in transit between two end points should only be handled at the physical, link and network layers. They should never be *pushed* to the transport and application layers. However, in some cases, an intermediary node must inspect the content of the payload before letting it pass. This is true, for instance with Network Based Intrusion Prevention systems [2] when carrying out Deep Packet Inspection. In such case, the packet goes up to the application layer to be scrutinized before being *pushed* back to the next hop. This is also true for firewalls who have rules that require the inspection of some TCP header values or of the application payload. If an IP datagram is split into two fragments, the IP layer needs to reassemble them in order to extract the layer 4 payload and to pass it over to the TCP layer which, then, can pass it to the application layer. Depending on the implementation, that payload, when resent, lies within a single reassembled IP datagram or is split into the two initial fragmented ones.

A test case has been built to detect such possible reassembly, in transit, of fragmented packets. The LANL2 test port has been used to send the packets because it enables to modify the datagram header and makes it possible to craft fragmented packets. Similarly, the server uses the LANL2 test port and has access to the raw packets sent. If the server receives two fragmented packets, its local verdict for the test is "Pass" and "Fail" otherwise. The local verdict of the client as well as of the MTC is always set to "Pass" if no error happens. Therefore, the global verdict will be uniquely defined by the server since "Fail" has a higher priority than "Pass", as explained in Section 4.2 on page 5.

A number of experiments have been run and have indeed confirmed that such a test would detect the presence of a number of elements whose existence, otherwise, would remain unknown.

The listing 1 gives a flavor of the TTCN-3 language expressiveness by providing some excerpts of the code required to create a fragmented packet. Listing 2 shows how the server can read a frame from the wire and in Listing 3 how it can check whether the received packet is a fragmented one or not.

```
raw_ip_1 := \\
      createIPpacket(payload:=raw_tcp_1,mfrag:='1'B,dest_addr:=dest_ip);
raw_ip_2 :=  \\
      createIPpacket(payload:=raw_tcp_2,foffset:=first_len/8,dest_addr:=dest_ip);
```

Listing 1: Creation of fragments by the Client

```
[] server_lan_p.receive(ASP_v2_LANL2:?) -> value eth_frame{ /*...*/ }
```

Listing 2: Reception of a frame by the Server

```
if(ip_header.mfrag == str2bit("1")){
                log("receiveFragPackets: received first fragment");
}
else if (ip_header.foffset != 0) {
                log("receiveFragPackets: received second fragment");
}
```

Listing 3: Detection of first or second fragment

6.3.2. Sequences Numbers

Sometimes, devices actively violate the *end-to-end* assumption by acting as a man in the middle at the transport layer. This is the case, for instance, for several kinds of proxies. They will pretend to be the server end point for the client and the client for the server. Two distinct TCP connections exist and the proxy pushes data from one to the other. It is worth noting that, to remain invisible, the proxy can spoof the other IP addresses so that its own will never appear in any of these two TCP connections.

One way to detect the existence of such man in the middle is to compare the initial sequence number sent by the client and the one received by the server. Only a very meticulous attacker would pick the very same initial sequence number for the TCP connection he initiates with the server.

Such a test case has been built by taking advantage of the IPL4 test port this time. This enables to let the test port takes care of the three way handsakes for us. The LANL2 test ports are also needed though to extract the sequence numbers.

To decide whether the test has failed or not, it is a must to have access to information that only the client, on one hand, and the server, on the other hand, possess. Unfortunately, the only information they are normally supposed to send back to the MTC is their local verdict! This is where the Titan environment proves to be very resourceful. Indeed, Titan offers some language extensions[2] and one of them enables the programmer to let a PTCs *return values*[3] to the MTC at the end of a test case, together with their local verdict. This feature is used to let the client (resp. server) send a "Pass" local verdict to the MTC, together with the initial sequence number sent (resp. received). The MTC compares the two values to produce its own local verdict which will then determine the final global verdict. Here to, a number of experiments have been run that have confirmed that most implementations of invisible proxies or man in the middle attacks did not bother replicating the initial sequence number among both TCP connections. They can thus be detected by this approach.

6.3.3. Exporter

In the last example, the focus is on encrypted channels, such as the ones in a SSL/TLS session. The situation considered is one where a very stealthy attacker would defeat the previous detection and would decrypt the data in transit and, then, reencrypt it. In theory, this could remain invisible to all. To decrypt the

[2]TTCN-3 Core Language offers a way to extend its functionalities and features thanks to the *extension* [1] clause.

[3]Loosely speaking. A detailed explanation falls outside the scope of this paper.

payload, the attacker needs to know the session key. This is possible if he manages to impersonate the server. However, by design of the key establishment protocols, there will necessarily be two distinct session keys for the connection from the client to the attacker and from the attacker to the server.

A test case has been built to assess whether or not the encrypted sessions seen by the client and by the server rely on the same secrets. However, this information, for good reasons, is not available to external applications but resides securely within the TLS process memory space. Nevertheless, it is possible to use a mechanism known as an *exporter*, defined in the RFC5705 [47] (previously referred to as a TLS extractor). Thanks to this, it is possible, after a successful TLS handshake, to extract an hexadecimal value derived from the master secret of the session. In the absence of a man in the middle attack, the client and the server must derive the same value.

Listing 4 indicates how to obtain such exporter value when using the IPL4 test port in TTCN-3. To decide whether the test fails or not, the same technique as in the previous case is used, by letting the MTC take care of the comparisons of the two exporter values.

```
var octetstring my_key := \\
f_IPL4_exportTlsKey(client_tcp_p,connId,tls_label,tls_context,1024);
```

Listing 4: Using the TLS exporter

7. Discussion and Lessons Learned

The vision is to have a number of end points (servers) disseminated around the world so that clients could, on one hand, test their own equipment when communicating to the Internet and, on the other hand, detect the presence of malicious elements on certain routes, possibly sharing that information with a larger community.

A number of test cases have been implemented, on top of the three described here above. The examples given aim at letting the reader understand how easily it could be to test, for instance:

- whether a firewall is correctly configured: which ports are blocked, which ones are forwarded, whether NATing is enabled, etc.
- whether a DPI is up and running by, e.g., sending a payload that matches a detection signature which should block the packet.
- whether some unknown device is interfering at any possible layer with a given connection.
- whether some man in the middle attack is taking place.
- whether some DNS poisoning attack has been realized.
- etc.

Based on the experiments, rejuvenating the concept of SUT by defining it as being an intangible object, as opposed to, e.g., a server implementation, appears to offer a lot of opportunities. To do this, TTCN-3 and Titan offer all the func-

tional blocks needed, not only to define the test campaigns but also to orchestrate the whole distributed architecture required. However, when running the experiments, a number of shortcomings appear that need to be addressed in order to build an easy to use and scalable testing environment. The most problematic issues are summarized here after:

- As explained before, the executable test suite must contain, among other things, the code for all test cases. Adding a single new test case amounts to change the whole executable, install it on all participating hosts and restarting all components. Not only is this a very costly operation but, worse, neither TTCN-3 nor Titan do offer a mechanism to implement such software update.
- The more test cases exist, the bigger the executable test suite becomes. In particular, the more Titan ports are used, the more code needs to be included. As of now, the size of the executable is, roughly, of 17 MBytes. 2/5 of it comes from the three ports used, 2/5 of it is the code for the MTC, PTC, etc., the last 1/5 corresponds to code written for the test cases.
- TTCN-3 is a very rich and well defined language but its readability is influenced by its historic ties with the C++ language. This will please those used to it but may be a show stopper for newcomers.
- The TTCN-3 standard defines a single way to establish the global verdict, based on priorities among local verdicts, as explained in the text. It is possible, as shown in the text, to come around this limitation by implementing a decision function within the MTC based on values provided by the other elements and by forcing them all to generate a "Pass" verdict. This is a practical solution but definitely not an elegant one.

All these issues could possibly be addressed by "*wrapping*" the TTCN-3 environment within another one providing the missing functionalities. A first attempt at doing this has been achieved by one of the authors in [48]. Further work needs to be done in order to decide whether this is the right way to go or if, instead, it is preferable to recreate a new environment, from scratch, specifically tailored at network testing, inspired by the many good ideas and concepts from TTCN-3.

8. Conclusion

In this work, the rationales for testing the security properties of an *end-to-end* connection between a client and a server have been presented. The standardized TTCN-3 environment, while not designed for such tasks, could be leveraged by redefining the notion of SUT, as shown in the text. A couple of exemplary use cases demonstrate the feasibility and the expressive power of the proposed approach. The pros and cons are discussed, of using TTCN-3 vs. building a new framework that would include the missing functionalities while benefiting from the well thought of architecture and concepts implemented over the years in TTCN-3 and Titan.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] ETSI (2016) Methods for Testing and Specification (MTS): The Testing and Test Control Notation Version 3; Part 1: Ttcn-3 Core Language. Vol. ETSI ES 201 873-1 V4.8.1, ETSI, Sophia Antipolis.

[2] Stiawan, D., Abdullah, A.H. and Idris, M.Y. (2010) The Trends of Intrusion Prevention System Network. 2010 2*nd International Conference on Education Technology and Computer*, Shanghai, 22-24 June 2010, V4-217-V4-221. https://doi.org/10.1109/ICETC.2010.5529697

[3] Cheswick, W.R., Bellovin, S.M. and Rubin, A.D. (2003) Firewalls and Internet Security: Repelling the Wily Hacker. Addison-Wesley Professional, Boston.

[4] Vallurupalli, D. (2019) Cloudflare Registers Trademark for the Network Is the Computer. Technical Report, Press Release, Cloud-Flare, San Francisco.

[5] Dolev, D. and Yao, A. (1983) On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, **29**, 198-208. https://doi.org/10.1109/TIT.1983.1056650

[6] Armitage, G. (2000) Quality of Service in IP Networks. SAMS Publishing, Indianapolis.

[7] Ioannidism S., Keromytis, A.D., Bellovin, S.M. and Smith, J.M. (2000) Implementing a Distributed Firewall. *Proceedings of the* 7th *ACM Conference on Computer and Communications Security*, Athens, November 2000, 190-199. https://doi.org/10.1145/352600.353052

[8] Fielding, R. and Reschke, J. (2014) Hypertext Transfer Protocol (http/1.1): Message Syntax and Routing. RFC 7230, Internet Requests for Comments, RFC Editor. http://www.rfc-editor.org/rfc/rfc7230.txt https://doi.org/10.17487/rfc7230

[9] Debar, H., Dacier, M. and Wespi, A. (2000) A Revised Taxonomy for Intrusion-Detection Systems. *Annales des telecommunications*, **55**, 361-378.

[10] Conti, M., Dragoni, N. and Lesyk, V. (2016) A Survey of Man in the Middle Attacks. *IEEE Communications Surveys Tutorials*, **18**, 2027-2051. https://doi.org/10.1109/COMST.2016.2548426

[11] ISO/IEC 9646 (1991) Information Technology; Conformance Testing Methodology and Framework (Parts 1-7). International Standard IS9646. International Organization for Standardization, Geneva.

[12] Grabowski, J., Schieferdecker, I. and Ulrich, A. (2014) History, Status, and Recent Trends of the Testing and Test Control Notation Version 3 (TTCN-3). *International Journal on Software Tools for Technology Transfer*, **16**, 215-225. https://doi.org/10.1007/s10009-014-0302-9

[13] Wu, J., Yang, L. and Luo, X. (2008) Jata: A Language for Distributed Component Testing. 2008 15*th Asia-Pacific Software Engineering Conference*, Beijing, 3-5 December 2008, 145-152. https://doi.org/10.1109/APSEC.2008.27

[14] Gorringe, C., Seavey, M. and Lopes, T. (2011) An Overview of the ATML Family and Related Standards. 2011 *IEEE AUTOTESTCON*, Baltimore, 12-15 September 2011, 117-123. https://doi.org/10.1109/AUTEST.2011.6058783

[15] Baker, P., Dai, Z.R., Grabowski, J., Schieferdecker, I. and Williams, C. (2007) Model-Driven Testing: Using the UML Testing Profile. Springer-Verlag, Berlin, Heidelberg. http://doi.org/10.1007/978-3-540-72563-3

[16] ETSI (2006) Conformance Test Specification for SIP (IETF RFC 3261). ETSI, TS 102 027. ETSI, Sophia Antipolis.

[17] Floch, A., Roudaut, F., Sabiguero, A. and Viho, C. (2005) Some Lessons from an Experiment Using ttcn-3 for the RIPng Testing. *IFIP International Conference on Testing of Communicating Systems*, Montreal, 31 May-2 June 2005, 318-332. https://doi.org/10.1007/11430230_22

[18] ETSI (2009) Conformance Testing for the Network layer of HiperMAN/WiMAX terminal devices. ETSI TS 102 624, ETSI, Sophia Antipolis.

[19] Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A. and Willcock, C. (2003) An Introduction to the Testing and Test Control Notation (TTCN-3). *Computer Networks*, **42**, 375-403. https://doi.org/10.1016/S1389-1286(03)00249-4

[20] Yin, X., Wang, Z., Jing, C. and Shi, X. (2008) A TTCN-3-Based Protocol Testing System and Its Extension. *Science in China Series F: Information Sciences*, **51**, 1703-1722. https://doi.org/10.1007/s11432-008-0156-4

[21] Schieferdecker, I. (2010) Test Automation with ttcn-3-State of the Art and a Future Perspective. *IFIP International Conference on Testing Software and Systems*, Natal, 8-10 November 2010, 1-14. https://doi.org/10.1007/978-3-642-16573-3_1

[22] Lahami, M., Fakhfakh, F., Krichen, M. and Jmaiel, M. (2012) Towards a ttcn-3 Test System for Runtime Testing of Adaptable and Distributed Systems. *IFIP International Conference on Testing Software and Systems*, Aalborg, 19-21 November 2012, 71-86. https://doi.org/10.1007/978-3-642-34691-0_7

[23] Brzezinski, K.M. (2007) Intrusion Detection as Passive Testing: Linguistic Support with ttcn-3. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Lucerne, 12-13 July 2007, 79-88. https://doi.org/10.1007/978-3-540-73614-1_5

[24] Jing, C., Wang, Z., Shi, X., Yin, X. and Wu, J. (2008) Mutation Testing of Protocol Messages Based on Extended TTCN-3. 22*nd International Conference on Advanced Information Networking and Applications* (*AINA* 2008), Ginowan, 25-28 March 2008, 667-674. https://doi.org/10.1109/AINA.2008.98

[25] Xu, L., Wu, J. and Liu, C. (2009) T3fah: A TTCN-3 Based Fuzzer with Attack Heuristics. 2009 *WRI World Congress on Computer Science and Information Engineering*, Vol. 7, Los Angeles, 31 March-2 April 2009, 744-749. https://doi.org/10.1109/csie.2009.706

[26] Stepien, B., Peyton, L. and Xiong, P. (2012) Using TTCN-3 as a Modeling Language for Web Penetration Testing. 2012 *IEEE International Conference on Industrial Technology*, Athens, 19-21 March 2012, 674-681. https://doi.org/10.1109/ICIT.2012.6210016

[27] Zou, C.C., Gao, L., Gong, W. and Towsley, D. (2003) Monitoring and Early Warning for Internet Worms. *Proceedings of the* 10*th ACM Conference on Computer and Communications Security*, Washington DC, October 2003, 190-199. https://doi.org/10.1145/948109.948136

[28] Bailey, M., Cooke, E., Jahanian, F., Nazario, J., Watson, D., *et al.* (2005) The Internet Motion Sensor: A Distributed Blackhole Monitoring System. *Proceedings of* 12*th ISOC Symposium on Network and Distributed Systems Security*, San Diego, Febuary 2005, 167-179.

[29] Alata, E., Dacier, M., Deswarte, Y., Kaaâniche, M., Kortchinsky, K., Nicomette, V.,

Pham, V.-H. and Pouget, F. (2006) Collection and Analysis of Attack Data Based on Honeypots Deployed on the Internet. In: Gollmann, D., Massacci, F. and Yautsiukhin, A., Eds., *Quality of Protection*, Springer, Boston, 79-91. https://doi.org/10.1007/978-0-387-36584-8_7

[30] Dacier, M., Pham, V.-H. and Thonnard, O. (2009) The WOMBAT Attack Attribution Method: Some Results. 2009 *International Conference on Information Systems Security*, Kolkata, 14-18 December 2009, 19-37.

[31] Pour, M.S., Bou-Harb, E., Varma, K., Neshenko, N., Pados, D.A. and Choo, K.-K.R. (2019) Comprehending the IoT Cyber Threat Landscape: A Data Dimensionality Reduction Technique to Infer and Characterize Internet-Scale IoT Probing Campaigns. *Digital Investigation*, **28**, S40-S49. https://doi.org/10.1016/j.diin.2019.01.014

[32] Matherly, J. (2015) Complete Guide to Shodan. Vol. 1, Shodan, LLC (2016-02-25).

[33] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M. and Halderman, J.A. (2015) A Search Engine Backed by Internet-Wide Scanning. *Proceedings of the* 22nd ACM *SIGSAC Conference on Computer and Communications Security*, Denver, October 2015, 542-553. https://doi.org/10.1145/2810103.2813703

[34] Reis, C., Gribble, S.D., Kohno, T. and Weaver, N.C. (2008) Detecting in-Flight Page Changes with Web Tripwires. 5*th USENIX Symposium on Networked Systems Design & Implementation*, Vol. 8, San Francisco, 16-18 April 2008, 31-44.

[35] Weaver, N., Kreibich, C., Dam, M. and Paxson, V. (2014) Here Be Web Proxies. *International Conference on Passive and Active Network Measurement*, Los Angeles, 10-11 March 2014, 183-192. https://doi.org/10.1007/978-3-319-04918-2_18

[36] Miller, S., Curran, K. and Lunney, T. (2015) Traffic Classification for the Detection of Anonymous Web Proxy Routing. *International Journal for Information Security Research*, **5**, 538-545. https://doi.org/10.20533/ijisr.2042.4639.2015.0061

[37] Kreibich, C., Weaver, N., Nechaev, B. and Paxson, V. (2010) Netalyzr: Illuminating the Edge Network. *Proceedings of the* 10*th ACM SIGCOMM Conference on Internet Measurement*, Melbourne, November 2010, 246-259. https://doi.org/10.1145/1879141.1879173

[38] Kreibich, C., Weaver, N., Maier, G., Nechaev, B. and Paxson, V. (2011) Experiences from Netalyzr with Engaging Users in End-System Measurement. *Proceedings of the* 1*st ACM SIGCOMM Workshop on Measurements up the Stack*, Toronto, August 2011, 25-30. https://doi.org/10.1145/2018602.2018609

[39] Weaver, N., Kreibich, C., Nechaev, B. and Paxson, V. (2011) Implications of Netalyzr's DNS Measurements. *Proceedings of the* 1*st Workshop on Securing and Trusting Internet Names* (SATIN), Teddington, April 2011.

[40] Gergo Ujhelyi, A.K. and Magyari, M (2021) Eclipse Titan Project Page. https://projects.eclipse.org/projects/tools.titan

[41] Eclipse (2021) titan.core. https://github.com/eclipse/titan.core

[42] Szalai, G. (2019) LANL2 Test Port for TTCN-3 Toolset with TITAN, Description. Version: 1551-CNL 113 519, Rev. A. https://gitlab.eclipse.org/eclipse/titan/titan.TestPorts.LANL2asp/-/blob/master/doc/LANL2aspCNL1135191551.adoc

[43] Szalai, G. (2019) IPL4asp Test Port for TTCN-3 Toolset with TITAN, Description. https://gitlab.eclipse.org/eclipse/titan/titan.TestPorts.IPL4asp/-/blob/master/doc/IPL4aspDescription.adoc

[44] Szalai, G. (2019) PIPE Test Port for TTCN-3 Toolset with TITAN, Description. Version: 1551-CNL 113 334, Rev. C.

https://gitlab.eclipse.org/eclipse/titan/titan.TestPorts.PIPEasp/-/blob/master/doc/PIPEaspCNL113334 1551.adoc

[45] Eychenne, H. and the Netfilter Core Team (2019) IPTABLES Man Page (1.8.3). https://ipset.netfilter.org/iptables.man.html

[46] Lyon, G.F. (2008) Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Com LLC (US), California.

[47] Rescorla, E. (2010) Keying Material Exporters for Transport Layer Security (TLS). Internet Requests for Comments, RFC Editor, RFC 5705. https://doi.org/10.17487/rfc5705

[48] Vitale, A. (2020) E-WASSI: Evolutionary Worldwide Application for System Security and Information. Corso di laurea magistrale in Ingegneria Informatica (Computer Engineering), Politecnico di Torino, Technical Report. https://webthesis.biblio.polito.it/18191/