Towards Automated Vulnerability Assessment

Saad Khan and Simon Parkinson
Department for Informatics, School of Computing and Engineering, University of Huddersfield, UK
Email: firstname.surname@hud.ac.uk

Abstract

Vulnerability assessment (VA) is a well established method for determining security weaknesses within a system. The VA process is heavily reliant on expert knowledge, something that is attributed to being in short supply. This paper explores the possibility of automating VA and demonstrates an initial proof-of-concept involving decision-making skills comparable with a human-expert. This is achieved through encoding a domain model to represent expert-like capabilities, and then using model-based VA planning to determine VA tasks. Although security evaluation is a complex task, through the help of such models, we can determine the ways to find potential vulnerabilities without an expert present. This technique allows time constrained assessments, where a 'risk factor' is also encoded to represent the significance of each security flaw. The ultimate goal of this work-in-progress is to realistically simulate a human vulnerability auditor. This paper demonstrates the first step towards that goal; a systematic transformation of the VA knowledge into a PDDL representation, accommodating a broad range of time constrained investigative actions. The output plan and its analysis evidently evinces many potential benefits such as increased feasibility and productivity.

Introduction

Security vulnerabilities exist in IT infrastructures within most organisations, and given the increasing size and importance of the infrastructure on a organisation's daily business, there is a pressing need to identify and mitigate security vulnerabilities. The organisation itself is responsible for protecting their IT resources against potential attacks, and this will often be performed through conducting periodic security assessments. However, an organisation may not always have the necessary expertise in-house and they will be required to pay for external consultancy. If an organisation does have in-house expertise to maintain their security, they are also required to maintain such expertise in this rapidly changing discipline. Both approaches incur a large financial cost and there is wide-scale motivation to decrease costs, as well as make an organisation more agile in that they are quicker to respond to detecting vulnerabilities as new threats develop. The general principle behind vulnerability assessment process can be summarised as (Kamongi et al. 2013):

1. Identify and taxonomise available system resources such as network and operating systems;
2. Prioritise resources or assets based on their importance level such as data sensitivity;
3. Determine threats to each resource and create potential point of vulnerabilities;
4. Based on the importance level, remove the most serious potential problems first and so on; and
5. Create a policy or guideline, that can minimise the consequences if an attack occurs in future.

A major bottleneck behind security assessments is the lack of knowledge and understanding of the latest potential threats. The adversaries are continuously becoming increasingly sophisticated in their attack mechanisms, and anyone who is not improving their security accordingly can become the victim of a damaging attack. Another important factor to consider is the inevitable human error during system configuration and use. In our previous work (Khan and Parkinson 2016), we created an expert system based on If-Then rules but the system produced mutually exclusive actions, was difficult to maintain and had very limited intelligence. A potential solution to these problems is through the use of computational intelligence to generate security evaluation plans in an automated manner. In general terms, the intelligent technique should consider all known evaluation techniques and determine their applicability to the system, and to identify mitigation plans for a complex system. The system should imitate and support the human expert's decision-making ability.

In this paper, we propose a system that applies Automated Planning (AP) to generate vulnerability assessment strategies for manual checking. It uses the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) for developing domain models and encoding problem instances. We use PDDL 2.2 for its support of durative actions, numeric fluents (Fox and Long 2003), and timed initial literals (Edelkamp and Hoffmann 2004). The domain model is essentially the description of knowledge, gathered from expert experience and published work detailing vulnerability assessment techniques. Each vulnerability is transformed into one or more actions. To demonstrate the suitability of the approach, we have also created a simple post-processor

that can automatically translate the output plan file to a human understandable format.

The paper is organised as follows: the first section provides a brief summary of vulnerability assessment and the applications of planning in cyber security systems. The next section is devoted to the detailed explanation and example results of the proposed solution. This leads to the experimental analysis section, whereby solutions are tested under different circumstances. This section also discusses the advantages of the proposed solution. Following this, the paper concludes and the direction of further work is provided.

Related work

Vulnerability assessment

Vulnerability assessment is the process of determining the security gaps of a system, which can be exploited by the attacker from inside or outside of organisation, to gain confidential data, financial benefits, amongst others (Umrao, Kaur, and Gupta 2012). Many generic vulnerability assessment tools are available that can identify known security flaws such as OpenVAS, Burp Suite, Nikto, Vega, AppScan, AVDS and Grabber (Owsap 2017). The purpose of these tools is to determine a system's security issues and stay ahead of attackers by constantly patching and mitigating identified vulnerabilities. Apart from the generic tools and approaches, some vulnerability detection techniques target specific applications. For example, (Benton, Camp, and Small 2013) performed a detailed security assessment for OpenFlow protocol, (Ristov, Gusev, and Donevski 2014) assessed the vulnerabilities of OpenStack's architectural components, (Zhao and Zhao 2015) analysed privacy and security issues of social media sites, (Barrere, Badonnel, and Festor 2014) identified and explained the vulnerabilities of autonomic systems and (Rahman, Ahmad, and Ramli 2014) discussed potential Wireless Body Area Network security vulnerabilities. Many patents also present different vulnerability assessment techniques, e.g. (Webb, Boscolo, and Gilde 2016) created a network appliance that can evaluate security of multiple networks concurrently. These specific evaluations are limited in use but provide deep insight into a particular product.

One of the major shortcomings of aforementioned approaches is that they require extensive knowledge for running and understanding the results. It is very difficult for a non-expert to conduct the security evaluation without first spending significant time to acquire the necessary expertise. Furthermore, most of the approaches do not consider time limits, prioritisation and real-time damages associated with the vulnerabilities. The damages might cause exposure of sensitive data, unavailability of crucial services and many others. These factors motivate the requirement for an automated system, which can decide and prioritise vulnerabilities based on time constraints and the potential for damage, and outputs the most feasible solution without relying on a human expert.

Applications of Planning in Cyber-Security

There have been successful exploration of the use of Automated Planning (AP) in different cyber security domains, mainly for generating attack plans for penetration testing (Riabov et al. 2016). In this work, courses of actions are generated based upon a system configuration; however, the goal is adversarial in that the aim is to compromise the system in efficient shortest path, albeit by a trusted security professional (widely termed white-hat hacking). Recent work by Sohrabi et. al pursues the use of hypothesis exploration for identifying potential attack plans in network security (Sohrabi, Udrea, and Riabov 2013; Sohrabi et al. 2016). Furthermore, recent research presents continued development of AP for penetration testing (*pen testing*) (Shmaryahu 2016a; Hoffmann 2015) discussing the need to overcome scalability limitations. The fundamental difference between pen testing and vulnerability assessment (VA) is that VA is searching for vulnerabilities that exist and mitigate them, where as pen testing is searching to exploit a series of vulnerabilities for adversarial gain.

Penetration testing frameworks are available, both commercial and open-source, which can perform expert-like security assessment through simulated attacks on different systems. One such example is *Metasploit* that can launch exploits and drop payloads to damage remote systems (Maynor 2011). The problem with such frameworks is that expert knowledge is required to manually select and launch the attacks. Although, some security weaknesses such as unpatched software and insecure ports can be identified by vulnerability scanning tools, but their results might not be comprehensive (Holm et al. 2011). Studies suggests that, if the attack plans are generated by a computer, there is potential to discover more plans than human expert, meanwhile helping the non-expert to avoid the complexity and save time, effort and resources. One such commercial tool (*Core Impact*) and uses AP to generate possible attack plans and performs real-time penetration testing (Sarraute, Richarte, and Lucángeli Obes 2011). It uses Probabilistic PDDL (PPDDL), which is capable of extending attack graphs models and handling probabilistic and numerical effects. The system also constructs AND-OR trees to determine candidate attacks paths towards a particular asset. The tool is also efficient in terms of execution time and in the generated network traffic. It's computational complexity is $O(n \log n)$, where n is the total number of actions in domain file. Similar work has been done by (Shmaryahu 2016b), where contingent plan trees are constructed for simulated pen testing.

One initial piece of work involved the use of classical planning to generate hypothetical attack scenarios to exploit the system (Boddy et al. 2005). The study simulates realistic adversary courses of action and mainly focuses on malicious insider's threat. The domain model includes 25 different objects (basic elements of computing), 124 predicates (information about system) and 56 actions (adversarys objectives), whereas each problem contains between 200 to 300 facts. Classical and forward heuristic planners, specifically FF-Metric (Hoffmann 2003) are used to generate attack plans. As writing domain models manually can be labour in-

tensive and prone to errors, M4 macros have been used to design large scale PDDL files, hence avoiding the need for representing actions and facts directly. Their tool also translates the planner output into human-readable format (post-processing) by using a Perl script.

Another paper (Obes, Sarraute, and Richarte 2013) uses planning to assess network security. First, a transformation algorithm is used to convert attack models into PDDL representation. Attack information containing requirements and exploits are encoded into a domain file, while the information about system such as networks, machines, operating systems, ports and running services are stored in problem files. The object types are the system properties such as privileges and operating systems, while predicates are essentially depicts the relationship among objects. This paper analyses the whole network, has up-to 1800 actions and hosts 700 exploits. However, as classical planning is used, the system cannot handle incomplete knowledge.

Partially Observable Markov Decision Processes (POMDP), can be used to overcome the limitation on incomplete knowledge by generating attack plans even if the planner is given incomplete knowledge and uncertainties (Sarraute, Buffet, and Hoffmann 2013a). POMDPs are capable of prioritising actions based on expected reward that is composed of asset value, time and risk of detection, to find the optimal terminal state. Further research (Sarraute, Buffet, and Hoffmann 2013b) investigates how to produce better attack plans for a particular machine within short period of time. Their solution employs intelligent vulnerability scanning actions through using POMDPs to find feasible attacks for each individual machine and inquires targeted network structure approximations on-demand. Despite all the advantages of POMDPs, they are complex and require large computational resources. It is also difficult to design the 'initial belief' for every real-world problem. As a solution, (Hoffmann 2015) presents a middle ground between classical planning and POMDPs called MDPs. The actions work same as before, but they do not perform scanning. Every outcome of action (effect) is assigned a probability regardless of host configuration predicates. The probability value depends on the level of attackers uncertainty in launching that particular action. As PDDL is not equipped to tackle these uncertainties, the paper also suggests a PDDL-like language that can allow probability values inside action.

Literature review shows that planning has been applied to automate pen testing, but to best of our knowledge, there is no such work in automating VA. According to the survey (Shah and Mehtre 2015), VA and pen testing are different in term of motivations and objectives. VA is applied to a system which is likely to have vulnerabilities, unlike penetration testing where system defences are tested. The penetration testing has specific goals and requires particular expertise, whereas VA finds and prioritises the assessment of system vulnerabilities. As the VA is first step towards maturing the security state of the entire system and focuses on both breadth over depth of analysis, we recognise the need for automation and we provide a feasible and resourceful solution.

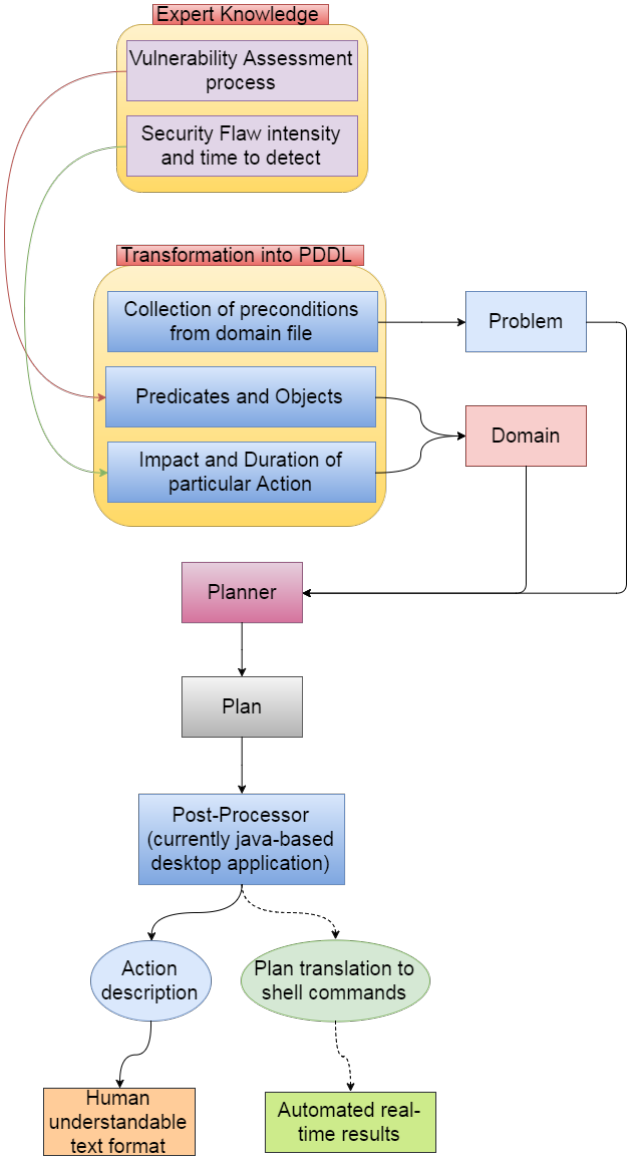

Figure 1: Architecture of proposed system

Automated Planning as a Solution

The purpose of our solution is to help the non-expert users to determine efficient and time-constrained VA checks and perform them manually. The solution is not aimed at replacing the human-expert, but rather in providing decision support aid to users of all technical abilities. The following sections contain a detailed description of the system design, domain modelling, problem description and a sample plan output.

System Design

This section discusses the overall system design and its components. Our proposed solution is inspired by (Sarraute, Richarte, and Lucángeli Obes 2011) as is shown in Figure 1. The explanation of each module is in the following:

Pre Vulnerability Assessment (VA) process – The purpose of VA is to systematically evaluate the security of any given

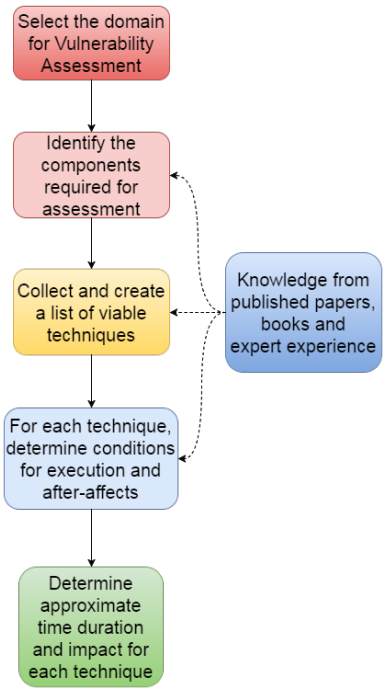

Figure 2: Pre-Vulnerability-Assessment process

system. The VA process should comply to well-established criteria and standards. For now, our solutions is mainly focused on *Authentication*, *Authorisation* and *Permission control* along with few data security assessment techniques. To develop a system that advises on relevant VA procedures, we need to manually collect authentic, expert and verified knowledge on existing VA techniques, e.g. using acquisition software such as (Parkinson and Crampton 2016). The formal preprocess of VA is described in figure 2. It shows the tasks that were conducted before creating the solution. The VA procedures consists of strict steps, which should be followed in the same manner and sequence, hence the need for systematic knowledge acquisition. The first step is to identify the domain whose vulnerabilities are going to be assessed. Then, with the help of expert knowledge and published work, relevant, applicable and useful VA techniques should be extracted manually, along with their preconditions and effects. These conditions and effects are modelled into PDDL later on.

Representing domain knowledge – After information is gathered regarding VA procedures, it is manually converted into predicates and objects, which signify the properties and relationships of individual VA procedures. The group of inter-related predicates form an action, whereas each VA process is defined by one or more actions. For each action, we are also estimating and modelling their durations of identification and potential damage level in case that vulnerability is exploited. The quantification of impact and duration values are then used to provide the most feasible results in accordance with given user-requirements. The user can input deadline value and output plans will inform about the most crucial VA procedures that should be conducted within

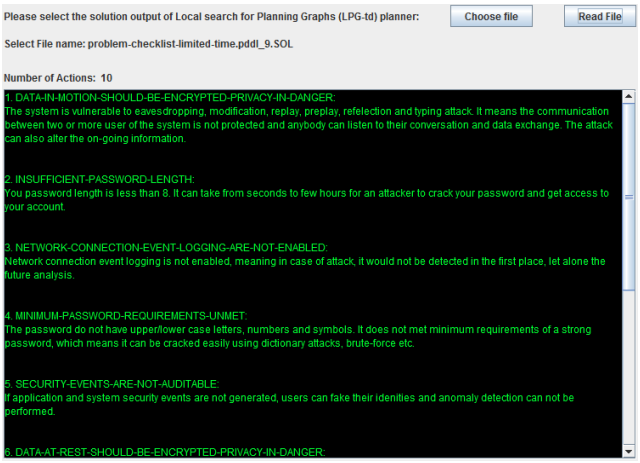

Figure 3: Plan's Post Processor

that limited time.

Problem description – The initial state is a collection of all preconditions from domain file which describe the underlying system. The goal state is empty except from the time duration limit. This is because the solutions aims to find the most feasible plan for VA within given a deadline, whilst maximising the overall impact. The system has static knowledge, which means it will always output the same set of actions. The only variation in plans is brought by imposing time constraints in each action, where the planner will choose actions based on their impact on the optimisation metric.

Planner – LPG-td (Local search for Planning Graphs) planner (Gerevini et al. 2004) is used to extract plans from domain and problem description files. LPG-td, an improved version of LPG, supports durative-actions and plan adaptation (as the goal state is not explicitly described) problems. We used LPG in this initial work due to its general good performance and handling of PDDL features.

Plan – Consists of actions, which represent the actual vulnerability. For example, *(SYSTEM-VULNERABLE-TO-DENIAL-OF-SERVICE-ATTACK SYSTEM)* is a single action of certain plan. It describes that the system is susceptible to denial of service attack. The plan would continue to have a sequence of actions, used to describe mitigation actions to pro-actively eradicate vulnerabilities, minimise threats and prevent future attacks.

Post-processing – The post processor is a Java-based application, whose only purpose is to elaborate the plan in a more human understandable format (shown in figure 3). It contains a simple mapping of actions to their respective descriptions. In future, we aim to enhance the preprocessor in a way, that can convert the actions into appropriate and complete shell commands, which can be executed to perform real-time automated VA operations.

Domain modelling

The domain file consists of 23 durative-actions. Each action contains parameters, duration, conditions and effects. The parameters define objects required for the action to work.

```
(:durative-action Minimum-password-requirements-unmet
  :parameters (?password)
  :duration (= ?duration 5)
  :condition (and
    (over all (No-Uppercase-Letters ?password))
    (over all (No-Lowercase-Letters ?password))
    (over all (No-Numbers ?password))
    (over all (No-Symbols ?password))
    (over all (not
    (Minimum-password-requirements-unmet-found ?password)))
    (over all (action-durations))
    (at start (not (operator-busy) ) )
  )
  :effect (and
    (at end
    (Minimum-password-requirements-unmet-found ?password))
    (at end (increase (total-impact) 8))
    (at end (increase (total-duration) 5))
    (at start (operator-busy))
    (at end (not (operator-busy) ) )
  )
)
```

Figure 4: Example PDDL action

The duration is the approximate time required to assess a particular vulnerability and depends on its complexity level. The conditions are composed of issues that needs to be true for the vulnerability to exist. There are also fixed predicates in each durative-action, called *action-durations* and *operator-busy*, which are used to sum up the duration of each action and to ensure they execute sequentially. The effect explains the damage of a particular vulnerability. It also defines the level of damage in terms of impact value, which is between 1 and 10, where 1 being minimal and 10 being the largest damage.

Table 1 shows the list of all actions, durations and impact levels, extracted from the expert knowledge. The duration of vulnerability assessment action might vary for different systems. The impact level too depends on the sensitivity of data, services and resources of the underlying system.

An example of durative-action is shown in Figure 4, where the purpose is to check if the system has weak password and might be vulnerable to password-cracking attacks. The complete explanation is in the following:

Predicates – The predicates are extracted and derived from the requirements of the vulnerability assessment. The information regarding vulnerabilities is collected from various sources such as books, papers, web articles and expert knowledge. A single vulnerability is represented by one or more predicates. In addition, there are two more predicates, *action-durations* and *operator-busy*. The *action-durations* is used to define a deadline for all actions in the output plan, where the time-limit is given by the user. The *operator-busy* is used to ensure actions are performed sequentially. The total-time of a plan should be less than or equal to *action-durations*, which is used as a timed initial literal to restrict the makespan.

Functions – There are two functions in the domain description: *total-impact* and *total-duration*. The *total-impact* is used to determine the accumulative impact value of all actions in the output. Its value is increased by the impact value of each action. Similarly, *total-duration* is the accumulation of each individual action's duration. By using these functions, the planned is able to select actions that have the great-

Table 1: Transformation of knowledge into Actions, their duration and impact level

#	Action name	Duration	Impact (total 10)
1	Insufficient-password-length	2	8
2	Minimum-password-requirements-unmet	5	8
3	Password-is-guessable	7	5
4	Insecure-password-storage	15	6
5	Password-brute-force-attack	10	4
6	Insecure-forgot-password-option	20	4
7	Insecure-single-factor-authentication	8	5
8	Infeasible-authentication-scheme	20	8
9	Access-vulnerability-File-system	25	7
10	Access-Control-Authorisation-vulnerability	30	5
11	Unmanaged-permission-of-application	10	7
12	Applications-might-be-outdated	15	9
13	Lack-of-network-firewalls	10	8
14	Lack-of-maintenance-in-network-firewalls	7	6
15	Default-Configuration-network-firewall-might-be-useless	10	5
16	Each-Node-Should-Have-Personal-firewall	9	4
17	System-Vulnerable-to-Denial-of-service-attack	5	8
18	Network-connection-events-logging-are-not-enabled	8	5
19	Access-control-events-are-not-auditable	10	6
20	Security-events-are-not-auditing	5	7
21	Data-at-rest-should-be-Encrypted-Privacy-in-danger	5	9
22	Data-at-motion-should-be-Encrypted-Privacy-in-danger	5	9
23	Feasible-encryption-algorithm-not-used	8	7
	Total	**249**	**150**

```
(:init
  (No-Uppercase-Letters password)
  (No-Lowercase-Letters password)
  (No-Numbers password)
  (No-Symbols password)
  <...>
  (= (total-impact) 0)
  (= (total-duration) 0)
  (at 0 (action-durations))
  (at 10 (not (action-durations)))
)
(:goal (and
  (<= (total-duration) 10)
))
(:metric
  maximise (total-impact)
)
```

Figure 5: Example PDDL problem definition

est impact within the available time frame specified through the Timed Initial Literal.

Durative-action – The purpose of using *temporal* actions is to model the time needed to execute a vulnerability assessment action. The action presented in Figure 4 illustrates that a vulnerable password would lack upper and lower-case letters, numbers or symbols. The condition also contains the negation of *effect* because our problem file does not have any explicit goal. Without this, we would have duplicate actions in the plan. The *action-durations* and *operator-busy* are used to limit the number of actions and remove their concurrency respectively.

Effect – Figure 4 shows that any password having the aforementioned issues does not meet the minimum requirements of a strong password, hence it is viable to password-cracking attacks. It states the impact level of the vulnerability as well as identifying the possible amount of damage it can cause, which is 8 out of 10 in this particular case. It also increases the accumulative impact and duration value (*total-impact* and *total-duration*). In adition, the effect starts by stating the *operator-busy* predicate, so that no other action can be executed at that instance. Once the action is completed (at end), *operator-busy* is reverted to its original state, freeing the lock and the planner proceeds to next action.

Problem Description

A sample part of problem file is also shown in figure 5. It contains the initial state, goal state and metric descriptions. The complete explanation on the construction of problem file is in the following:

Init – Describes the complete initial state of system in terms of properties such as the password has no lower or upper case letters and many others. The initial state is a collection of all predicates the represent the system under examination that can subsequently be used for vulnerability assessment. The objects represent the constant names of assets under assessment. For example, *password* is an object, whose strength level is rated in the aforementioned domain description.

```
0.0003: (FEASIBLE-ENCRYPTION-ALGORITHM-NOT-USED
                        ENCRYPTIONALGORITHM)

8.0005: (DATA-IN-MOTION-SHOULD-BE-ENCRYPTED-PRIVACY-IN-DANGER
                        DATAINMOTION)

13.0007: (DATA-AT-REST-SHOULD-BE-ENCRYPTED-PRIVACY-IN-DANGER
                        DATAATREST)

18.0010: (SYSTEM-VULNERABLE-TO-DENIAL-OF-SERVICE-ATTACK
                        SYSTEM)

23.0012: (MINIMUM-PASSWORD-REQUIREMENTS-UNMET
                        PASSWORD)
```

Figure 6: Example PDDL plan output

Goal – Notice there is no explicit goal to reach as we want all of those actions, which should be completed within the given deadline. Thus, we only check whether the accumulative value of duration, *total-duration*, is less or equal to deadline-value (which is 10 in this case). Once the condition is satisfied, the planner should stop generating the plan, even if there can be more actions.

Metric – The requirement for our system is to output those actions, which have the maximum impact and can be completed within deadline. For maximising the impact, we have used *maximize* feature on *total-impact* function. It will enable the planner to only select the most feasible action with respect to their impact value, if various options become available within limited time.

Planer output

A sample plan is shown in figure 6. It took *1 second* to search this plan on *Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz* of processor with *16GB RAM*. The operating system was *32-bit Ubuntu Kylin*. The deadline was specified as 30 minutes and the planner output provides a plan with the total duration of *28* minutes. The plan shows VA tasks that should be conducted to evaluate the security. The planner was executed several times *(-n 10)* to reach a plan better quality. If the plan output file is provided to the post processor (figure 3), it can elaborate the plan in further details. The post processor is a primitive application which matches the actions against pre-determined sentences to provide more detail instructions with context for the user.

Experimental Results and Analysis

This section presents the results of proposed solution and evaluate them to demonstrate the advantages in terms of usability and applicability of this technique. Using the same domain file, we provided the planner with various problem files, all of them having different time constrained deadlines, ranging from 10 to 250 minutes in duration. For each deadline, the planner was executed more than once, i.e. until it could not find any better plan within the specified 5 minutes of cut-off time. The results of various plans are shown in Figure 7. The x-axis shows the total number of actions, while the y-axis shows the deadlines and accumulated impact value of each plan. Furthermore, planner is displaying the correct results as the output matched our expected output that was derived manually.

The number and impact value of the actions are directly proportional to the deadline. This feature significantly maximises the efficiency of VA process, as more important vulnerabilities can be identified within specific amount of time. Furthermore, the total value of duration in domain file is 249 and the total number of actions are 23 (shown in Table 1). With the deadline of 250 minutes, which is more amount than the sum of all durations of actions, the planner outputs all 23 actions as expected. It means that the full potential of solution can be used if user has enough time. It is also observed that some different deadlines (e.g. of 60 and 70 minutes) present the same amount of actions (10), but with the different impact values (72 and 76 respectively). This proves that our resultant plan will try to maximise the overall impact, while choosing only crucial and minimum number of actions. Hence, the user will be able to detect important vulnerabilities in a shorter time span and protect the system against common, yet harmful attacks.

Potential advantages

Our solution is beneficial for both experts and non-experts. It should be noticed that the solution is not supposed to replace human experts, but assist them in a useful, resourceful and practical way. Following are the benefits that our solution can provide in terms of vulnerability assessment.

Cost – One company charge 1495-USD for single vulnerability assessment of an IT infrastructure with up-to 100 individual internal Internet Protocol (IP) addresses or nodes and takes minimum of two-weeks. But with our solution, any company or individual can get the vulnerability assessment free of charge, within a significantly lower timeframe.

Less effort and more productivity – As the planner automatically outputs the VA tasks, there is no effort required to conduct tiresome searching to find suitable techniques and results in reduction of time, without compromising the quality. Also, the plan itself allocates an appropriate amount of time for each assessment task, hence the whole VA process becomes systematic, precise and efficient.

Quality and effectiveness – The quality of solution depends on the quality of knowledge in domain model. The knowledge of our solution is collected from renowned research outlets and experts. So, the solution is capable of mimicking human expert abilities, hence making it somewhat equally effective.

Feasibility – There are some cases where a company does not want to utilise 3rd party contractors or outsourced vulnerability assessment operations. It is possibly that due to lack of access to experts, the company are paranoid of exposing their private data and system configuration. Using this solution, one can perform in-house VA processes on-demand.

Decide custom time frame – Generally, the VA process is performed on monthly, quarterly, semi-annually or annually basis. But, with our solution at hand, there is no restriction of predefined schedule. Furthermore, one can decide their own custom time-frame and obtain the list of VA tasks in a specified time window.

Scope of domain model – The domain models are manually defined and based on human knowledge. Although this paper focuses on a specific aspect of cyber security, the domain model can describe any number of techniques and methodologies from multiple areas simultaneously. It is just a matter of modelling the knowledge into domain file. Hence, it would not be wrong to state that our proposed system can comprehend the knowledge of multiple human experts inside a single domain model and provide a better and holistic plan, as well as strategy by considering various areas of cyber security.

Figure 7: Relationship between different deadlines, and their impact and number of actions. It shows that (deadline durations $\propto$ number of actions & accumulative impact).

Conclusion and Future work

We have shown in this paper that vulnerability assessment techniques can be modelled into planning problem, where they are solved more efficiently along with integrating new functionality such as time constraints. The proposed solution, though currently work-in-progress, has shown significant initial results in aiding the non-expert users to conduct vulnerability assessment tasks own their own. This paper discussed the complete design of proposed solution and the details on how to transform the expert knowledge to the planning domain. The results successfully depicts that it is not necessary for a non-expert to rely on others. The plan itself can inform the user in manually performing comprehensive VA process. The domain model is created from expert knowledge and hence the VA procedures mimic the abilities of an expert as well. By using our domain model, planners such as *LPG-td* and user deadline requirements, optimal plans can be generated based on their threat level and time duration.

As our final goal is to produce a real-time automated VA solution, the following important questions remains as regards to future work. First, how to increase the quality and quantity of knowledge in domain models? It essentially leads towards incorporating better knowledge acquisition techniques from the experts. Second, how to deal with incomplete knowledge? One possible solution would be using probabilistic planning techniques that can generate discrete actions. Last but not least, how this solution can become more beneficial and usable? It can be done by transforming the plan into executable commands in accordance with the underlying system.

References

Barrere, M.; Badonnel, R.; and Festor, O. 2014. Vulnerability assessment in autonomic networks and services: a survey. *IEEE communications surveys & tutorials* 16(2):988–1004.

Benton, K.; Camp, L. J.; and Small, C. 2013. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 151–152. ACM.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21.

Edelkamp, S., and Hoffmann, J. 2004. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC04), at ICAPS04*.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. Lpg-td: a fully automated planner for pddl2. 2 domains. In *In Proc. of the 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04) International Planning Competition abstracts*. Citeseer.

Hoffmann, J. 2003. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Hoffmann, J. 2015. Simulated penetration testing: From" dijkstra" to" turing test++". In *ICAPS*, 364–372.

Holm, H.; Sommestad, T.; Almroth, J.; and Persson, M. 2011. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security* 19(4):231–247.

Kamongi, P.; Kotikela, S.; Kavi, K.; Gomathisankaran, M.; and Singhal, A. 2013. Vulcan: Vulnerability assessment framework for cloud computing. In *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on*, 218–226. IEEE.

Khan, S., and Parkinson, S. 2016. Towards a multi-tiered knowledge-based system for autonomous cloud security auditing. In *Proceedings of the AAAI-17 Workshop on Artificial Intelligence for Cyber Security (AICS)*. AAAI.

Maynor, D. 2011. *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2013. Attack planning in the real world. *arXiv preprint arXiv:1306.4044*.

Owsap. 2017. Vulnerability scanning tools.

Parkinson, S., and Crampton, A. 2016. Identification of irregularities and allocation suggestion of relative file system permissions. *Journal of Information Security and Applications* 30:27–39.

Rahman, A. F. A.; Ahmad, R.; and Ramli, S. N. 2014. Forensics readiness for wireless body area network (wban) system. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, 177–180. IEEE.

Riabov, A.; Sohrabi, S.; Udrea, O.; and Hassanzadeh, O. 2016. Efficient high quality plan exploration for network security. In *International Scheduling and Planning Applications woRKshop (SPARK)*.

Ristov, S.; Gusev, M.; and Donevski, A. 2014. Security vulnerability assessment of openstack cloud. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2014 Sixth International Conference on*, 95–100. IEEE.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2013a. Penetration testing== pomdp solving? *arXiv preprint arXiv:1306.4714*.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2013b. Pomdps make better hackers: Accounting for uncertainty in penetration testing. *arXiv preprint arXiv:1307.8182*.

Sarraute, C.; Richarte, G.; and Lucángeli Obes, J. 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 71–80. ACM.

Shah, S., and Mehtre, B. M. 2015. An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques* 11(1):27–49.

Shmaryahu, D. 2016a. Constructing plan trees for simulated penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*, 121.

Shmaryahu, D. 2016b. Constructing plan trees for simulated penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*, 121.

Sohrabi, S.; Riabov, A.; Udrea, O.; and Hassanzadeh, O. 2016. Finding diverse high-quality plans for hypothesis generation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*.

Sohrabi, S.; Udrea, O.; and Riabov, A. V. 2013. Hypothesis exploration for malware detection using planning. *Edited By: Nicola Policella and Nilufer Onder* 29.

Umrao, S.; Kaur, M.; and Gupta, G. K. 2012. Vulnerability assessment and penetration testing. *International Journal of Computer & Communication Technology* 3(6–8):71–74.

Webb, E. M.; Boscolo, C. D.; and Gilde, R. G. 2016. Network appliance for vulnerability assessment auditing over multiple networks. US Patent App. 15/079,224.

Zhao, J., and Zhao, S. Y. 2015. Security and vulnerability assessment of social media sites: An exploratory study. *Journal of Education for Business* 90(8):458–466.