Generating attack scenarios for the validation of security protocol implementations

Eliane Martins[1], Anderson Morais[1], Ana Cavalli[2]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Caixa Postal 13.083 -970 – Campinas – SP – Brazil

[2]Institut Telecom – Telecom & Management Sud-Paris
Evry, France.

{eliane,Anderson.Morais}@ic.unicamp.br, Ana.Cavalli@it-sudparis.eu

Abstract. *In this paper we consider the validation of security protocols, whose aim is to ensure some security properties when the communication medium is not reliable. The goal is to uncover protocol vulnerabilities that an attacker can exploit and cause security failures. Our approach uses a fault injector to inject attacks into a communication system and observe whether the security properties are violated. One of the key problems is: how to generate successful attacks that will indicate the existence of vulnerabilities? We propose an approach that is similar to model-based testing, as we derive attack scenarios from an attack model representing known attacks to the protocol under test. The approach can be completely supported by tools, as is shown in the paper.*

1. Introduction

Distribution is a dominant aspect in computer systems nowadays, either for big, service oriented applications or for embedded systems in cars or spacecrafts. Communication systems are a key factor for distribution to take place. What we cannot ignore, however, are the communication systems vulnerabilities , resulting not only from bugs introduced during development but also from intrinsic characteristics of such systems. For example, wireless networks are more vulnerable than their wired counterparts because of intrinsic characteristics such as the limited resources of the mobile terminals, or the absence of physical limits, which makes it possible to any device to connect to the network, given that it is in the range of the transmitted signal.

Vulnerability removal is thus very important to reduce the risk of successful attacks. Different Verification and Validation (V&V) techniques can be used for that purpose, such as analysis [ERG+07], formal proofs [GM95], model checking [Low98], or testing [PRO99, OAC07]. In this study we propose the use of Software-Implemented Fault Injection (SWIFI) for vulnerability detection and subsequent removal. SWIFI is a powerful technique to validate dependability aspects and has been used in various application domains, such as communications protocols [DJM96, DJT+05].

Fault injection has also been used to validate security properties. For example, Thompson et al [TWM02] tested software security in a hostile environment. Faults were injected during runtime by monitoring and modifying system calls made by the application to the operating system. In this sense, they emulate the behavior of a hostile environment.

However, they do not mention how to associate this behavior with attacks. Another work presents the design and development of a fault injector for the testing of firewalls and intrusion detection systems [WW03]. The fault injector provides functions to the user, such as *dnsspoof*, *tcpkill* and others, to simulate attacks to TCP/IP protocols. The AJECT tool [NAC+06] also inject attacks for vulnerability detection and removal of security protocols. In these works it is not mentioned how to model the attacks that are used during testing.

We present how to model attacks to generate attack scenarios to be injected during runtime. Attacks are modeled based on information about successful attempts and known vulnerabilities that are available from different sources (books, Internet, and so on). Given that we mean to test security protocols for wireless networks, we based our model on information about this kind of network, although the approach can also be applied to other protocols as well. The contributions of this work are with respect to: i) fault representativeness; ii) how to inject attacks. In what concerns fault representativeness, we define a set of faults that correspond to known successful attacks. For the second point we define attack scenarios in terms of communication faults to be injected.

The paper is structured as follows: Section 2 presents some main concepts necessary to understand the approach description. Section 3 contains a description of the approach showing in details how to obtain the attack scenarios and how to map them to communication faults. Section 4 applies the proposed approach to a case study and Section 5 closes the paper, presenting some directions for future works.

2. Context of the work

2.1. Security impairments

Figure 1 represents the causes of failures of a security system due to several classes of faults [CCD+00, NAC+06].

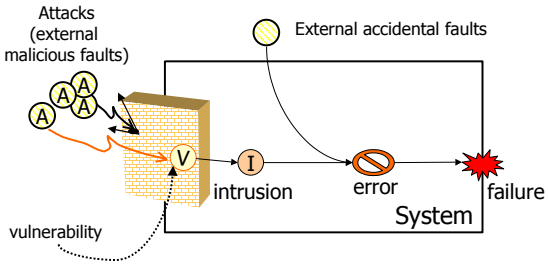


Figure 1. Threats to security.

From the figure we can see that failures can be caused by two categories of faults: malicious (attacks) and accidental ones. *Attacks* are malicious external activities aimed to intentionally violate one or more security properties of the system. Attacks can be passive or active. A passive attacker attempts to learn or make use of information from the system but does not affect system resources. An active attacker, on the other hand, attempts to alter system resources or affect their operation. Attackers exploit system vulnerabilities to obtain their intent. A *vulnerability* is malicious or non-malicious faults introduced during development phases (requirements, analysis, design or configuration) of the system or in the way it is used, that could be exploited to create intrusion. An *intrusion* is a malicious interaction fault resulting from an attack that has been successful in exploiting a vulnerability. An intrusion can take the system to an erroneous state. In case the errors are

not detected and recovered, a *failure* will occur, i.e., the system specification, and, in particular, the system security properties are violated. As the figure also shows, some attacks may not lead to intrusions, as they can be prevented by the existing system defenses. Figure 1 also shows that failures can occur as a consequence of non-malicious or accidental external faults, i.e., faults that are consequences of failures in components external to the system boundaries that interact with it (e.g., operating system or hardware faults).

For security validation, in the same way as for dependability, two activities can be performed [ALR+04]: fault removal and fault forecasting. *Fault removal* aims at reducing the presence of vulnerabilities in the system. *Fault forecasting* aims at estimating the presence, creation and consequences of vulnerabilities, attacks and intrusion [CCD+00]. In this study the focus of validation is fault removal. The intent is to detect system vulnerabilities in the presence of faults, malicious (attacks) or non-malicious ones. Given that the target systems are security protocols, the fault model comprises attacks to these protocols as well as communication faults.

2.2. Security properties

The basic security attributes are confidentiality, integrity and availability [ALR+04]. *Confidentiality* is the prevention of unauthorized disclosure of information. *Integrity* is the absence of improper system state alteration, that is, the prevention of unauthorized modification or deletion of information. *Availability* is the prevention of unnecessary withholding of information; in other terms, it is readiness for use.

In what concerns security protocols, other properties are commonly used, which can be expressed as a combination of the aforementioned security attributes. For example, *non-repudiation* is the availability and integrity of some information about the message, such as message origin (e.g., creator identity, time of emission) or message reception (e.g., receiver identity, time of reception).

2.3. Attacks to wireless networks

Wireless networks present more security risks than their wired counterparts, for their very nature. First of all, wireless networks are not physically limited, in the sense that the radio waves used as communication channel propagate quite well. Other limitations can be pointed out: channel capacity, network bandwidth, memory and processing power. Besides, they are more subject to noise and other interference in the channel. These are some of their inherent vulnerabilities that are exploited by attackers.

In the remainder of this section we present some well known attacks. We use the taxonomy given by Welch and Lathrop [WL03], in which they classify attacks according to the properties they attempt to violate. Attacks against confidentiality include passive eavesdropping, traffic analysis and wardriving. These are all examples of passive attacks, which are difficult to detect, since just by using a wireless network adapter, an attacker can eavesdrop on network traffic to capture information for analysis. For example, in *wireless wardriving*, someone with a computer equipped with wireless card and a GPS (Global Positioning System) drives through areas in searching of wireless networks. In this way, they can know not only the existence of a network, but also the physical location of wireless APs (Access Points). These are devices that are part of the basic wireless network infra-structure to control the client stations. *Traffic analysis* is a type of attack against confidentiality that can also be performed this way. The attacker can gain access to some

confidential information such as source and destination of packets, type of packets, among others.

Although passive attack may not be malicious, they can be used for an attacker to get enough information to launch active attacks. *Active eavesdropping* is an attack against confidentiality and integrity. Like in passive eavesdropping, the attacker obtains knowledge about unauthorized information. Besides, he or she can also modify packets; for example, by changing the IP (Internet Protocol) address of the destination to that of the attacker machine (*IP spoofing*). *Unauthorized access* is another kind of attack in which the attacker gains access to the network to use its services for free. Depending on the security mechanisms available, the attacker can either gain access to the wired network. *Man-in-the-Middle* (MITM) attacks also violate both confidentiality and data integrity. The attacker makes connections with the authorized participants and relay messages between them, acting as a proxy. Depending on the security mechanisms available, he or she can also establish a rogue AP, which serves as a proxy between the participant and the real wireless AP.

The so-called *Denial-of-Service* (DoS) attacks are attempts against availability that make the network unavailable for an indefinite period of time. This attack can take various forms [Gei04, ch.8]. The attacker can flood the network with packets that consumes the limited resources of a wireless network, forcing it to shut down. Attackers can also jam the wireless network operation using spurious radio frequencies to interfere with signal reception. Jamming can even occur accidentally, for example, by the presence of other devices that operates on the same frequency as the wireless network.

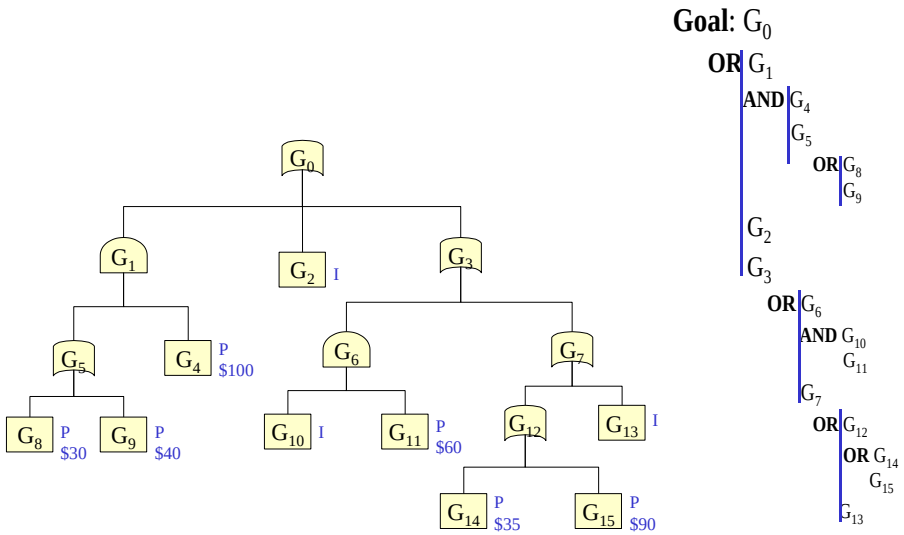
Of course, countermeasures exist to mitigate the risks from these attacks, such as the use of authentication (of both client and AP), message encryption, and others. It is the intent of this work to validate the security mechanisms employed, by injecting attacks into the communication system. The approach used to generate such attacks is presented in the next section.

2.4. Modeling attacks and vulnerabilities

Attacks can be modeled in many different ways, using formal models - as for example, Petri nets - or based on UML notations, such as use cases. We decided to use attack trees [Sch99] as they are suitable for our purposes for several reasons: (i) they focus on goals that can be transformed on attacks against the protocol implementation; (ii) they allow to describe, in a more structured way than natural language, the actions that should be performed for a successful attack; (iii) the model is easy to understand, even for practitioners with less skills in formal models; (iv) they allow a hierarchical representation, in which higher-level goals are broken down in sub-goals, until the desired refinement level is achieved; (v) it is possible to define attack patterns [MEL01] based on most common attacks to a given protocol; these patterns can be further reused in the validation of other implementations.

In attack trees, the root node represents the achievement of the ultimate goal of the attack. Each child node represents sub-goals that have to be accomplished for the parent goal to succeed. Parent nodes can be related to their children by an OR or an AND relationship. In an OR relationship, if any of the child nodes sub-goals are accomplished then the parent node is successful. With an AND relationship, all of the child node sub-goals must be accomplished for the parent node to be successful. The leaves of the tree, i.e.,

nodes that are no longer decomposed, represent attacker actions. Individual intrusion scenarios are generated by traversing the tree in a depth-first manner. Only leaves are kept in the scenario, as they represent actions that should be performed for an attack to succeed. In an analogy to test case generation, the objective is to cover all actions represented in the leaves. Attack trees can be represented graphically or textually. Figure 2 shows an example of a graphical notation that can be used, and the corresponding textual notation. On the bottom of the figure, the attack scenarios that can be generated for the tree are also shown. The notation <a, b, c> represents a scenario, with leaf nodes being considered in the order: a → b → c. So, for the tree shown, seven scenarios are possible.



Attack scenarios:

< G8, G4>, < G9, G4>, < G2>, < G10, G11>, < G13>, < G14>, < G15>

Figure 2. Attack tree (graphical and textual notation) and attack scenarios.

It is possible to associate attributes to the nodes, such as cost, probabilities or logical values. In this way, it is possible to select attack scenarios that are most likely or less costly, for example. In order to obtain such scenarios, it is necessary to propagate the attribute values from a leaf to the top of the tree [ERG+07]. Figure 2 presents annotations on the leaf nodes representing a boolean value (P-possible, I-impossible) and cost.

3. The proposed approach

Our approach for vulnerability detection (and subsequent removal) is based on the use of fault injection. This technique consists on the deliberate introduction of faults into a computer system to observe its behavior [AAA+90]. Fault injection can take various forms; here we use software-implemented fault injection (SWIFI), in which special software (the fault injector) is used to mimic the consequences not only of hardware failures but also of software faults, but we only consider the former. Faults are injected during runtime. Making an analogy with mutation testing [DGK+88], instead of mutating code to emulate software faults, we mutate system state, to emulate failure modes of the system environment.

The faults injected are based on the communication failure modes [CAS+85], which can be grouped in three classes: omission, arbitrary and timing failures. Omission

failures can be emulated by intercepting all messages coming from a specific host (crash failures) or only messages sent (send omission failures) or received (receive omission failures) by a host. Arbitrary (or Byzantine) failures can be emulated by corrupting messages received by the IUT. Timing failures, more specifically, performance failures can be emulated by delaying message delivery longer than specified.

Fault injection experiments have two inputs [AAA+90]: the *faultload*, representing the faults to inject, and the *workload*, which contains the inputs to activate system execution. This set can be generated using various techniques: by deriving inputs from a behavior model or by using a traffic generator, or even, real inputs, provided during real system operation. Our concern in this study is faultload generation. In order to define the faultload, we need first to define the objectives of fault injection in the case of vulnerability removal. Fault injection can be used either to emulate real attacks or to emulate accidental faults produced as a consequence of environmental failures (communication channel, hosts, among others), according to Figure 1. In this study we are interested in the first case. The faultload generation approach is then aimed to ask the question: *how to emulate real attacks using communication faults*?

Our solution is analogous to model-based testing, in the sense that the faultload is generated based on a model representing the attacks, more precisely, the attack trees, introduced in Section 2.4. The steps necessary to accomplish this activity are discussed in the reminder of this section.

3.1. Definition of attacker capabilities

In order to determine whether an attack is feasible or not, we need first to determine the attacker capabilities. In our approach the fault injector aims to emulate the attacker attempts to cause the system to fail. As in security analysis studies (e.g. [Low98, Gei04]), the attacker has complete control over the network. Hence, he or she can intercept messages, modify them, and deliver any message to any participant. In addition, since our attacker is implemented by a communication fault injector, he/she has also the possibility to delete, delay or replicate messages, as mentioned in Section 3.

The attacker cannot store the entire message he or she has received, but can store some message fields for further reference. Besides, the attacker knows the state of the participants, that is, he or she knows which messages a participant is expecting to receive, and what they will send after having received the message.

Another aspect that should be taken into account to determine attack feasibility is the actual test architecture. In other words, it depends on system controllability and observability the actual fault injector owns. Although some attacks are inside the scope of attacker capabilities, they could not be injected due to fault injector limitations. This separation is useful in that we distinguish the limitations of the validation technique from those of the implemented test architecture.

3.2. Identification of attacker goals

As stated in Section 3, we need a faultload that is representative of real faults, or, given the context of this study, it should be representative of real attacks. Then, to define attacks to be injected, knowledge about vulnerabilities as well as about well succeeded attacks is necessary. This information is available in many different sources about security (ex.:

NIST - National Vulnerability Database, SANS Institute), so what we have to do is to collect them and organize in a suitable way for the testing purposes.

After getting all attacks against the target protocol, they are categorized according to the properties they violate. This is useful in that it makes easier to determine the properties violated during the results analysis.

3.3. Construction of attack trees and selection of attack scenarios

Once the attack goals were identified, the attack tree can be constructed, as described in Section 2.4. The root represents the generic ultimate goal of attacking the target protocol implementation. The second level represent the properties that the attacker attempts to violate, according to the categorization made in the previous step. Then the sub-goals are inserted, corresponding to the attacks previously identified. The user can further refine some of the attack goals, according to his or her needs. Then, using the attacker capabilities identified in Section 3.1, it's associated to the leaves of the tree the logic attributes:

- `<possible_according_to_attacker_capabilities, possible_according_to_actual_testbed>.`

After that, the scenarios are selected so as to cover all leaves which satisfy those both attributes.

3.4. Refinement of the selected scenarios

Once the attack scenarios were identified, they generally should be refined. Since the attack tree express attack goals in the same abstract notation as in security requirements, we need to make refinements before implementing the scenarios. This step can be omitted if the tester is already satisfied with the attack scenario description. This step was only added to avoid the addition of unnecessary complexity to the attack tree. In this way, only scenarios that will be exercised during the validation should be refined.

3.5. Mapping attacks into faults

Attack scenarios adequately refined are still expressed in textual language. We need some specification language to represent them in a more specific format, that is, a notation that represents the steps to be performed by a generic fault injector. In this study we use active rules, proposed initially to represent active databases [PDW+93]. The reason is that these are simple, yet powerful paradigm to represent reactive behavior. Besides, when using runtime fault injection it is necessary to define a mechanism to trigger the injection of the faults. Active rules are of the form: <trigger, condition, action>. The semantic is the following: if the *trigger* event occurs and the *condition* is verified, then the *action* is executed. Of course, these elements are defined according to the fault injector capabilities. In our case, a *trigger* is an external event, such as the reception of a specific message or timer expiration. We are assuming that the fault injector can support that a *condition* may be established in terms of the message fields content, state variables and message flow [DJT+05].

The same occurs with the actions that can be performed by the fault injector (the attacker in our approach). Taking into consideration the attacker capabilities defined in Section 3.1, we can establish the actions that the injector can perform as: `intercept`, `corrupt`, `drop`, `replicate`, `delay` and `impersonate`. The latter means that the attacker can send a message to one legitimate participant as if he or she were another

legitimate participant. Other actions not related to fault injection need also to be defined, as for example, `store`, to store a message field temporarily. For fault injection actions we need also to define [AAA+90]: *fault location* that designates the element to be injected, e.g., an entire message or a field in a message and *periodicity*, which indicates whether the faults are permanent (applicable to all messages), transient (applicable to a single message) or intermittent (applicable to a set of messages).

4. The case study

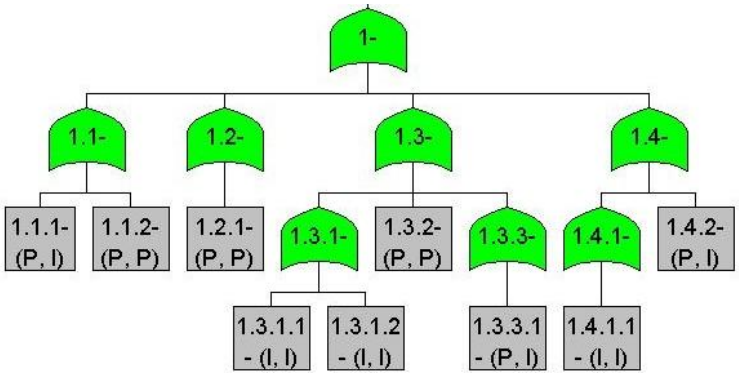
The case study is the security layer of the WAP (Wireless Application Protocol) protocol stack. The WAP protocol was chosen because in previous work were carried out robustness testing with the transaction protocol (WTP - Wireless Transaction Protocol) of the WAP gateway in presence of faults [CMM08]. Giving continuity to the work, we use the same WAP gateway implementation to evaluate the proposed approach for security validation in presence of faults. We use the WTLS (Wireless Transport Layer Security) layer of the WAP gateway that implements security functionalities offering *privacy*, data *integrity* and *authentication* for WAP applications [WAP01]. In this way, the WTLS provides security in the transactions between client (mobile terminal) and server (gateway) in wireless networks, allowing the use of WAP in applications such as mobile banking or e-commerce. WTLS is based upon Transaction Layer Security (TLS) protocol, which was developed as an internet standard version of the Secure Socket Layer (SSL).

The WTLS is divided in layers, having the Record Protocol (RP) as base [WTL01]. The RP establishes secure communications using the three way handshake with its four client protocols: first the Handshake Protocol opens a connection; second, the Change Cipher Spec Protocol decides about the cipher suite to be used; and third the Alert Protocol sends notification messages. Finally the Application Protocol is merely an interface for data transmission.

4.1. Attack modeling

Some tools are available to support attack tree construction and scenario selection. In this study we used the *SecureITree* tool [Ame08]. The attack tree was constructed from the WTLS security vulnerabilities at [Saa00]. Figure 3 shows the graphic and textual representation of the attack tree, structured as defined in Section 3.3. The root node represents the final objective of all of the attacks: to explore the WTLS's vulnerabilities. To achieve this objective an attacker should violate one of the protocol properties, as integrity, privacy, and authentication. Then each descending node of the root is decomposed in sub-trees or leaves that describe more specific attack goals or actions to achieve the objective represented by the node.

The attack goals were obtained, as recommended in Section 3.2, from information available about known WTLS attacks or vulnerabilities. For the sake of space, these attacks are not presented here. The reader should refer to [Saa00] for more details. The logic attributes values - possible (P) and impossible (I) - associated to the leaves are defined as explained in Section 3.3.



Legend:

(`possible_according_to_attacker_capabilities`, `possible_according_to_actual_testbed`)

P – Possible, I – Impossible

Goal: Attack the WAP server

OR 1- Attack WTLS

- **OR 1.1-** Attack integrity (truncation attack)
  - **OR 1.1.1-** Truncation attack in any message
  - **1.1.2-** Truncation attack in '*_close_notify' message
- **1.2-** Attack integrity (MAC protocol)
- **OR 1.2.1-** Exploit 40-bit XOR MAC (Message Authentication Code) weakness
- **1.3-** Attack privacy (key problems)
  - **OR 1.3.1-** Exploit key (brute force)
    - **OR 1.3.1.1-** Break 40bit DES (Data Encryption Standard) encryption (brute force)
    - **1.3.1.2-** Probable plaintext attacks (brute force)
  - **1.3.2-** Determine change of keys (plaintext leaks)
  - **1.3.3-** Predictable Initial Vector (IV)
    - **OR 1.3.3.1-** Determine initial IV (plaintext leak)
- **1.4-** Attack authentication
  - **OR 1.4.1-** Attack key exchange (handshake)
    - **OR 1.4.1.1-** RSA PKCS #1 (RSA Cryptography Standard) attack
  - **1.4.2-** Man in the middle attacks

Figure 3. WTLS attack tree (graphical and textual notation).

The attack scenarios were sorted according to leaves attributes values. All the attack scenarios having the value Possible for both attributes on the top of the list can be selected as they can be executed by the attacker and can be mapped to a fault scenario in the current test architecture. The selected attack scenarios are: <1.1.2>, <1.2.1>, <1.3.2>.

4.2. Attack scenario specification

After obtaining the scenarios, the tester should analyze whether they have enough information to be converted to a less informal notation. As example, below is the refinement of scenario <1.1.2>:

1.1.2- Truncation attack in '*_close_notify' message

AND 1.1.2.1- Check if message is '*_close_notify' type

1.1.2.2- Drop '*_close_notify' message

1.1.2.3- Drop last record of connection

The actions presented above are then converted to the trigger-condition-action notation, recommended in Section 3.5. The advantage of having this intermediate language to represent the attack scenarios is that the latter can be represented independently of the language used by a fault injection tool to represent fault scenarios.

1. (**1.1.2.1-** Check if message is *_close_notify type) and (**1.1.2.2-** Drop *_close_notify message)

- trigger = an UDP packet is sent from the client (mobile) to the server (WAP gateway).
- condition = (packet type byte is Alert Message) and (message type is 'connection_close_notify' or 'session_close_notify').
- action = drop message and set state variable (true value) to indicate next record truncation.

2. **1.1.2.3-** Drop last record of connection

- trigger = an UDP packet is sent from the client (mobile) to the server (WAP gateway).
- condition = state variable is true.
- action = drop record.

The rules represent an abstract fault scenario, i.e., they do not have parameters instantiated (such as location and periodicity) and are also platform independent. Further refinement is necessary to obtain the executable fault scenarios. An algorithm is being developed for the tool that is being used to inject the faults, Firmament [DJT+05].

5. Conclusions and Future works

In this paper we proposed a method for the generation of attack scenarios for security testing. Attacks are generated from a model, in the form of an attack tree, which represents known attacks or vulnerabilities of a protocol. The attack scenarios are performed by a communication fault injector. The proposed method shows how to select attack scenarios that are feasible for actual test architecture. We illustrate the method using as example a real world protocol. The attack tree was constructed based on available information about vulnerabilities of the target protocol.

The contribution of this work is that, as far as we know, this is the first time that an attack tree is used to derive scenarios for attack injection purposes. Besides, we proposed a notation to be used to represent these scenarios in a notation that is independent of the actual fault injector.

One difficulty with attack trees is to decide what level of refinement to use to define the attacker actions at the leaf nodes. The use of the generic notation to represent the scenarios can cope with this difficulty, as actions can be represented in this notation in the tree.

Among the future works we can mention: (i) how to decide the number of faults to inject for a given scenario, given that faults have parameters that can vary, in this way creating variations of a given attack scenario. The question is: how to define the number of variations to be created? (ii) how to detect whether an attack has been well succeeded or not. For that purpose, previous work on properties analysis [BCN+05, CMM08] is being envisaged.

References

[AAA+90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications," IEEE Transactions on Software Engineering, vol. 16, no. 2, pp. 166-182, February 1990.

[ALR+04] Avizienis, A. Laprie, J.-C. Randell, B. Landwehr, C. "Basic concepts and taxonomy of dependable and secure computing". IEEE Transactions on Dependable and Secure Computing, Volume: 1, Issue: 1, Jan.-March 2004, pp 11-33.

[Ame08] Amenaza Technologies Limited, SecureITree. Obtained in May/2008 at: http://www.amenaza.com/software.php.

[BCN+05] E. Bayse, A. Cavalli, M. Nunez and F. Zaidi, "A Passive Testing Approach based on Invariants: Application to the WAP", Computer Networks, 48, pp247-266, 2005.

[CAS+85] F. Cristian, H. Aghili, H. R. Strong, and D. Dolev. "Atomic broadcast: From simple message diffusion to Byzantine agreement", inProc. 15th Fault-Tolerant Comput. Symp., Ann Arbor, MI, 1985, pp. 200-205.

[CCD+00] C. Cachin, J. Camenisch, M. Dacier, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe, J.C. Laprie, J.C. Lebraud, D. Long, T. McCutcheon, J. Muller, F. Petzold, B. Pfitzmann, D. Powell, B. Randell, M. Schunter, V. Shoup, P. Verissimo, G. Trouessin, R.J. Stroud, M. Waidner, and I. Welch, "Malicious- and Accidental-Fault Tolerance in Internet Applications: Reference Model and Use Cases," LAAS report no. 00280, MAFTIA, Project IST-1999-11583, p. 113, Aug. 2000.

[CMM08] A. Cavalli, E. Martins, A. Morais. "Use of invariant properties to evaluate the results of fault-injection-based robustness testing of protocol implementations". Proc. of 4th Workshop on Advances in Model Based Testing (AMOST'08), in conjunction with 1st. International Conference on Software Testing, Verification and Validation (ICST), Lillehammer, Norway, 9-11/Abr/2008.

[DJM96] S. Dawson, F. Jahanian, and T. Mitton. "Orchestra: A fault injection environment for distributed systems". Proc. 26th International Symposium on Fault-Tolerant Computing (FTCS), pp 404-414, Sendai, Japan, June 1996.

[DGK+88] R. DeMillo, D. Guindi, K. King, M. M. McCracken, and J. Offutt. "An Extended Overview of the Mothra Software Testing Environment," Second Workshop

on Software Testing, Verification, and Analysis, pages 142-151, Banff, Canada, July 1988.

[DJT+05] R.J. Drebes, G. Jacques-Silva, J. F. da Trindade, T. S. Weber. “A Kernel-based Communication Fault Injector for Dependability Testing of Distributed Systems”. Parallel and Distributed Systems: Testing and Debugging (PADTAD-3), 2005, Haifa, Israel.

[ERG+07] K. S. Edge, R. A. Raines, R. O. Baldwin, M. A. Grimaila, and R. Bennington, "The Use of Attack and Protection Trees to Analyze Security for an Online Banking System," the Hawaii International Conference on System Sciences-HICSS-40, Kauai, Hawaii, January 2007, 8 pages (CD)

[Gei04] J. Geier, Wireless networks first steps. Cisco Press, 2004.

[GM95] James W. Gray III, John McLean: “Using temporal logic to specify and verify cryptographic protocols”. Proc. of 8th. IEEE Computer Security Foundations Workshop (CSFW '95), March 13-15, 1995, Kenmare, County Kerry, Ireland, pp 108-117.

[Low98] G. Lowe. "Towards a Completeness Result for Model Checking of Security Protocols", Proceedings of the 11th Computer Security Foundations Workshop (PCSFW), IEEE Computer Society Press, 1998.

[MEL01] A.P.Moore, R.J.Ellison, R.C.Linger. Attack Modeling for Information Security and Survivability. Technical Note CMU/SEI-2001-TN-001. March 2001.

[NAC+06] N. Neves, J. Antunes, M. Correia, P. Veríssimo, R.Neves. “Using Attack Injection to Discover New Vulnerabilities”. In: Proc. of the International Conference on Dependable Systems and Networks (DSN), 2006, pp 457-466.

[OAC07] J.M. Orset, B. Alcalde and A. Cavalli. “A formal approach for detecting attacks in ad hoc networks”. International Journal of Network Security, 2(2): 141-149, 2007.

[PDW+93] N.W. Paton, O. Diaz, M.H. Williams, J. Campin, A. Dinn, and A. Jaime. “Dimensions of Active Behaviour”. In N. W. Paton and M H. Williams, editors, Rules in Database Systems, Edinburgh 1993, pages 40--57, 1993. Obtained in June/2008 at: http://citeseer.ist.psu.edu/paton93dimensions.html

[PRO99] PROTOS - Security Testing of Protocol Implementations. Obtained in Mai/2008 at: http://www.ee.oulu.fi/research/ouspg/protos/.

[Saa00] M.J. Saarinen, M.J., “Attacks Against the WAP WTLS Protocol”, May 2000. Obtained in May/2008 at: http://www.cc.jyu.fi/~mjos/.

[Sch99] B. Schneier, B., “Attack Trees: Modeling Security Threats,” Dr. Dobb’s Journal, December 1999.

[TWM02] Herbert H. Thompson, James A. Whittaker, Florence E. Mottay. “Software security vulnerability testing in hostile environments”. ACM Symposium on Applied Computing (SAC) 2002: 260-264. Madrid, Spain.

[WW03] P.C.H.Wanner, R.F.Weber. “Fault Injection Tool for Network Security Evaluation”. LNCS Volume 2847/2003. Dependable Computing. Pp 127-136, Sept/2003.

[WAP01] WAP Forum, “WAP Architecture, Version 12-July-2001”. Obtained in May/2008 at: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html.

[WTL01] WAP Forum, “Wireless Transport Layer Security, Version 06-Apr-2001”. Obtained in May/2008 at: http://www.openmobilealliance.org/tech/affiliates/wap/ wapindex. html.

[WL03] D.Welch, S. Lathrop. “Wireless security threat taxonomy”. In Proc. Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society, Jun 18-20, 2003, pp76-83.