ETSI TR 101 590 V1.1.1 (2013-03)

IMS Network Testing (INT);
IMS/NGN Security Testing and Robustness Benchmark

Reference
DTR/INT-00066

Keywords
robustness, security, testing

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee IMS Network Testing (INT).

1 Scope

The present document aims to introduce the basic principles and value of robustness testing, and highlight the importance of negative testing in a complex surroundings like IMS. It will introduce some guidelines for performing fuzz testing and provide a template for creating a test plan.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Ari Takanen, Jared DeMott, Charles Miller: "Fuzzing for Software Security Testing and Quality Assurance", Artech House, 2008. ISBN 1-59693-2147, 978-1-59693-2142.

[i.2] IEEE Std 610.12-1990: "IEEE Standard Glossary of Software Engineering Terminology".

[i.3] Software Assurance Model.

NOTE: Available at http://www.opensamm.org/.

[i.4] Gary McGraw, Brian Chess & Sammy Migues: "Building Security In Maturity Model".

NOTE: Available at http://www.bsi-mm.com/.

[i.5] The Microsoft Security Development Lifecycle (SDL).

NOTE: Available at http://www.microsoft.com/security/sdl/.

[i.6] Robindhra Mangtani, GSMA: "Security Issues in Femtocell Deployment", FCG.05. 23 July 2008.

[i.7] Yen-Ming Chen. IPTV: "Triple Play; Triple Threats". PacSec 2006.

NOTE: Available at http://pacsec.jp/psj06/psj06chen-e.pdf

[i.8] Raquel L. Hill, Suvda Myagmar, Roy Campbell: "Threat Analysis of GNU Software Radio", University of Illinois.

NOTE: Available at http://srg.cs.uiuc.edu/swradio/threat_wwc05.pdf.

[i.9] LORCON wireless packet injection library.

NOTE: Available at https://code.google.com/p/lorcon/.

[i.10] Month of Kernel Bugs (MoKB) archive.

NOTE: Available at http://projects.info-pull.com/mokb/.

[i.11] Andy Asava. Pure Security for VoIP Mobile Multimedia: "Femtocell Security Requirements", CDG Technology Forum. May 2, 2007.

NOTE: Available at http://www.cdg.org/news/events/cdmaseminar/070502_TechForum/.

[i.12] Barton P. Miller, Lars Fredriksen, Bryan So: "An Empirical Study of the Reliability of UNIX Utilities", University of Wisconsin Madison.

NOTE: Available at ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf.

[i.13] Takanen, Ari. Fuzzing: "The Past, the Present and the Future", SSTIC 2009.

NOTE: Available at http://actes.sstic.org/SSTIC09/Fuzzing-the_Past-the_Present_and_the_Future/.

[i.14] Wikipedia: "IP Multimedia Subsystem".

NOTE: Available at http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem.

[i.15] Peter Mell, Karen Scarfone and Sasha Romanosky: "A Complete Guide to the Common Vulnerability Scoring System Version 2.0".

NOTE: Available at http://www.first.org/cvss/cvss-guide.html.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

load testing: load testing uses large volumes of valid protocol traffic to ensure that a system is able to handle a predefined amount of traffic

performance testing: performance testing uses large volumes of valid protocol traffic to find the limits of how much protocol traffic a system is able to handle

robustness testing: robustness testing sends large volumes of invalid, malformed or otherwise unexpected traffic to the SUT in order to make it fail

NOTE: It is usually able to find completely new vulnerabilities from a tested system, in addition to pointing out existing, already known vulnerabilities. Robustness testing is also called fuzzing or fuzz testing.

vulnerability scanning: vulnerability scanning tests a system for security vulnerabilities that have already been reported in the public, i.e. known vulnerabilities

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

BG	Border Gateway
BSIMM	Building Security in Maturity Model
CPE	Consumer Premise Equipment
CSCF	Call Session Control Function
CVSS	Common Vulnerability Scoring System
CWMP	CPE (Customer Premises Equipment) WAN Management Protocol
DHCP	Dynamic Host Configuation Protocol
DoS	Denial of Service
EAP	Extensible Authentication Protocol

GSM	Global System for Mobile Communications
GSMA	Global System for Mobile Communications (GSM) Association
GTP	GPRS Tunnelling Protocol
HSS	Home Subscriber Server
HTTP	Hyper Text Transfer Protocol
HW	Hardware
ICMP	Internet Control Management Protocol
IDS/IPS	Intrusion Detection/Prevention System
IGMP	Internet Group Management Protocol
IMS	IP Multimedia Subsystem
IP	Internet Protcol
IPTV	Internet Protocol TeleVision
LTE	Long Term Evolution
MAC	Medium Access Control layer
MME	Mobility Management Entity
MSF	Multiservice Switching Forum
NAS	Network Attached Storage
NAT	Network Address Translation
NFS	Network File System (protocol)
PDN	Packet Data Network
PDU	Protocol Data Unit
PGW	PDN Gateway
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
SAMM	Software Assurance Maturity Model
SDL	Secure Development Lifecycle
SDR	Software Defined Radio
SGW	Signalling Gateway
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
SUT	System Under Test
TFTP	Trivial File Transfer Protocol
UA-AS	User Agent to Application Server
UE	User Equipment
WAG	WLAN Access Gateway
WLAN	Wireless LAN
XCAP	XML Configuration Access Protocol
XML	Extensible Markup Language

4 Introduction to Robustness Testing

Robustness testing is based on the systematic creation of a very large number of protocol messages (tens or hundreds of thousands) that contain exceptional elements simulating malicious attacks. In security community this testing approach is also called fuzzing [i.1]. This method provides a proactive way of assessing software robustness which, in turn, is defined as *"the ability of software to tolerate exceptional input and stressful environment conditions" [i.2]*. A piece of software which is not robust fails when facing such circumstances. A malicious intruder can take advantage of robustness shortcomings to compromise the system running the software. In fact, a large portion of the information security vulnerabilities reported in public is caused by robustness weaknesses. Robustness problems can be exploited for example by intruders seeking to cause a denial-of-service condition by feeding maliciously formatted inputs into the vulnerable component. Certain types of robustness flaws (e.g. common buffer overflows) can also be exploited to run externally supplied code on the vulnerable component.

The software vulnerabilities found in robustness testing are primarily caused by implementation-time mistakes (i.e. mistakes made during programming). Many of these mistakes are also vulnerabilities from a security point of view. During testing, these mistakes can manifest themselves in various ways:

- The component crashes and then possibly restarts.
- The component hangs in a busy loop, causing a permanent Denial-of-Service situation.
- The component slows down momentarily causing a temporary Denial-of-Service situation.

- The component fails to provide useful services causing a Denial-of-Service situation (i.e. new network connections are refused).

On the programming languages level, there are numerous possible types of mistakes which can cause robustness problems: missing length checks, pointer failures, index handing failures, memory allocation problems, threading problems, and so on.

Not all problems have a direct security impact, yet their removal always promotes the reliability of the assessed software component. In addition to increased information security, software robustness promotes software quality in general. A robust piece of software contains fewer bugs, which in turn increases user satisfaction and provides better uptime for the systems running the software. Proactive robustness analysis provides tools for assessing software quality. It is a complementary method to traditional process-based quality systems and code audits. Robustness weaknesses easily slip through ordinary code auditing and testing since robustness problems generally do not manifest themselves during normal operations. They only become visible when someone or something presents the implementation with a carefully constructed "malicious piece of input", or corrupted data.

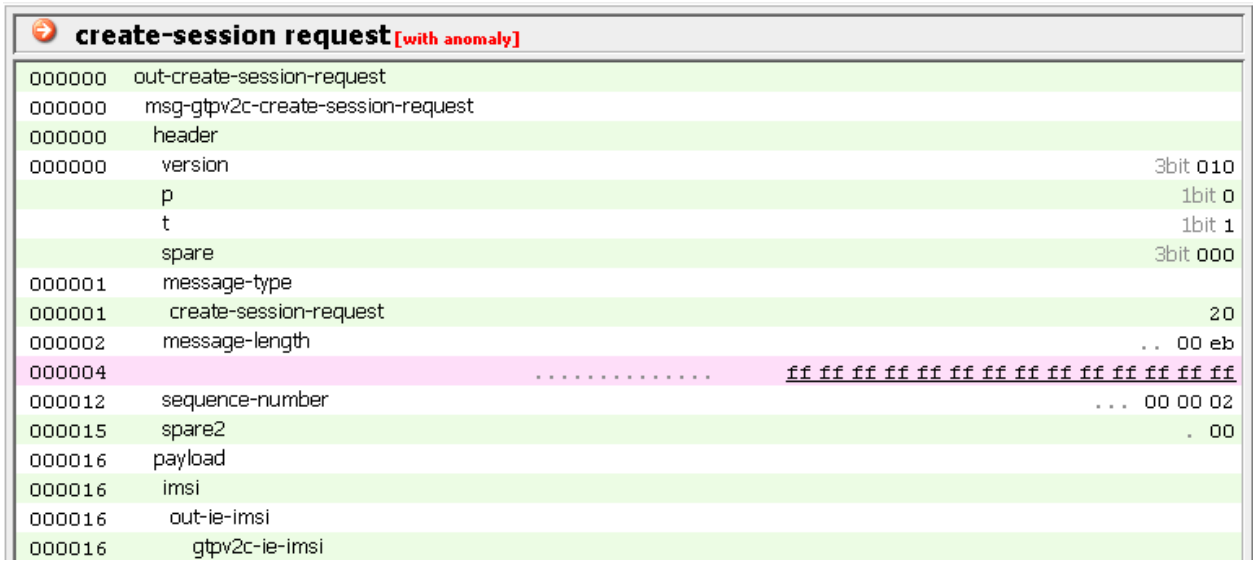

Figure 1: Example of a GTPv2 Create-Session-Request message with anomalous message length

4.1 Robustness testing in IMS

Complex systems tend to have more vulnerabilities than simple ones, therefore they should be tested with particular care.

The operators and service providers looking to operating in IMS surroundings are faced with the enormous complexity of the IMS network. Interoperability is a challenge, of course, and the various interfaces, number of players, protocols and applications pose a significant challenge for IMS security. While the advantage of a generally open, IP-based architecture is its flexibility, it also creates multiple interface points that operate as potential attack surfaces whose robustness and reliability should be secured.

Given the both ubiquitous and availability-critical nature of telecommunications services, flaws in telecommunications protocols may lead to widespread service disruptions which further emphasises the requirement for robust solutions.

5 Role of robustness testing in industry

As stated in the introduction, the technical purpose of robustness testing is to find and fix software errors which may or may not have security implications. It is important to expand beyond the technical know-how and look at the reasons why robustness testing is important and why it is being increasingly deployed by various organizations in different sectors.

5.1 Drivers for security and reliability testing with network equipment

If we look at the issue from the perspective of hardening a single network element, we can identify the following challenges:

- Fast release cycles
- Expanding code base
- Increasing amount of functionalities and communication interfaces integrated into devices
- Decreasing level of control over the code base due to outsourcing and software IP acquisition

While all these factors increase the need to produce hardened, secure and reliable devices, they also make the traditional processes, like code review, difficult or even impossible. Indeed, even open source software components which enjoy large scale peer review and deployment suffer from security incidents. As a result, several companies have started endorsing the Secure Software/System Development Lifecycle, promoting the idea that security should be built in rather than added on after deployment.

Continuous security and reliability testing with application specific test automation performing robustness testing is one of the integral components of secure development practices as cited in the *Software Assurance Maturity Model (SAMM)*, the *Building Security in Maturity Model (BSIMM)* and *Microsoft Secure Development Lifecycle (SDL)* [i.3], [i.4] and **[**i.5**]**.

5.2 Drivers for security and reliability testing with converging networks and IP-based services

The convergence of networks and the introduction of IP-based technologies open up many traditionally closed areas to new types of attacks. Protocol level attacks are commonplace in the Internet world. With the emergence of IMS, IPTV, and LTE the protocol level threats are becoming reality in the previously tightly controlled telecoms domains. It is out of the scope of the present document to analyze the threats for each of these technologies in detail, but some examples are given. To create a more thorough threat analysis from a protocol security point of view, all the architectures implementing the aforementioned services should be tested separately.

5.2.1 User plane traffic

From the service provider perspective, the hardest aspect to control is the user plane traffic. Sending malformed protocol data on the IP or any higher layer application protocol layer requires little sophistication. Moreover, there are plenty of tools available for launching protocol level attacks. The emergence of open and programmable UE platforms further increases the possibilities of would be hackers. Malformed data does not have to be connected with an intentional hacking attempt. Instead, it is more likely to be the result of device malfunction.

In the case of LTE and IMS, there are network elements which inspect user plane traffic for specific services such as deep packet inspection, IP header compression, NAT and so on. Therefore, there is a risk that service disruption might originate deep in the core network. To mitigate this type of threat, network elements should be tested for robustness both individually and also in a system configuration which enables end-to-end testing.

5.2.2 Consumer Premise Equipment (CPE) can pose a threat to core networks

Devices like Femtocells/HeNBs and IPTV Set Top Boxes enable partial core network access from consumer homes. This is significant paradigm change compared to closed cellular networks or traditional TV. By compromising CPE security and/or impersonating a CPE, an attacker could launch protocol level attacks directly against the core network. Other risk scenarios concerning the compromised CPE equipment include fraudulent usage of services, DoS attacks and eavesdropping in the case of Femtocells/HeNB. *Security Issues in Femtocell deployment* [i.6] by GSMA discusses Femtocell threats and suggests countermeasures in greater detail. Identified threats include device identity tampering, management interface weaknesses and Backhaul network security. Robustness testing can be used to mitigate these threats.

In IPTV networks, signaling that originates from CPEs reaches several nodes within the Content Source and Management Networks. During the Set-Top-Box booting, standard protocols such as DHCP, TFTP and NFS are used. Channel selections based on IGMP and Video on Demand operations are controlled with the RTSP. SIP, HTTP and XML content are used for various signaling purposes. All these protocols run on top of TCP/IP, creating a large attack surface that can be used to perform protocol attacks against the IPTV service infrastructure [i.7]. Typically, the IPTV infrastructure is protected with perimeter security (IDS/IPS, Firewall), but while these static security measures are necessary, they can only protect against known threats.

Remote management and the configuration of CPE devices with the protocols like CWMP (TR-69) is another aspect to consider. These protocols can provide an interface for tampering the CPE. They also include a channel back to configuration server. If CPE is compromised or impersonated by a computer, yet another attack vector against the core network opens up. CWMP (TR-69) is already used to configure Set-Top-Boxes and, in the future, it will probably be used in Femtocells/HeNBs as well.

5.2.3 Control plane security and integrity

In the case of R1 in LTE, the control plane initially remains closed. There is a theoretical possibility that by compromising the security of a CPE, one could gain protocol level access to the control plane and disrupt the core network from there. A more realistic threat, however, is posed by the emergence of software defined radios (SDR) and open UE platforms. For GSM, a software defined radio already exists. *Threat Analysis of GNU Software Radio* [i.8] discusses different threat aspects, in particular the potential for signal jamming with unauthorized frequency, which is the most serious threat to service availability. In LTE, most of the radio signaling terminates in the eNB, but the NAS connection does not terminate until the MME. Therefore, access into the NAS layer would also provide a pathway inside the core network.

Due to the early stage of the technology, control plane threat scenarios are still speculation. Still, a parallel to the Internet world can be drawn from Wi-Fi: It took a relatively long time from the emergence of Wi-Fi as a commonplace technology to the first public security incidents to appear. Wi-Fi matured as a technology in the late 90's, but it took more than five years after that before the first incidents with widespread publicity took place. The key here was the emergence of easy, low-cost technology with programmable environments for accessing the Wi-Fi MAC layer. Some initial frame injection drivers were introduced in 2002 -2004, but they were not maintained and did not evolve into general-purpose frame injection and hacking frameworks. It was not until mid-2006, when the first widely noted implementation-level vulnerabilities started to appear. *Lorcon wireless packet injection library* [i.9] was the first general framework to abstract HW intricacies, making frame injection relatively easy. Around the same time, the *Month of Kernel Bugs* [i.10] sought to disclose one kernel-level vulnerability from a common operating system each day of the month, including attacks using the 802.11 wireless frames.

5.3 Fuzzers as a hacker tool

Fuzzers [i.1] are typical tools for hackers looking for vulnerabilities in a system or seeking to initiate Denial-of-Service attacks. According to statistics the largest single attack category against SIP implementations was fuzzing attacks [i.11], attacks based on broken message structures.

A well known vulnerability category is buffer overflows. By allowing remote code execution, buffer overflows enable attackers to gain control over the victim's system. Buffer overflows are initially found by altering the input in various different ways and by experimenting with invalid inputs. This is exactly what fuzzing does. The typical workflow for creating an exploit is to search for vulnerabilities using fuzzing and upon finding one, to verify its exploitability. Probably the most common indication of exploitability is the access to instruction pointer: If the attacker is able to manipulate the program's instruction pointer value using invalid input values, the likelihood that the vulnerability is exploitable is very high. When the vulnerability is found to be exploitable, the next step is to create an actual exploit, which could enable execution of malicious code injected by attackers.

In the case of DoS attacks, input with unexpected anomalous data or broken data structures often amplifies the effect. In other words, system may be able to tolerate certain amount of load, but when that load is negative, the system ends up in a DoS state. This is another common outcome of fuzzing used for malicious purposes.

By using robustness testing and fuzzing pro-actively before the deployment, it is possible to harden the devices and services against similar attacks in the production environment.

6 Characteristics of robustness testers and fuzzers

Typically robustness testers and fuzzers are either software based tools or hardware appliances. Commercial tools are usually all-inclusive, meaning that there is a predefined set of protocols supported by the tests with no development required from end user. In addition, extension capabilities for supporting proprietary protocol extensions or proprietary protocols are usually provided.

Most open source tools are fuzzing frameworks [i.1], which provide the users with a basis for creating a fuzz test tool for their chosen protocol. To be more specific, users are typically expected to describe the protocol and its functionality, while the framework automatically generates the test cases with anomalous content, or attack patterns.

Regardless of the tool type, the following functionalities are commonly offered:

- **Automatic generation of test cases or pre-built test cases:** As noted earlier, robustness testing is based on sending large number of malformed protocol messages to simulate attacks. The number of test cases varies from a few thousands to hundreds of thousands and, therefore, it is not feasible to produce the test material manually.
- **Automated execution of tests:** Due to the large amount of test material, fully automated test execution is the only feasible approach.
- **Simple pass/fail criteria - crash or no crash:** This is the original pass/fail criteria used by fuzzing and supported by all tools. Lately, there has been progress in the detection of more fine-grained anomalous behavior of system under test.
- **Built-in monitoring of a system under test:** SUT monitoring is usually called "instrumentation" and it is usually divided into in-band and out-of-band instrumentation. Instrumentation refers to a mechanism used to monitor the state of SUT. In-band instrumentation is used to obtain the crash - no crash status. It usually consists of sending a valid protocol request to the SUT. If, for example, the tested protocol is IP, the test tool can use valid ICMP Echo for checking whether the SUT still responds. Out-of-band instrumentation refers to external mechanisms used to obtain SUT state information. A test tool performing SNMP query on a SUT is an example of out-of-band instrumentation.

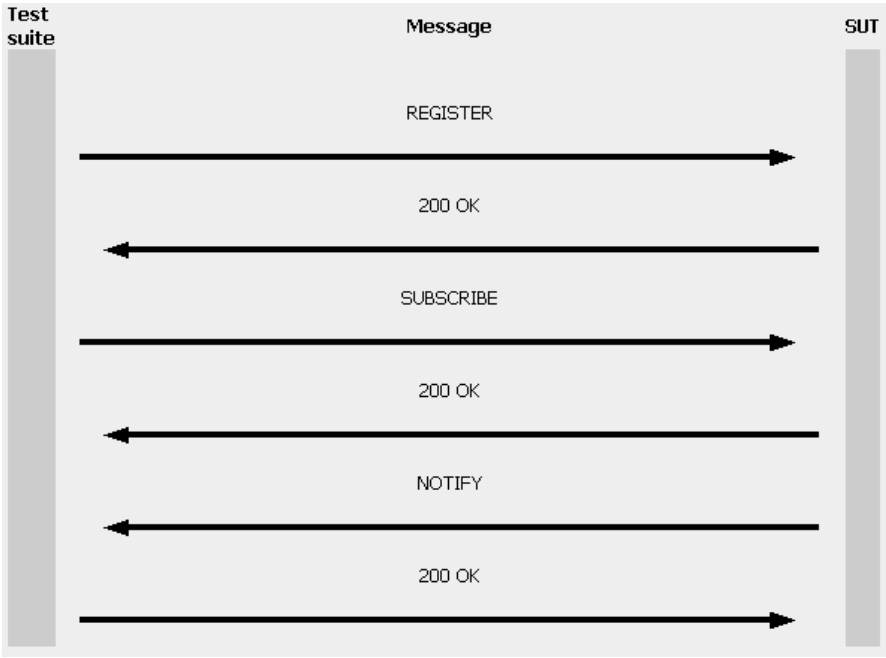

Figure 2: Example message flow from IMS Registration, where an attack can be in any of the messages

6.1 What can be fuzz tested

In a broad sense any application or interface processing some kind of input from users or other devices can and should be tested with fuzzing to harden it against unexpected input. The first fuzzing implementation was designed to test Unix command line utilities and their tolerance for unexpected input [i.1] and [i.12]. Today, fuzzing is used in very diverse areas. Network protocols and their security is the primary interest of this technical report, but other application areas like GUI's, digital media parsers, web services and hardware driver interfaces can also be tested by fuzzing.

7 Model-based and Mutation-based fuzzing

There are two main approaches to fuzzing: model-based testing and mutation. The model-based approach is also commonly known as robustness testing due to its more systematic, non-random methods.

In the model-based approach, test tools are created based on protocol specifications. An executable simulation of the protocol under test is created. In other words, if the system under test is LTE/EPC PGW and S5/S8 is the tested interface, the test tool would simulate the SGW. Mutation-based fuzzers are based on traffic captures that are captured from the interface of interest. The traffic capture is fed into the test tool, which then generates "mutations" of the captured packets and feeds them to the system under test.

In following clauses, both approaches are briefly described and their relative strengths discussed [i.13]. A more complete description of different fuzzer types can be found from book *Fuzzing for Software Security Testing and Quality Assurance [i.1]*.

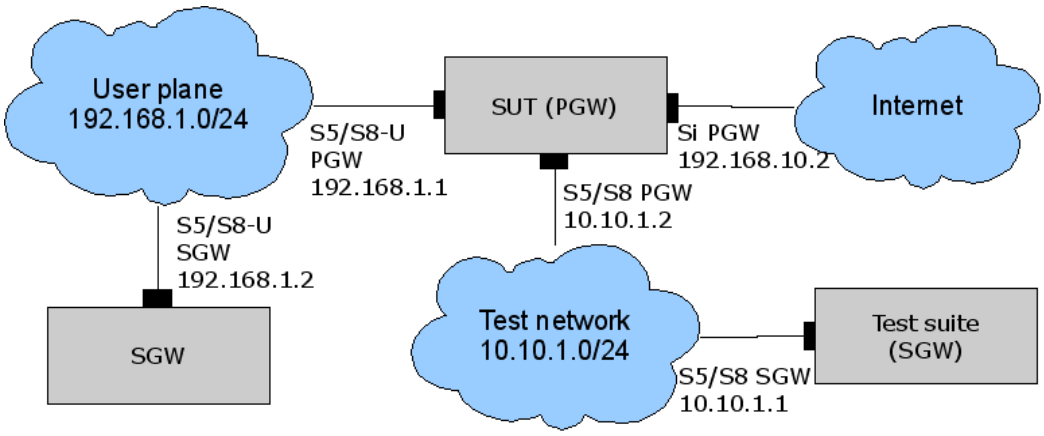

Figure 3: Example of robustness test setup for LTE/EPC PGW where test suite simulates SGW

7.1 Model-based Fuzzing - Robustness testing

In model-based fuzzing, an executable model is created from communication protocol specifications and state-diagrams. In the process of model-based fuzzer development, each data element (from each protocol PDU) is identified based on protocol specifications and added to the model. In many cases, the specifications also give metadata information about the different fields, such as allowed values for given element or the range of values. This metadata information is used in the test material creation process in which targeted invalid values are inserted in the test cases. Typically, the invalid values include boundary checks and values that significantly deviate from the allowed ones. In model-based fuzzing, test generation is systematic, and often involves no randomness at all. The test material generation process can be fully automated or manual. In both approaches, the protocol model is populated with invalid values designed to stress system under test. The end result of the model based fuzzer creation is the full implementation of one of the end-points of a communication network, specifically, the end which primarily sends anomalous inputs. The main benefits of model-based approach are:

- **Measurable test coverage:** As the tests are typically generated from the interface/protocol specifications themselves, the tools will also be able to enumerate the specification coverage of the tests. All tested protocol functionalities, elements, sequences and the actual anomalies used in the tests will be included to the test results. Everything that is tested is easy to reproduce.

- **Optimized test efficiency:** A model-based approach contains an optimized set of tests to cover the specifications, thus providing thorough test coverage of the actual implementation. Traffic capture based mutation fuzzing generally ignores rarely used protocol features (as they are not seen on wire and captured). In many cases, these rarely used protocol elements are the most vulnerable parts, because they are not subjected to heavy day-to-day usage and as a result of that there is hardly any bug elimination. From a security perspective it does not matter whether the vulnerability is in a widely or rarely used part of the protocol: As long as the problem is there, the system is 100 % vulnerable even if the feature is only used in 1 % of day-to-day operations.

- **Test execution time:** Intelligent Model-Based tests use a targeted approach to test all the protocol PDU's, all the fields in the PDU's and all the PDU exchanges. Since the tested protocol is thoroughly known, the most effective attack patterns for different fields can be applied. This helps in keeping the test run time reasonable without compromising the comprehensiveness of the tests. Short test execution times also allow the integration of tests into regression sets and automated nightly and weekly test runs.

- **Stateful testing support:** Model-based testing enables easy support for stateful testing since the tests are generated based on specifications and state machines to begin with. This enables test tool to simulate real implementation and test all the state transitions of the SUT protocol parser.

7.2 Mutation-based fuzzing

The most straightforward method of building a fuzzer is based on re-using a test case from feature testing or performance testing, whether it is a test script or a captured message sequence, and then augmenting that piece of data with mutations, or anomalies. The simplest form of mutation fuzzing is based on random data modifications such as bit flipping and data insertion. More advanced mutation-based fuzzers can break down the structures used in the message exchanges, and tag those building blocks with meta-data, which is used in the mutation process. This approach produces tests that are better targeted at weaknesses commonly associated with certain types of protocol elements. The main benefits of the mutation-based approach are:

- **Scalability to new protocols:** Mutation-based fuzzers do not require protocol specifications to analyze protocols. Instead they utilize network traffic captures to generate test material for a given protocol. Mutation-based fuzzing a very attractive alternative when no prior tools exist for the given protocol and there is no incentive to invest in the development or purchase of such tools.

- **Easy to execute:** Since tests generation is based on traffic captures, the protocol options required to establish interoperability with the SUT are already set correctly when the fuzz tests are executed against same target the capture was taken from. Usually there is an easy way to change the basic settings like addresses and ports when executing against a SUT in a different network setup.

- **Proprietary protocols:** A wide range of proprietary protocols for which neither commercial nor open source tools exists are used in industry. Mutation-based fuzzing solutions are a viable alternative for these protocols.

8 Case study: Planning robustness testing for IMS service

In this case study, we outline the process of planning robustness tests for IP-based services. IMS architecture is used as the example target system.

An integral part of the planning process is the attack surface analysis, which is performed by studying the exposed protocol interfaces in the system. First, we outline the general framework that has been used in real-world service engagements for the attack surface analysis and then apply this to the example architecture. The primary purpose of the planning is to identify the most critical interfaces and to prioritize the testing effort. Moreover, mapping out the attack surface plays an important role in the overall threat analysis.

It should be noted that the use case presented here is written from the perspective of a service being deployed, and the interfaces are analyzed from the perspective of an end user or an outside attacker. Other perspectives include the network equipment manufacturer view, and the trust boundaries between operators. These are briefly discussed in the conclusions of this case study.

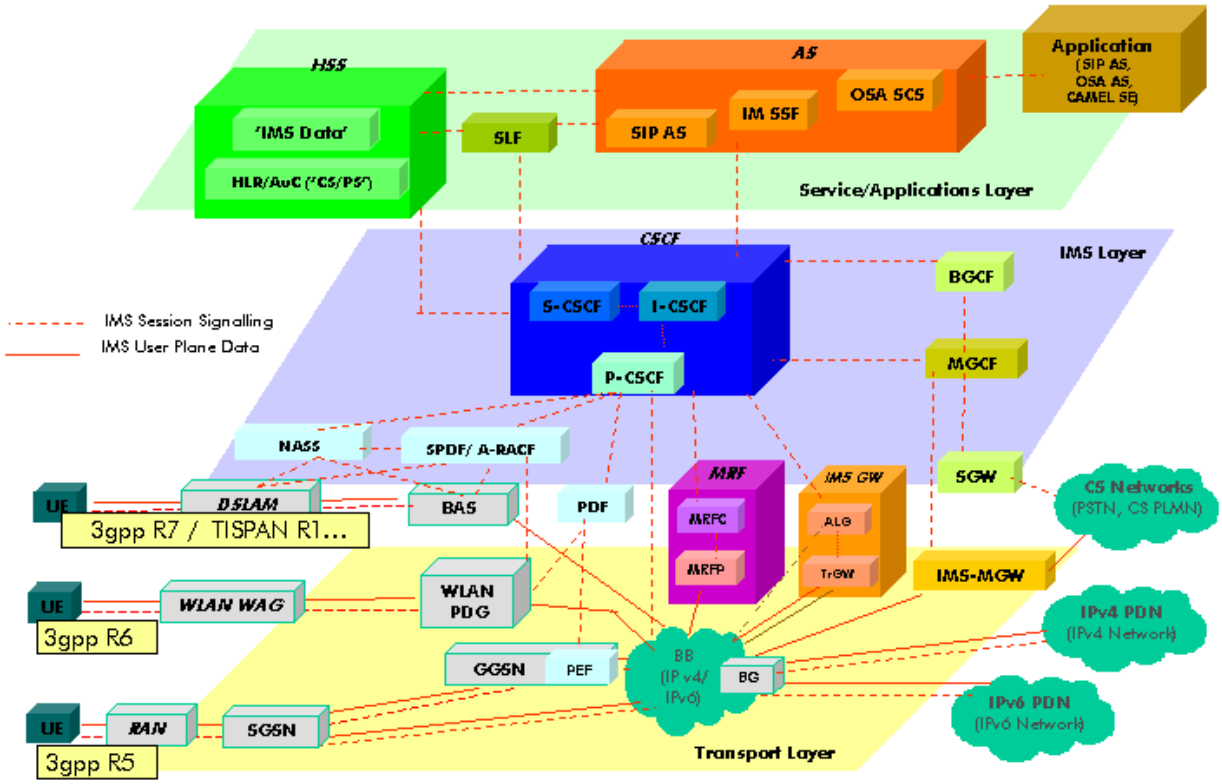

Figure 4: IMS reference architecture [i.14]

8.1 Attack Surface analysis primer

The attack surface analysis is synonymous to studying the protocol interfaces and understanding the data paths inside the network implementing the service. This is a starting point for mapping threats and understanding the exposure of a system. With this information you can, for example, limit the scope of testing required to achieve sufficient confidence in a service. You will also understand how attackers could gain access to the valuable information inside your network. There are several methods for conducting an attack surface analysis including scanning, passive monitoring, deducting the physical properties of the system, and observing the configuration. These methods are complementary, and more than one of them should be used to map all the possible ways into the system's open interfaces.

Under a relatively simple cover, the complexity inside the devices is growing exponentially. Present day networked devices such as mobile phones, multi-access terminals, routers and switches contain a large number of protocol interfaces.

To address this growing complexity, we first need to have the methodology in place for discovering and analyzing the interfaces of a given device, service or network offers. Browsing the documentation is usually a start, but it does not tell the entire truth about the running device. Interface analysis is the first step in blackbox robustness testing.

Here is a list of several different ways to map a set of open interfaces in a given device:

1) technical specifications
2) scanning or probing
3) passive monitoring
4) on-host process and resource monitoring
5) observing the physical properties of the device
6) observing the configuration or the user interface of the device

All the methods listed above are complementary and more than one of these should be used to gain the best insight possible into the open interfaces.

8.1.1 Technical Specifications

The device's technical specifications and/or network architecture provide a starting point for the interface analysis. Specifications are supposed to cover all the intended functionalities supported by the protocol interfaces.

Specifications, however, are not sufficient by themselves, since:

1) They are often complex and hard to read.
2) They may lack some information altogether, since they are not necessarily up-to-date.
3) The device may have unspecified functionalities that not even the vendor is aware of.

8.1.2 Scanning or Probing

Active scanning and probing should be used to identify the wired and wireless interfaces of a devices and systems. Active probing aims to identify the actual active protocol interfaces inside the various physical interfaces of a given device. Active probing is complementary to a specifications review. It should be used to verify the results the specifications have indicated and more importantly to reveal potential interfaces not revealed by the review. Different tools should be used to scan the different physical interfaces of the device. For example, nmap (http://www.nmap.org) is a widely used tool for scanning the open ports in IP networks.

8.1.3 Passive Monitoring

In addition to active probing, passive monitoring should be used to gain better knowledge of the actual device operations. Subsequently, this reveals the protocols and interfaces used for these operations. Passive monitoring should target at least the following phases of the device operation:

1) Start-up
2) Normal operation
3) Failure and subsequent service restart
4) Shut down

It is likely that the device or the system under inspection will perform functionalities that are not evident via perusing the specifications or active probing. Such unexpected functionalities may be, for example, a call-home feature, where the device reports the failure it has encountered to the device vendor. An example of a network flow analysis using a passive monitoring tool is shown in figure 5.

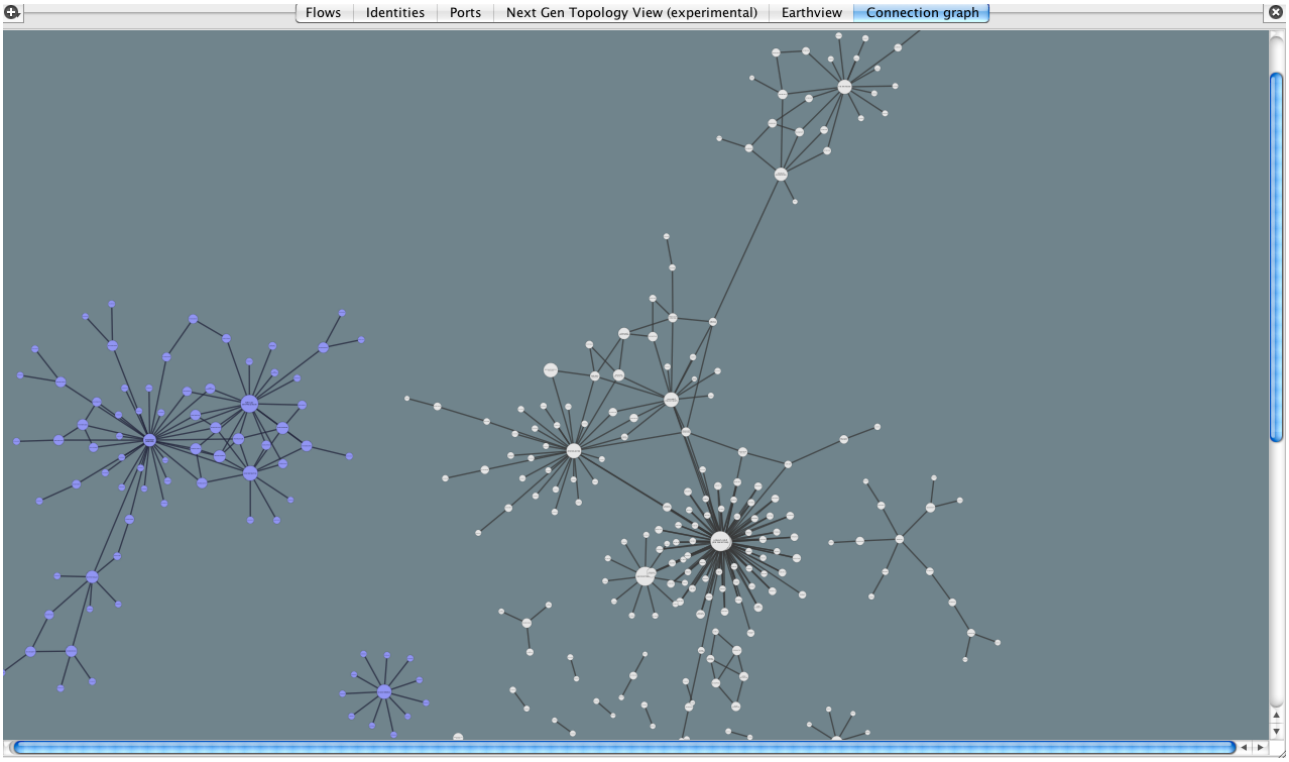

Figure 5: Network flows in IMS test network visualized using a passive monitoring tool

8.1.4 On-host process and resource monitoring

Most operating systems and platforms provide utilities to map processes that listen to network resources. Using these utilities is much more reliable than relying on external scanning or probing.

8.1.5 Physical Properties

Physical properties of a device often reveal what types of hidden functionalities the analysis is likely to expose. All the ports and the connectors that are visible to outside world are obvious targets for the analysis. It is also possible to disassemble the device to get further information on the potential interfaces to connect to. These could be, for example:

- Memory slots
- Disconnected, but easily re-enabled ports
- Debugging ports on the circuit board

In an examination of a device's physical properties, the following features should also be taken into account:

- All the ports and slots the device has for interconnectivity
- Power slots
- Buttons
- External connectors

8.1.6 Configuration

The device configurations used to build the service should be examined to gain knowledge of the device itself as well as the possible open interfaces. Different services available on the device can result in different configurations. For this reason, it is important to know which are the potentially available services and not only what is enabled on the device in its default or current configuration.

8.1.7 User Interface

The purpose of the user interface inspection is to gain insight into the processes running in a system. The systematic observation of the available applications and the configurations they allow is likely to reveal interfaces that would never be located with external monitoring methods. Examples of such interfaces are different images, archives, audio formats and certificate stores. Many of the device's applications initialize client-side connections and accept these formats.

8.2 Expected outcome of Attack Surface Analysis

As stated earlier, modern networks are extremely sophisticated and while the complexity is increasing, the security requirements are becoming stricter. It is apparent that random testing is no longer a viable option for testing network level connections.

Attack surface analysis facilitates threat analysis on two different levels. While helping to identify the most critical interfaces, it can also assist in finding and shutting down unnecessary open ports or services. By minimizing unnecessary complexity, customer support cases and security exposures can be prevented and the testing effort can be concentrated on truly critical interfaces.

8.3 Attack Surface Analysis for example architecture by studying the interfaces

In our example analysis, we can only access Wiki-based specifications. Therefore we cannot rely on port scanning, traffic monitoring, on-host monitoring, or physical properties of devices. Since it is not a concrete, real-world deployment, some assumptions about the protocols and their termination points are necessary. Port scanning and passive monitoring were deemed likely to expose additional data paths and termination points. After all, an example should give a fairly realistic view, no matter how limited it is.

For prioritization, the CVSS Exploitability Metric was used (explained later).

8.3.1 Interfaces and protocols

Interface list in table 1 considers interfaces which terminate signaling that originates from the UE or from outside the network, or performs packet translation for data originating from these sources.

In addition to the protocols listed above, it is likely that protocols like DHCP, IPSec, and SigComp are found in the network. Since the termination points are not obviously based on the attached diagram they are not included in the analysis, even though in a real-world scenario they should be.

Table 1: Example list of interfaces in IMS

Interface	Protocols	Comment
UE-SGSN	GTP	
UE- WLAN WAG	802.11	
UE-P-CSCF	SIP	Commonly replaced with Session Border Controller
UE-S-CSCF	SIP	
UA-AS	HTTP, XCAP	Subscriber information
UE-IMS-MGW	RTP	
IPv4/6-BG	IPv4, IPv6	
UE - IPv4 PDN	User plane protocols	End-to-end traffic originating from UE and terminating in IPv4 PDN

It is also important to test the routing protocols and Layer 2 switching protocols, which are overlooked by this example. While these protocols are more challenging to attack than application level protocols, both have had their share of security incidents. Data paths are more difficult to predict with these protocols and applying perimeter security is challenging. Therefore the key is to ensure the robustness of the devices implementing the routing and switching functions.

8.3.2 CVSS Scoring

The CVSSv2 Exploitability Metric was used to prioritize the interfaces. CVSS is an industry standard for classifying vulnerabilities [i.15]:

"The Common Vulnerability Scoring System (CVSS) provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. CVSS consists of 3 groups: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score. The Base group represents the intrinsic qualities of a vulnerability. The Temporal group reflects the characteristics of a vulnerability that change over time. The Environmental group represents the characteristics of a vulnerability that are unique to any user's environment. CVSS enables IT managers, vulnerability bulletin providers, security vendors, application vendors and researchers to all benefit by adopting this common language of scoring IT vulnerabilities."

While the primary use case of CVSS is to estimate the impact of realized vulnerabilities, Exploitability Metrics provide an easy subset for calculating numerical values for potential threat. Components and component values of the Exploitability Metrics are:

- Access Vector (AV)
    1) Network [N]
    2) Adjacent network [A]
    3) Local [L]
    4) Undefined
- Access Complexity [AC]
    1) Low [L]
    2) Medium [M]
    3) High [H]
    4) Undefined
- Authentication (Au)
    1) None [N]
    2) Single instance [S]
    3) Multiple instances [M]
    4) Undefined

The highest score and the highest risk (10) is accredited to an interface with the following features:

1) Access Vector: Network
2) Access Complexity: Low
3) Authentication: None

Expressing this vector using the terms defined in the CVSS Guide provides: AV:[N]/AC:[L]/Au:[N]. This format was used to create a table presented in the next clause. For details on how to calculate the numerical values, please refer to CVSS equations.

8.3.3 Scoring

Only one protocol from each interface will be included in this analysis. While it is important to analyze end-to-end user plane protocol traffic, the scores calculated for this interface are very general. To be able to calculate the scores more accurately, we should have more information about the level of user plane traffic parsing, for example from deep packet inspection, filtering and compression taking place within the network. The functions/nodes should then be identified and the vectors calculated for them individually.

IPv4 Packet Data Network interface towards the IMS Transport layer is omitted, because there are no details on the implementation.

Table 2: Attack vector scores for IMS interfaces

Interface	Protocol	Vector	CVSS Score	Comment
UE-SGSN	GTP	AV:[N]/AC:[H]/Au:[N]	4,9	
UE- WLAN WAG	802.11	AV:[N]/AC:[M]/Au:[N]	8,6	
UE-P-CSCF	SIP	AV:[N]/AC:[L]/Au:[N]	10	
UE-S-CSCF	SIP	AV:[N]/AC:[L]/Au:[S]	8	
UA-AS	HTTP	AV:[N]/AC:[L]/Au:[S]	8	
UE-IMS-MGW	RTP	AV:[N]/AC:[M]/Au:[S]	6,8	
UE- IPv4/6-PDN	IPv4	AV:[N]/AC:[M]/Au:[S]	8,6	

8.4 Attack Surface Analysis conclusions and recommendations

The purpose of the Attack Surface Analysis is to identify the most critical interfaces and to help in prioritizing the testing effort. In an ideal world, all the interfaces would be tested, but in reality, budgets, deployment schedules and the availability of tools often impose limitations on what is feasible. Thorough attack surface analysis facilitates the selection process and it is equally important for the outcome of the tests to eliminate unnecessary open services and ports.

8.4.1 Attack Surface Analysis as part of Threat Analysis

When looking at threat analysis from protocol perspective, the attack surface analysis is clearly a critical component. In addition, the motivations and the likelihood of an attack should be considered when making the final decision about what to test. Factors like financial gain or the possibility to acquire user database information increase the likelihood of attacks. Overall threat analysis is a broader topic than protocol level analysis and therefore it is out of the scope the present document. On the other hand, factors like the physical tampering of devices, access control, and IT policies on passwords are considered.

It should be noted that CVSS is just one possible way of prioritizing interfaces for testing. There are other formal classification methods and in many cases common sense is enough. In this example, the impact of compromising interface has not been taken into account. For example, attacking the WLAN Gateway has a high score, but it is questionable how serious the damage caused by compromising it could be. For applying CVSS more broadly in threat analysis, the use of Impact Metrics can be considered. Impact Metrics is another component of the CVSS Base Score and it provides a tool for estimating the impact of compromising a certain function or service.

8.4.2 Alternative perspectives on Attack Surface and Threat Analysis

This case study was written from the perspective of a service provider and the threat was considered to come from outside. As mentioned earlier in the introduction of this case study, other relevant perspectives are the network equipment manufacturer view and the trust boundaries between operators.

8.4.3 Attack Surface Analysis of network equipment

When performing the attack surface analysis on a single network device, the framework presented earlier still applies but the focus is slightly different compared to the service level analysis. The most important premise for analysis is that no assumptions about the open interfaces can be made. In service level analysis, it is no longer necessary to concentrate on the interfaces used. Indeed, on a device level all the interfaces should be hardened. Even management ports, which were never meant to be accessed through outside local network, may be accidentally enabled.

In a network equipment level analysis, the physical properties and the configurations/user interface can be given more attention compared to a service level analysis. Scanning the device and passive monitoring are not as crucial, since the supported protocols are known. From an auditing perspective it might still be a good idea to check that the enabled services match the configuration and specifications.

8.4.4 Trust boundaries as Attack Surface

Technical part of the trust boundary attack surface analysis is very similar to a case presented earlier. The interfaces and protocols forming boundaries between operators are identified and their exploitability and impact of incidence are estimated.

To keep the metrics comparable, it is important to decide whether trust boundaries are analyzed as a separate case or together with external threats. This should be clearly stated in all report. In case both external interfaces and trust boundaries are considered, it is probably safest to analyze them on the same scale.

On one hand, trust boundaries are more protected than external interfaces, but on the other hand, the exposure of critical information and functions is also much greater with trust boundaries. This is true both in terms of functions with granted access over trust boundaries and the number of protocols exposed. Therefore, in the assessment of the possibility of an attack and the magnitude of its impacts play a crucial role in the overall threat analysis involving trust boundaries.

9 Test plans and certification

9.1 Robustness test plan

In creating robustness test cases for a protocol, the number of different possible inputs is infinite. For this reason, a subset of inputs covering the largest possible number of different protocol messages and elements with the best possible test efficiency should be carefully selected and created. In systematic test material creation, the goal should be to apply anomaly values for every element in the protocol PDU (including structural level anomalies, like leaving an element out of the PDU altogether) and for all the PDU's used by a given protocol. In addition, message sequence level anomalies should be present in good quality robustness test set. Repeating messages, omitting messages from a sequence and sending unexpected messages can confuse the protocol parser state machine with serious consequences, should these vulnerabilities remain in the production version. Any available tools for code coverage metrics can be used to provide information on the proportion of software statements covered by the tests.

From the test plan creation perspective the fact that the number of possible input combinations is infinite imposes some differences compared to interoperability test plans. It will not be possible to define the input data for all the test cases since the typical case amount varies from thousands to hundreds of thousands, depending on the protocol. It is still possible to describe a test setup including protocols, message sequences and tested network elements in detail. The suggestion is to limit the amount of details on this level and to provide each test scenario with an example of all the anomalous messages sent during the test.

9.2 Robustness certification

To date, no vendor-independent certification body or process exists for robustness testing in the telecommunication industry. The basic challenge with robustness certification reflect the difference between interoperability test plans and robustness test plans: Traditional certification assumes that the inputs of a system are specified in detail and the expected response from the SUT is equally carefully specified. As discussed in the previous clause, it is not feasible to provide detailed specifications on all the robustness test inputs. In the case of certification, the pass/fail criteria also cause equal challenges. The original fuzzing criteria "component crash" or "no crash" are still valid, but the milder problem indicators, such as excess processor power consumption or excess memory consumption/corruption, are harder to detect and judge. The responses from the SUT are difficult to verify because one may argue that it is equally fine for an implementation just to ignore anomalous request than to return an error code.

Furthermore, passing the robustness tests is in not a guarantee for a vulnerability-free system. A complete security audit of a system requires many actions besides robustness analysis and it should also cover analysis of the design of the system in general, its usability, and other security aspects. Robustness analysis can be used as an integral part of a complete audit, or as a standalone method to provide insight into the security and quality of the tested software component. Therefore, the certification should indicate a tolerance of anomalous protocol input, i.e. robustness, and it should not be confused with an all-around security certificate.

Should robustness certification be pursued, the tools criteria used in the certification should be defined in vendor neutral, yet concrete, terms. The tool criteria should include requirements on test material coverage, for example, it should guarantee tests for all elements of the PDU, all the PDU's used in the protocol and sequence level tests. SUT monitoring mechanisms and common practices for pass/fail verdicts should also be agreed on.

Despite the challenges with robustness certification, it is an interesting topic for further discussion. There are clear indicators that the role of fuzzing and robustness testing is growing as companies are seeking to increase built-in security rather than adding security features on top of inherently insecure systems. Therefore, it is conceivable that the interest for robustness certification will also increase.

10 Conclusions and further work

In the present document, we have discussed robustness testing and its relation to fuzzing. The role of robustness testing in industry was discussed, before taking a deeper look at the technical aspects of robustness testing tools and the fuzzers on the market today. Relations between robustness testing, attack surface analysis and threat analysis were also illustrated using a case study, and finally, some ideas for future work were discussed.

In addition to agreeing on a format for test plans and certification considerations, further work on attack surface- and threat analysis on LTE, IMS, and IPTV networks could be in the general interest for future work.

An example of test plan is contained in archive tr_101590v010101p0.zip which accompanies the present document.

History

Document history		
V1.1.1	March 2013	Publication