Planning the Attack! Or How to use AI in Security Testing?

Josip Bozic and **Franz Wotawa***
Institute for Software Technology
Graz University of Technology
A-8010 Graz, Austria
{jbozic,wotawa}@ist.tugraz.at

Abstract

Testing is one effective method for quality assurance. Generating and executing tests is a labor consuming task and there has been a lot of effort spent in test automation where the focus has been mainly on functional or penetration testing but not specifically on security testing. In this paper, we discuss two already introduced approaches for automated security testing that are based on AI planning. The approaches map attack models and security protocol definitions to AI planning problems in order to generate test cases. Furthermore, utilizing plan execution together with generated plans allows also for automating the test execution. The objective of the paper is to further stimulate research in this field. Thus we not only discuss the foundations behind and their applications, but also outline challenges and further research directions.

1 Introduction

Our modern society relies more and more on communication infrastructure. Its importance increases due to the availability of services like e-government, online banking, online shops, or social media applications that are used more or less on an everyday basis. All these applications including their communication backbone need to be reliable and secure preventing malicious entities from harming us, e.g., stealing money from our bank accounts or gaining access to our private data. Although, the security issue has gained a lot of attention over the past decades, our systems including infrastructure are still vulnerable against attacks. One reason behind is definitely the system's complexity comprising interacting heterogenous systems where it is very difficult to assure no vulnerabilities that an attacker can exploit during an attack. Vulnerability prevention – for example during programming – is, therefore, an important topic. [Hoglund and McGraw, 2004] give a very nice introduction into how attackers identify and make use of weaknesses in software in order to come up with an exploit. One challenge behind preventing from vulnerabilities is that an attack may utilize not only one weakness but a combination of shortcomings. Hence, identifying all vulnerabilities would require finding arbitrary sequences of interactions with a system that can be used for an exploit.

Focusing on finding new vulnerabilities is an important topic of security where the objective is to identify the vulnerability before an attacker can make use of it. This fight between the attacker and the system's defenders may never stop but it would still be an advantage to avoid known vulnerabilities to be in new systems. It is astonishing that this is not the case. When looking at the top ten list of OWASP[1] over the past 4 years, the first three attacks, i.e., injection, broken authentication and session management, and cross-site scripting (XSS), have not changed. Hence, it seems that it is necessary not only to focus on new vulnerabilities but to make sure that known ones are detected before a new software is deployed. This observation has been our main motivation behind our security testing research activities of the last 5 years, where we have been working on applying AI planning to security testing with the aim of automating security testing for avoiding known vulnerabilities in new system releases.

So, why relying on AI planning? When looking at attack description it is obvious that an attack comprises a sequence of steps to be carried out. For example, when performing an XSS over the web, an attacker has to access a web page, obtain the text fields, and afterwards, try to communicate source code to the server using these text fields. When introducing malicious code the attacker may try different program language fragments in order to identify the one that is used on side of the server. A sequence of such activities is obviously close to a plan, which comprises a set of actions each having pre-conditions and effects. This is very much similar to attack activities, where a certain activity can only be carried out when its pre-condition is fulfilled leading to its effect when being executed.

In this paper, we report on our work on planning for security testing. In particular, we show how AI planning can be used for modeling attacks thus allowing to automate security testing in detail. We further discuss two application areas. One is the domain of web applications and the other one is security protocol testing. Although, the main focus of the work is on finding known vulnerabilities it might be the case that AI planning is also capable of identifying unknown

*Authors are listed in alphabetical order.

[1] `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`, accessed: 2017-05-13

ones. This might be the case when specifying a lot of actions and using a planner generating arbitrary plans.

This paper is organized as follows. First, the applications and related research are discussed in Section 2 where we consider other research papers dealing with the use of AI planning for testing. In particular, we recall approaches dealing with pre- and post-conditions of methods to be used for test data generation, and using planning in the context of GUI testing. In Section 3 there is a discussion of planning and its use in security testing. Afterwards, the planning model is described in great detail in Section 4. The case study in Section 5 visualizes the approach on an example. Finally, we conclude the paper and discuss open challenges in Section 6.

2 Related Work

Planning, i.e, searching for actions that lead from a goal state to a final state, has been an active research field since the beginnings of AI. [Fikes and Nilsson, 1971] introduced the planning problem comprising an initial state, a goal state, and actions, and planning methodology STRIPS for generating sequences of actions for a planning problem. [Howe *et al.*, 1997] are one of the first that utilize the planning problem for testing. In their paper, the authors describe test case generation as a planning problem. The introduced the system Sleuth relies on the planner UCPOP 2.0. [Howe *et al.*, 1997] also explain the process of different plan generations according to changing states in the specification.

[Leitner and Bloem, 2005] introduce a planning-based testing strategy combining planning with a learning-based algorithm to generate plans at runtime. [Leitner and Bloem, 2005] suggested first, to extract a simplified model from a system under test (SUT). Afterwards, the planning problem is inferred from the generated model and a planner generates a weak solution, which represents a test case. The interpreter executes the test and the traversed state transitions are recorded. In case the execution leads to the specified goal state, the routine can be tested. Otherwise, the state transitions are included into the problem specification and a new plan is generated according to stronger pre-conditions. In this way, the resulting new test case is supposed to bring the test execution closer to success. The main difference to our approach is the absence of problem specification inference and the learning strategy. On the contrary, in both cases different plans are generated according to dynamic changes in the specification.

[Memon *et al.*, 2000b] and [Memon *et al.*, 2000a] introduce PATHS, an automated GUI testing system that relies on a planning. The paper puts emphasis on the test case generation part of the system. The specification of the planning problem is based on GUI applications, whereas the system relies upon the planner IPP [Koehler *et al.*, 1997], an extension of [Blum and Furst, 1997]. Here a planner generates multiple partial-order plans that are executed, eventually leading from an initial to the goal state. The test case generation process is divided into two phases, respectively the setup and the plan-generation phase. In the first part, PATHS makes an abstract model of the GUI and extracts GUI events, which represent abstract operators. Then the tester assigns corresponding pre- and post-conditions to these operators. Afterwards, in the plan generation phase, the initial and goal states are specified. Finally, PATHS generates the test cases that are executed on the concrete level. When an error is detected, the execution of the plan terminates. In this case, test oracles are used after each testing step. An important feature of this approach is the generation of alternative plans. This is done by creating sub-plans that represent a decomposition of an abstract operator from the high-level plan. The authors elaborate their algorithm for test case generation and present some experiments, which were conducted on WordPad. In addition to test case generation, PATHS also deals with test oracles and regression testing. The main difference to our approach, besides the application domain, is the generation of sub-plans, i.e. the generation of plans at two levels. On the other hand, the concept of using events for the action specification is done for protocol testing as well. But instead of using GUI events, we apply the TLS event structure from the protocol specification.

[Armando *et al.*, 2010] introduced an adaptation of planning to testing of security protocols. Other papers that deal with planning-based testing include [Galler *et al.*, 2010] and [Schnelte and Guldali, 2010]. [Beurdouche *et al.*, 2015b] introduce FlexTLS, a testing framework for TLS. Besides emulating the processing of TLS events, the implementation offers the possibility to manipulate the concrete parameter values and sequence of events. Some other works that address testing of protocols include, but are not restricted to [de Ruiter and Poll, 2015; Mavrogiannopoulos *et al.*, 2012; Morais *et al.*, 2009] and [AlFardan and Paterson, 2013].

3 Planning in Security Testing

The basic motivation behind the adaptation of planning is the fact that every interaction between a client and a system can be represented as sequence of actions. An interaction consists of multiple states, where each of the states is connected with another state when applying an action. Every action has some specific pre-condition that has to be fulfilled in order to be triggered. When triggered, the execution of this action leads into a new state. If this concept is put into the context of security testing, then every attack against a system can be seen as a sequence of actions as well. In the real world, an attacker has to carry out a set of malicious interactions with the victim system in order to cause some damage or retrieve data. Thus, the actions are specified formally in a way that they should resemble the individual attack steps undertaken by the attacker. Then, a planner generates a sequence of actions, i.e. a plan, which represents a full attack and, as such, a test case. In fact, this depiction acts as a blueprint for an attack. By automating the test case generation and execution, the attacker is emulated in an iterative manner. An initial state defines the starting point from where the attack is started. On the other side, the final state specifies the condition where an attack has been successfully carried out. The result of a test case can be either a FAIL or PASS. If an attack was successful, the tester can track the set of actions and the corresponding values that lead to the vulnerability detection. Afterwards, steps can be taken in order to cover the issue. We take the following un-

derlying definitions of the planning problem from [Bozic *et al.*, 2017]:

Definition 1 *The planning problem is a quadruple* (P, I, G, A) *where* P *denotes the set of predicates,* I *the initial state,* G *the goal state and* A *the set of actions. A predicate* $p \in P$ *presents a first order logic formula and acts as a condition in an action definition. An action* $a \in A$ *is defined by its parameters, set of pre-conditions and effects. The corresponding functions* $pre(a)$ *and* $post(a)$ *consist of predicates from* P*. The initial and goal states are defined with predicates that are true in these states.*

The plan generation starts from the initial state I that is given by some initial values. Then, the set of action is traversed in order to check whether a pre-conditions is satisfied by the initial configuration. In general, if some predicate of an action a in a state S is valid in $pre(a)$, then $post(a)$ will lead the execution into the next state S', i.e. $S \xrightarrow{a} S'$.

Definition 2 *A plan is a sequence of actions* $\langle a_1, \ldots, a_n \rangle$ *and presents a solution to a planning problem* (P, I, G, A) *so that* $I \xrightarrow{a_1} S_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} S_{n-1} \xrightarrow{a_n} G$.

From the implementation point of view, we assume specifying the planning problem using two files, namely the domain description file and the planning problem file. We also assume that the specification is done using the planning language PDDL [McDermott *et al.*, 1998]. The overall implementation can be divided into two parts: First, the type of the SUT is chosen, which in our case is either a web application or a security protocol. Then, according to the analysis of the structure of the system, an insight is obtained about its (mal-)functionality. Afterwards, descriptions of known system-related attacks are analyzed and an attack model is crafted according to inferred information. Whereas in some of the previous works (e.g. [Bozic and Wotawa, 2014b]) this is realized in form of UML state machines, we construct a planning model. Figure 1 depicts this procedure. This model depicts all necessary data definitions in order for the planner to generate a plan. It should be noted that the plan by itself represents only an abstract test case. Second, an execution framework reads the plan and executes it against a concrete SUT. During execution, concrete values are assigned to the parameters of the individual actions. This represents the concretization phase of the approach. Since the plan depicts a way to a successful security breach, if the execution reaches the goal state, it is assumed that a vulnerability has been detected. In case that an execution does not reach the final state, a new plan is generated with a different sequence of actions. Finally, the overall execution terminates after the last test case has been carried out.

4 Modeling the Planning Specification

When using planning for security testing, the construction and adaptation of the model depends heavily upon the type of the SUT and the attacks to be consider like SQL injection (SQLI) and cross-site scripting (XSS) in case of web applications and man in the middle attacks (MITM) in case

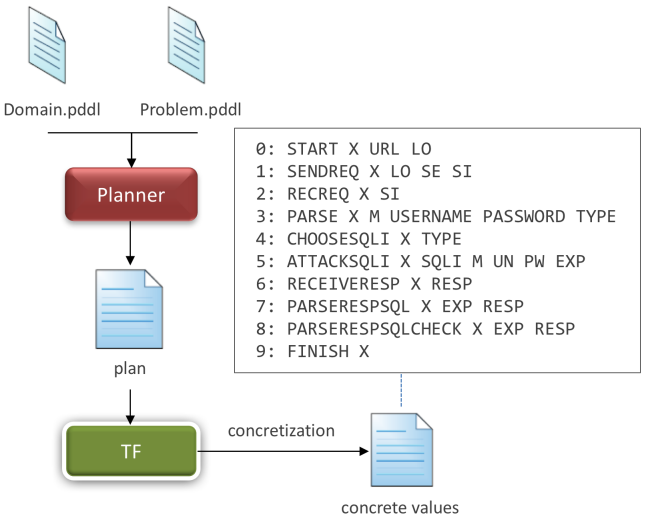

Figure 1: Plan generation

of security protocol SSL/TLS. Both applications will be explained below in greater detail. Although the resulting model differs in both case studies, the methodology is the same in both cases. This leads to the assumption that the proposed approach can be adapted to testing of other systems and attack types as well. [Bozic and Wotawa, 2014a] and [Bozic and Wotawa, 2015] explained the application of planning for security testing of websites, [Bozic *et al.*, 2017] introduced planning for protocol testing.

4.1 Web Applications

SQLI is for many years the most common injection attack on web applications (see Footnote 1). This attack targets the database behind a web application. So basically every implementation on the internet could be addressed. Despite sophisticated prevention mechanisms the attack still represents a major threat. Therefore, automating the testing process for the detection of this type of vulnerability still constitutes an important task.

First, let's take a look at a typical injection scenario. Usually an application sends SQL queries to the database, thus retrieving, deleting of manipulating its content. SQL by itself has a valid syntax that is submitted to the database. If the user gives the instruction to the implementation for accessing the data, where the number of possible access commands is usually restricted by the application. A simple query does look like:

```
SELECT * FROM workers WHERE password =
"+pField+"
```

This command would usually retrieve data from the table `workers` that matches the password. However, an attacker could trick the application to access the database even if s/he does not have a valid password. For example, s/he could insert the malicious SQL code into the input field:

```
x' OR 1 = 1;--
```

In this case, the executed query will look like:

```
SELECT * FROM workers WHERE password =
"x' OR 1 = 1;-- "
```

Since the statement `OR 1=1` will always make the query to be true, this would result in obtaining of potentially sensible data.

The injected malicious code represents a concrete value, which is meant for one test case.

As already mentioned, it's necessary to specify the planning problem via two different PDDL files. The domain encompasses data that is present in every problem. The necessary data includes, among others, the definition of types, predicates and actions. A reduced domain for a SQLI model is shown below.

```
(define (domain sqli-domain)
 (:requirements
    :strips :typing :fluents :adl)
 (:types
    active
    address
    status-lo
    status-si
    status-se)
 (:constants
    init - active
    no yes - status-lo
    sqli rxss sxss - type)
 (:predicates
    (inInitial ?x)
    (Empty ?url)
    (inAddressed ?x)
    (inSentReq ?x)
    (Logged ?lo)
    (statusinit ?si))
 (:functions
    (Logged ?lo - status-lo)
    (sent ?se - status-se)
    (statusinit ?si - status-si))
 (:action Start
    :parameters(
      ?x - active
      ?url - address
      ?lo - status-lo)
    :precondition (and
      (inInitial ?x)
      (not (Empty ?url)))
    :effect (and
      (inAddressed ?x)
      (not (inInitial ?x))
      (Logged yes)))
 (:action SendReq
    :parameters(
      ?x - active
      ?lo - status-lo
      ?se - status-se
      ?si - status-si
      ?lo - status-lo)
    :precondition (and
      (inAddressed ?x)
      (Logged yes))
    :effect (and
      (inSentReq ?x)
      (not (inAddressed ?x))
      (assign (sent ?se) 1)
      (statusinit two))))
```

Domain description for SQLI

As can be seen on this small sample, the individual actions and its parameters are modeled on the HTTP protocol between a client and the server. While the predicates indicate the current status of an object, functions are responsible for manipulation of data. For example, in order to start the testing process, an URL address needs to be given. An excerpt from the corresponding problem file reads as follows.

```
(define (problem sqli-problem)
 (:domain sqli-domain)
 (:objects
    x - active
    si - status-si
    lo - status-lo
    se - status-se
    url - address)
(:init
   (inInitial x)
   (Logged no)
   (not (statusinit two))
   (= (sent se) 0)
   (not (Empty url)))
(:goal
   (inFinal x)))
```

Problem description for SQLI

The problem file instantiates the objects of a type that has been defined in the domain. Here the initial and goal states can be found as well. Whereas the initial one defines the starting point, e.g. the idle state in execution, the final counterpart indicates a vulnerability breach according to the test oracle. The plan generation depends heavily upon these two definitions. If the initial values are changed, either manually by the tester or dynamically during execution, a different plan may be generated. Such a generated plan is depicted below.

```
0: START X URL LO
1: SENDREQ X LO SE SI
2: RECREQ X SI
3: PARSE X M USERNAME PASSWORD TYPE
4: CHOOSESQLI X TYPE
5: ATTACKSQLI X SQLI M UN PW EXP
6: RECEIVERESP X RESP
7: PARSERESPSQL X EXP RESP
8: PARSERESPSQLCHECK X EXP RESP
9: FINISH X
```

Plan for SQLI

If the final state cannot be reached by any arrangement of action, then no plan will be generated. In the proposed ap-

proach the planning system Metric-FF[2] is used, although the results can be obtained with other planners like Fast Downward [3] or LPG [4].

4.2 SSL/TLS Protocol

Security protocols play an important role in ensuring of a secure communication channel between client and server. One of such protocols is TLS and its predecessor, SSL. A private channel is established between the peers by applying a handshake procedure. Here encryption keys are exchanged, after which the data can be exchanged between both sides.

The current TLS standard [Dierks and Rescorla, 2008] defines the message flow and data structure of TLS events in a handshake procedure. There is a set of TLS event types with its own parameters that are exchanged between two peers. The procedure begins with the `ClientHello` message and finalizes with the server's `Finished` message.

The current version of TLS is version 1.2 but despite its security measures, the protocol still proves to be vulnerable to attacks. Only recently new vulnerabilities like Heartbleed [5] and DROWN [Aviram *et al.*, 2016] have been discovered. This leads to the conclusion that various TLS implementations, e.g. OpenSSL [6] or GnuTLS [7], still need to be tested against known attacks to check whether they are secure.

In order to specify protocol security testing as a planning problem, necessary information has to be obtained from the TLS standard. The proposed approach targets the handshake procedure, because many of the known security leaks are found there. Here individual TLS events are specified as actions in the planning problem. Here the example from [Bozic *et al.*, 2017] is taken, where the initial `ClientHello` message does usually have the following structure.

```
Protocol Version: TLS12
Client Random: 90 68 ...
Session ID:
Supported Cipher Suites: 00 30
Supported Compression Methods: 01
```

This message contains a set of variables, like the protocol version etc., with corresponding concrete values. In order to specify the problem in the planning way, the structure of the event is adapted to the object-type hierarchy of PDDL. The resulting action encompasses all of the event's original parameters, whereas the tester defines the pre- and postconditions. The resulting action is shown below.

```
(:action ClientHello
 :parameters(
   ?pv - ProtocolVersion
   ?crandom - ClientRandom
   ?sid - SessionID
   ?ccs - ClientCipherSuite
   ?ccm - ClientCompressionMethod
   ?x - active)
 :precondition (inInitial ?x)
 :effect (and (inClientHelloSent ?x)
         (not (inInitial ?x))) )
```

Basically, this mapping between TLS events and PDDL's action is assigned to the other messages as well. In this way, the plan execution may resemble the usual message flow as specified in the standard. An excerpt of the domain specification, which encompasses the client's initial message is given below.

```
(define (domain tls-domain)
 (:requirements
    :strips :typing :equality :adl)
 (:types
    active
    Priority
    ProtocolVersion
    ClientRandom
    SessionID
    ClientCipherSuite
    ClientCompressionMethod)
 (:constants
    rsa dh - KeyAlg)
 (:predicates
    (inInitial ?x)
    (inClientHelloSent ?x)
    (inClientHandshakeFinished ?x)
    (inFinal ?x)
    (KEY ?key))
 (:functions
    (KEY ?key - KeyAlg))
 (:action ClientHello
    :parameters(
      ?pv - ProtocolVersion
      ?crandom - ClientRandom
      ?sid - SessionID
      ?ccs - ClientCipherSuite
      ?ccm - ClientCompressionMethod
      ?x - active)
    :precondition (
      (inInitial ?x))
    :effect (and
      (inClientHelloSent ?x)
      (not (inInitial ?x)))))
```

Domain description for TLS

A piece of problem file with the definitions of objects and initial and goal states looks as follows.

```
(define (problem tls-problem)
 (:domain tls-domain)
 (:objects
    x - active
    pv - ProtocolVersion
    crandom - ClientRandom
```

[2] https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html, accessed: 2016-12-02

[3] http://www.fast-downward.org/, accessed: 2017-13-03

[4] http://lpg.unibs.it/lpg/, accessed: 2017-13-03

[5] see http://heartbleed.com/, accessed: 2016-12-06

[6] https://www.openssl.org/, accessed: 2017-05-14

[7] https://www.gnutls.org/, accessed: 2017-05-14

```
    ccs - ClientCipherSuite
    ccm - ClientCompressionMethod)
(:init
   (inInitial x)
   (= (CSYNsent s) 0)
   (ServerACK no)
   (KEY rsa))
(:goal
   (inFinal x)))
```

Problem description for TLS

Finally, a planner generates a plan. As already mentioned, the depicted sequence of actions represents an abstract test case. Usually the next step will be concrete value assignment although some vulnerabilities, e.g. SKIP [Beurdouche *et al.*, 2015a], can be triggered on the abstract level alone by relying only on default values.

```
0: CLIENTHELLO PV CRANDOM SID CCS CCM X
1: SERVERHELLO PV SRANDOM SID SCS SCM X
2: SERVERCERTIFICATE CLIST X
3: SERVERHELLODONE PMS YC X
4: CLIENTKEYEXCHANGERSA PMS X
5: CLIENTCHANGECIPHERSPEC SRANDOM
   CRANDOM CT X
6: CLIENTFINISHED FC X
7: SERVERCHANGECIPHERSPEC SRANDOM
   CRANDOM CT X
8: SERVERFINISHED FS X
```

Plan for TLS handshake

5 Case Study

Here a demonstration is given how to execute an attack according to its planning specification in case of the security protocol TLS. As a simple example the already SKIP attack is chosen. Attacks that target the protocol's vulnerability usually happen when the attacker intercepts the communication channel between the peers. In this way she or he acts as a man-in-the-middle and takes the role of a client towards the server and vice versa. The vulnerability happens when the handshake procedure concludes with establishing an encryption-free communication channel. This should not happen because TLS is meant to enforce an encryption on exchanged data. However, in this scenario no notification is triggered and it is assumed that the handshake has been finalized in an appropriate manner.

The standard event flow, as depicted in Section 4.2, demands that after the server sends a certificate, a finishing message is generated. However, in this attack some obligatory messages and parameters are skipped, i.e. the server signature is omitted and no encryption is established. Finally, if the client accepts the server's `Finished` message, data in an unencrypted form can be exchanged. Such situation offers an attacker the possibility to read data in plain text format. In order to generate such a test case, the planning specification has to be adapted.

Until now, the precondition of the sever's `Finished` message has been the previous sending of its `ChangeCipherSpec`message. This means that a server sends its final message only after the client has received its the mentioned message. So the precondition has been specified as:

```
:precondition(inServerCCS ?x)
```

But since some of the messages have to be skipped in order for SKIP to succeed, the precondition of the final server-side message is changed to:

```
:precondition(inServerCertificateSent ?x)
```

Then, the invoked plan will generate a small plan that looks like follows.

```
0: CLIENTHELLO PV CRANDOM SID CCS CCM X
1: SERVERHELLO PV SRANDOM SID SCS SCM X
2: SERVERCERTIFICATE CLIST X
3: SERVERFINISHED FS X
```

Plan for SKIP

The test oracle for this attack represents a check whether the client has received server's `Finished` message, albeit previously omitted events. If the testing framework is able to execute all actions without triggering some unintended notification, it is assumed that an unencrypted communication channel is established. As notices, concrete values do not need to be assigned for this attack.

Although this depicts a simple example, it demonstrates the connection between the planning specification, test case generation and execution. Planning problems can be defined for every attack. Also, by changing the definition, unintended sequences of actions could lead to exposition of security leaks. Thus, negative testing can be realized in the planning way as well.

6 Conclusion and Future Work

AI planning offers very certain means for representing attack models and security protocols and thus provides a very good bases for automating the security testing challenge. In particular, attack models can be very nicely represented as AI planning problem from which attacks can be obtained. In contrast to direct representation of attacks, for example, using a directed graph, the representation as AI planning problem may not only specify one attack but many. Every plan that can be obtained from the attack representation is itself a new attack. Hence, when combining different attack representations, we might also obtain new attacks that are combinations of already known attack scenarios. The same holds for security protocols where different kind of knowledge can be nicely integrated, e.g., knowledge about certain actions and actions that have to be performed because of the specification of the protocol.

In addition, to plan generation for obtaining abstract test cases, plan execution provides means for executing the security tests. For this purpose, the abstract plans have to be concretized. For example, instead of using a place holder of a user name or password, a real user name and password has to be used during execution. This mapping, however, can be

done easily and the whole concretization can be fully automated. One challenge that still remains to a certain extent is the test oracle problem. Identifying when a test case is failing and passing is not always that straight forward. Hence, further research has to be done for providing general solutions for the oracle problem in the context of security testing and AI planning.

Additional challenges are the granularity of actions and a more reactive plan execution. Granularity in this context means to identify the basic actions used to generate plans. Is it better to make an action login that submits user name and password to a server, or to set both of them separately before invoking a call? The granularity obviously depends on the application domain. However, when trying to integrate various attacks the actions must be reusable and easy to develop. Hence, there is a need for further investigating on the right representation of actions. Reactive plan execution is another weakness of the current implementation. In many cases a plan trying to exploit a known vulnerability cannot be executed. Whenever, the execution has to be terminated, it would sometimes be good not to start the whole planning and execution phase again but to restart at the current situation. A closer integration of plan generation and execution, maybe similar to teleo-reactive planning [Nilsson, 1994] might be appropriate.

References

[AlFardan and Paterson, 2013] N. J. AlFardan and K. G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013.

[Armando *et al.*, 2010] A. Armando, R. Carbone, L. Compagna, K. Li, and G. Pellegrino. Model-checking driven security testing of web-based application. In *Third International Conference on Software Testing, Verification and Validation Workshops ICSTW)*, pages 361–370, 2010.

[Aviram *et al.*, 2016] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Kasper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS Using SSLv2. In *Proceedings of 25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[Beurdouche *et al.*, 2015a] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015.

[Beurdouche *et al.*, 2015b] B. Beurdouche, A. Delignat-Lavaud, N. Kobeissi, A. Pironti, and K. Bhargavan. FlexTLS: A Tool for Testing TLS Implementations. In *9th USENIX Workshop on Offensive Technologies (WOOT'15)*, 2015.

[Blum and Furst, 1997] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997.

[Bozic and Wotawa, 2014a] J. Bozic and F. Wotawa. Plan It! Automated Security Testing Based on Planning. In *Proceedings of the 26th IFIP WG 6.1 International Conference (ICTSS'14)*, pages 48–62, September 2014.

[Bozic and Wotawa, 2014b] J. Bozic and F. Wotawa. Security testing based on attack patterns. In *Proceedings of the 5th International Workshop on Security Testing (SECTEST'14)*, 2014.

[Bozic and Wotawa, 2015] J. Bozic and F. Wotawa. PURITY: a Planning-based secURITY testing tool. In *2015 IEEE International Conference on Software Quality, Reliability and Security-Companion (QRS-C)*, pages 46–55, 2015.

[Bozic *et al.*, 2017] Josip Bozic, Kristoffer Kleine, Dimitris E. Simos, and Franz Wotawa. Planning-based security testing of the ssl/tls protocol. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2017.

[de Ruiter and Poll, 2015] J. de Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 193–206, 2015.

[Dierks and Rescorla, 2008] T. Dierks and E. Rescorla. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. `https://tools.ietf.org/html/rfc5246`, August 2008. Accessed: 2016-12-03.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Artificial Intelligence*, pages 189–208, 1971.

[Galler *et al.*, 2010] S. J. Galler, C. Zehentner, and F. Wotawa. AIana: An AI Planning System for Test Data Generation. In *1st Workshop on Testing Object-Oriented Software Systems*, pages 30–37, 2010.

[Hoglund and McGraw, 2004] Greg Hoglund and Gary McGraw. *Exploiting Software: How to Break Code*. Addison-Wesley, 2004. ISBN: 0-201-78695-8.

[Howe *et al.*, 1997] A. E. Howe, A. von Mayrhauser, and R. T. Mraz. Test case generation as an ai planning problem. In *Automated Software Engineering, 4*, pages 77–106, 1997.

[Koehler *et al.*, 1997] J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an adl subset. In *Lecture Notes in Computer Science*, 1997.

[Leitner and Bloem, 2005] A. Leitner and R. Bloem. Automatic Testing Through Planning. Technical report, Technische Universität Graz, Institute for Software Technology, 2005.

[Mavrogiannopoulos *et al.*, 2012] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A Cross-Protocol Attack on the TLS Protocol. In *ACM CCS 12: 19th Conference on Computer and Communications Security*, 2012.

[McDermott *et al.*, 1998] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld,

and D. Wilkins. PDDL - The Planning Domain Definition Language. In *The AIPS-98 Planning Competition Comitee*, 1998.

[Memon *et al.*, 2000a] A. M. Memon, M. E. Pollack, , and M. L. Soffa. A planning-based approach to gui testing. In *Proceedings of the 13th International Software / Internet Quality Week (QW'00)*, 2000.

[Memon *et al.*, 2000b] A. M. Memon, M. E. Pollack, and M. L. Soffa. Plan generation for gui testing. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pages 226–235, 2000.

[Morais *et al.*, 2009] A. Morais, E. Martins, A. Cavalli, and W. Jimenez. Security Protocol Testing Using Attack Trees. In *CSE (2), IEEE Computer Society (2009)*, pages 690–697, 2009.

[Nilsson, 1994] N. J. Nilsson. Teleo-reactive programs for agent control. In *Journal of Artificial Intelligence Research, 1*, pages 139–158, 1994.

[Schnelte and Guldali, 2010] M. Schnelte and B. Guldali. Test Case Generation for Visual Contracts Using AI Planning. In *INFORMATIK 2010, Beitraege der 40. Jahrestagung der Gesellschaft fuer Informatik e.V. (GI)*, pages 369–374, 2010.

[Williams and Wichers,] Jeff Williams and Dave Wichers. Owasp top ten project. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. Accessed: 2017-05-13.