Towards Risk-Aware Planning of Service Delivery Operations

Mitesh Vasa
IBM Research Labs
Bangalore, India
Email: mitesh.vasa@in.ibm.com

Ashok Jadatharan
IBM Research Labs
Bangalore, India
Email: ashokgautham@in.ibm.com

Biplav Srivastava
IBM Research Labs
Bangalore, India
Email: sbiplav@in.ibm.com

***Abstract*—Infrastructure and Application Service Delivery typically comprises of complex workflows such as software installation/configuration, OS configuration, file system management etc. To manage costs as well as SLAs, enterprises constantly look towards automation of these workflows. While declarative approaches for automated workflow planning have been described in the literature, they do not address the risk exposure of the system. We present a novel two-phase risk-aware approach to planning which allows to "look-ahead" beyond the desired end state and compute a vulnerability score that measures how vulnerable the system could be in future, if the plan were to be executed now. We demonstrate our approach on a web service migration use-case using SGPlan as the preference-based planner. We also implement an end-to-end framework based on Puppet and validate it on a test bed of virtual machines. The implementation validates the generality of our approach in finding a less-vulnerable plan in scenarios where there are multiple plans.**

***Keywords*-Service Delivery; Automation; System Administration; AI Task Planners; Security; Risk; Vulnerability;**

I. Introduction

Over the past decade, there has been an exponential growth in IT systems investment globally. This increase in investment has fueled the need for an increasing number of IT infrastructure service professionals and service providers. A typical IT Infrastructure System (ITIS) contains various components such as server hardware, server operating systems, middleware systems, databases, application software, networking systems, storage sub-systems and security systems. ITIS is generally complex in nature because it is built with heterogeneous systems that are integrated together to function seamlessly. It is also highly critical to business because most of the businesses now completely rely on it. Due to these reasons, many of the IT infrastructure administrative tasks have been traditionally performed by system administrators (SAs), who need to ensure that the systems are in good health and run consistently throughout the year with very little or no breakdown. In the event of a breakdown, they diagnose the issue and remediate it to bring the system back to working condition within a narrow time window as defined by Service Level Agreements (SLAs), thus minimizing the impact to business.

Although the trend has been towards automation of system administration tasks, many of them are still manually performed, which can sometimes result in human errors leading to significant business impact and losses. Most large & mid-size IT service delivery providers adopt industry best practices such as Information Technology Infrastructure Library (ITIL) [1], Lean Six Sigma [1] etc. to cut labor costs and have better process governance and automation tools. Traditionally, IT automation has been achieved by SAs developing custom tools and scripts that target a particular device, operating system or application under consideration. But with the heterogeneous environment and the rate at which newer technologies are emerging, it becomes just too expensive to keep configuring and maintaining these *imperative* tools or porting them from one technology to another. Newer configuration management tools like Puppet [2], CFEngine [2] etc. have adopted *declarative* specification languages. This allows SAs to specify what needs to be done without specifying how it needs to be done. However, these tools do not provide a mechanism to evaluate the security risk exposure of the system.

In this paper, we introduce a novel two-phase risk-aware declarative approach to automate workflows by leveraging AI planners. A risk-aware workflow or a plan takes into consideration the security threats that the system will be exposed to at a future point of time, if the plan were to be executed at the current point of time. We model enterprise policies as preferences that must be satisfied by any valid plan. We also model security threats as a series of malicious actions that can lead to undesirable effects. In the first phase, we search for workflows or plans that satisfy the policy preferences and in the second phase, we employ a look-ahead approach to find any undesirable states that the plans may lead to. We evaluate each plan using a weighted vulnerability score. Higher the score, the more vulnerable the system would be. We show that we can help detect safer workflows and that we can do it quicker, relative to the execution of the plan's actions. We demonstrate the feasibility of our approach on a testbed with a small number of virtual machines (VMs) using a combination of SGPlan planner and Puppet manifests. Despite the small scale of experiments we performed, the generality of the

[1] https://www.axelos.com/itil
[2] https://puppetlabs.com/

978-1-4673-7281-7/15 $31.00 © 2015 IEEE
DOI 10.1109/SCC.2015.21

two components of the framework lends our approach to scalability. We scope our work to those workflows for which at least one plan can be discovered.

The paper is organized as follows. Section II gives a brief overview of AI planning as applied to service delivery framework. Section III describes our use-case of migrating a web service to a new VM. Next, Section IV lays out a two-phase approach for finding plans which can lead to undesirable states. Section V shows how the vulnerability score is computed. We also describe the implementation of plan's actions using Puppet and its execution in a test bed. Section VI describes other scenarios where our proposed extension can be applied. Finally, section VII concludes our paper and describes potential areas of future work.

II. Background

In this section, we give a brief overview of the prior art in planning vis-a-vis service composition and workflow automation to set the context for the rest of the paper.

A. AI Planning

Classical AI planning involves finding a sequence of actions, referred to as the plan, that when applied to the initial state in the domain, takes the system to a desired goal state. AI planners have been built to address a wide variety of problem domains ranging from robotics and space astronomy to manufacturing, mining and even gaming [3]. Over the years, various temporal planners, preference-based planners, planners that can deal with uncertainty, multi-agent planners etc. have been developed. The Planning Definition and Domain Language (PDDL) [4] is the de-facto standard for expressing problem and domain descriptions. For our experiments, we have used a preference-based planner called SGPlan [5]. Preference-based planning [6] extends the classical planning problem by allowing to express criteria that can generate plans of better *quality*.

B. Planning & Workflow Automation

AI planning is increasingly being used for IT configuration management and administration [7]. Weber et al. [8] used the FF-planning system for checking undoability of actions and generating undo workflows in cloud environments. Hagen et al. [9] developed a declarative approach for IT change planning in data centers that allowed reasoning of domain objects like databases, virtual machines etc. They applied planning directly over an object oriented configuration management database (CMDB). Similarly, the CHAMPS system [10] addressed IT change management processes by using planning to optimize resource selection & execution time and Maghraoui et al. [11] bridged the gap between a declarative plan and procedural provisioning operations using partial order planning. Herry et al. [12] applied automated planning for change configuration to search for an "autonomic" workflow and execute the resulting workflow using ControlTier and Puppet. They further improved on it [13] by specifying global constraints. In contrast, our work not only allows specifying global preferences, but also identifies vulnerabilities.

C. Planning & Service Composition

Peer [14] provided a comprehensive survey of the planning techniques applied to web service composition. He covered not only plan generation but also problem representation, goal formalization and plan representation. Apart from giving an overview of solutions based on classic planning techniques, he also covered planning with control knowledge. Chen & Yan [15] used a combination of GraphPlan with Dijkstra's algorithm to simultaneously satisfy the functional requirements of the problem and QoS criteria. Liu et al. [16] did not use planning, but introduced failure-causing trees to reduce the losses caused by task execution failure during web service composition. While we do not consider QoS in our composition, we solve a similar bi-objective function of satisfying the preferences while minimizing the vulnerabilities of the system.

D. Planning & Security

In the context of IT systems security and also our paper, vulnerability refers to a particular weakness in the system (for example, a missing security patch) that enables a threat agent (such as a hacker) to initiate malicious actions (such as stealing sensitive data) with a higher probability of success. There have been recent efforts to use AI planning for penetration testing and attack graph generation. Lippmann et al. [17] have compiled an impressive survey of approaches generating attack graphs [18] and compared them based on their performance, scalability etc. Boddy et al. [19] proposed a framework called Behavioral Adversary Modeling System (BAMS) that could detect course of actions leading to potential intrusions by coercing a traditional planner (Metric-FF) to generate multiple plans, and filtering them out in a post-processing phase. They modeled the behavioral aspects of an hacker/adversary. Sarraute et al. [20] went a step further by handling uncertainty in the target domain model. Obes et al. [21] integrated SGPlan planner with a pentesting tool and even validated the vulnerabilities by executing the attack path on target systems.

Our focus in this paper is to uncover vulnerabilities that can occur at a future point of time based on seemingly routine workflow operations that are performed at the current point of time. In this regard, our work is different from all the above cited works as well as from formal approaches around risk-aware business process modeling that focus on enterprise & process level risk [22] [23]. We employ a novel look-ahead strategy to compute vulnerability score - a weighted sum of undesired states that the system can end up in. We define undesired state as one where a series of malicious actions can be executed by a threat agent.

We restrict the scope of our work to modeling attacks of unauthorized entities, although the framework can handle other suites of threat goals. Modeling malicious actions and quantifying vulnerability during the planning phase is vital for service delivery organizations because a retrospective rollback of executed actions is often too costly for the enterprise, fraught with penalties and SLA breaches.

III. An Example

AI Planners need as input an initial state, a goal state, a set of predicates over the objects, and a set of actions that transform the state of the system. Action have a set of preconditions that must hold true for the action to be enabled and a set of effects that will hold true after the action is executed. Actions are typically defined in the domain PDDL file, whereas the initial state, goal state and preferences are expressed in the problem PDDL file. Most planners require these two files to generate a plan.

For planning purposes, we consider a real-world workflow of migrating a web-service to a web server on a new VM. Figure 1 shows the objects defined in the *Migrate* domain and their instances. We define only the basic objects required to model the use-case such as *vm*, *file*, *webserver* and *service*. In our use-case, we instantiate three different web server objects (*ws1*, *ws2*, *ws3*), three corresponding service objects (*ws1svc*, *ws2svc*, *ws3svc*) and two VM objects (*vm1*, *vm2*) in addition to a web service OSGI bundle (*osgi*) object.

```
(define (problem MigrateWebSvc)
        (:domain Migrate)
(:objects
    vm1 vm2 - vm
    osgi - file
    ws1 ws2 ws3 - webserver
    ws1svc ws2svc ws3svc - service
)
```

Figure 1. Objects & instances

Figure 2 describes the initial state of a system - the set of predicates which hold *true* before planning commences. It can be deduced from the list of predicates that the initial state makes available the usage of any of the two VMs, three web servers and the three file transfer protocols to copy and download files (i.e., *scp*, *sftp* and *telnet*). In addition, we specify that the VM *vm2* is firewall-enabled.

Figure 3 shows few sample actions defined in the domain PDDL file. For example, the action *startService* has a pre-condition that a web-server should be installed but not already started. The effect of the action is that the web server's service has started. Overall, we defined a total of 17 actions using 28 predicates in the domain PDDL file to model our use-case.

Figure 4 shows the goal state for our use-case, which is to migrate the web service to the new web server. It also defines two preferences. Preference *p1* specifies that for all VMs that are available, firewall-enabled ones are preferred. Preference *p2* specifies that for all files that are copied over the network, secure protocols like SFTP or SCP are preferable over unsecure ones.

In Figure 5, we specify two undesired goal states. The first one specifies a state where a service that is always supposed to be running is no longer running. The second

```
(:init
   (isVMAvailable vm1)
   (isVMAvailable vm2)
   (isWS1 ws1) (isWS2 ws2) (isWS3 ws3)
   (isSCP) (isSFTP) (isTelnet)
   (isOSGI osgi)
   (isServiceOf ws1svc ws1)
   (isServiceOf ws2svc ws2)
   (isServiceOf wc3svc ws3)
   (isFirewallEnabled vm2) )
```

Figure 2. Initial state

```
;;; Start a service
(:action startService
:parameters(?y - service ?s - webserver)
:precondition (and (isInstalled ?s)
   (isServiceOf ?y ?s)
   (not(isServiceStarted ?y)))
:effect (isServiceStarted ?y) )

;;; Migrate OSGI Bundle
(:action migrate
:parameters(?f - file ?w - webserver
   ?s - service)
:precondition (and (isFilePresent ?f)
   (isOSGI ?f)(isServiceStarted ?s)
   (isServiceOf ?s ?w)(canHostWebSvc ?w))
:effect (isMigrated ?f))
```

Figure 3. Sample PDDL actions

```
(:goal (isMigrated osgi)

;;;Use firewall-enabled VM
 (preference p1 (forall (?x - vm)
  (imply (isVMAvailable ?x)
         (isFirewallEnabled ?x))))

;;;Use secure protocol to transfer file
 (preference p2 (forall (?f - file)
  (imply (isFileCopied ?f)
  (or (fromSFTP ?f) (fromSCP ?f))))))
```

Figure 4. Goal state

```
;;; Goal-1
(forall (?s - service)
     (not (isServiceStarted ?s)))
;;; Goal-2
(canExecuteRemoteCode)
```

Figure 5. Undesired goals

goal specifies a state where a man-in-the-middle kind of attack can sniff data flowing in and that can lead to execution of malicious code remotely. We specify undesired goal states upfront and then perform look-ahead operations beyond the plan's actions to verify if those states can be reached.

IV. Approach

In real-world, executing a plan recommended by a planner translates to making actual changes on the IT systems, which could be anywhere from installation or reconfiguration to deletion, termination, reboot etc. Some of these actions, although perfectly valid and necessary for achieving the goal state, may lead to a systemic threat in future. This can happen because a planner considers only those actions whose "effects" help it move closer to the goal state, but ignores the "side-effects" that those actions generate. Our work described in this paper discovers these side-effects of a plan and measures the risk exposure of the plan by computing a vulnerability score. We divide our approach in two phases:

- In Phase-1, we apply preference-based planning to find a preferred plan.
- In Phase-2, we use a look-ahead approach to assess whether the preferred plan of Phase-1 leads to an undesired state, if it were to be executed now.

Phase-2 planning requires a set of malicious actions. The main reason behind defining malicious actions for Phase-2 separately is Separation of Duties. While domain actions for Phase-1 are typically defined by subject matter experts, the malicious actions need to be defined and maintained by security experts with the intention to model vulnerabilities that routinely impact IT systems. They model the actions that a 'hacker' or 'spamware' might take to exploit loopholes and gain illegal access or perform malicious activities. Such vulnerabilities are typically available in the Common Vulnerabilities & Exposures (CVE) list [3], Bugtraq [4], vulnerability scanner tools like Nessus [5] etc. Another reason for separating the domain actions in Phase-1 from the malicious actions in Phase-2 is the fact that one wouldn't want the Phase-1 planner to incorporate malicious actions in its plan. Malicious actions are simply used to check if a plan can reach an undesired state and make the system vulnerable to threats; they are not meant to be implemented and executed. We iteratively remove actions that are part of the preferred plan of Phase-1 thereby forcing the planner to find multiple alternative plans and setting the stage to compare various plans. It is possible that removal of an action may not always lead to a newer plan, in which case we simply reinsert the action back and proceed to the removal of next action.

Below we describe an example of how the look-ahead approach works. Table I shows the results of Phase-1 where preference-based planning is used to achieve a goal state $\{d\}$ given an initial state $\{a\}$. The plan P consists of actions $\{A, B, C\}$ in that sequence. The predicates shown in the last column form the effects of the action. For example, predicate $\{b\}$ is set to true for action A thus enabling action B whereas predicates $\{b', b''\}$ form the side-effects of the action. Table II describes the results of Phase-2 where the initial state $\{a, b, c, d, b', b'', d', d''\}$ is the union of the Phase-1 initial state $\{a\}$ and the combined effects of all the actions of plan P (i.e. the last column of Table I). The undesired goal states for Phase-2 are $\{k'\}, \{r'\}$ and $\{z'\}$, which are the effects of the malicious actions $\{K\}, \{R\}$ and $\{Z\}$ respectively.

Table I
Phase-1 planning with initial state $\{a\}$ and goal state $\{d\}$

Step	Preconditions	Actions	Effects
0	$\{a\}$	A	$\{b, b', b''\}$
1	$\{b\}$	B	$\{c\}$
2	$\{c\}$	C	$\{d, d', d''\}$

In Table II, there are three plans corresponding to the three undesired goal states, out of which the first two contribute to the vulnerability score. In other words, if plan P from Table I were to be executed on the system, it would expose the system to two security threats. The first plan in the table II consists of actions $\{J, K\}$ and the preconditions that enable action J are $\{b'', d\}$, both of which were effects of actions A and C respectively in Phase-1. Similarly, the second plan $\{P, Q, R\}$ also increases the vulnerability score because it reaches state $\{r'\}$ using several different predicates from the initial state. As for the third undesired goal state, there is only one action that can reach it (Z) and one of the precondition for that action ($\{a'\}$) is absent in the initial state. In other words, the plan P does not make the system vulnerable to the security threat exposed by malicious action $\{Z\}$.

We formally define vulnerability score as the weighted sum of undesired goal states that the plan can lead a system to.

$$V_s = \sum_{g \in G_u} w_g \cdot f(s, g)$$

where

$$f(s, g) = \begin{cases} 1 & \text{if } \exists p \; s \overset{p}{\rightsquigarrow} g \\ 0 & \text{otherwise} \end{cases}$$

[3] http://cve.mitre.org/
[4] http://www.securityfocus.com/archive
[5] http://www.tenable.com/products/nessus

Table II
PHASE-2 PLANNING WITH INITIAL STATE $\{a, b, c, d, b', b'', d', d''\}$ AND MULTIPLE GOAL STATES $\{k'\}, \{r'\}, \{z'\}$

Step	Pre-cond.	Actions	Effects	Vulnerable?
0	$\{b'', d\}$	J	$\{k\}$	Yes
1	$\{k\}$	K	$\{k'\}$	
0	$\{a, c, d''\}$	P	$\{q\}$	Yes
1	$\{b, q\}$	Q	$\{r\}$	
2	$\{b', r\}$	R	$\{r'\}$	
0	$\{a', b'\}$	Z	$\{z'\}$	No

s	State of the system after executing Phase-1 plan
G_u	Set of undesirable goals
$s \overset{p}{\rightsquigarrow} g$	Plan p applied to system in state s gets to goal g
w_g	weight assigned to the undesired goal state

The vulnerability score of a plan is an arbitrary positive real number where the higher the number, the more vulnerable the system could be. The weights assigned to undesired goal states range from $[0 - 1]$ and indicate the security expert's perception of risk given the context. Our modeling is simple to understand and yet non-restrictive. It is worthwhile to note that not all actions defined by a security expert for Phase-2 are malicious. For example, actions J, P and Q do not put the system in an undesired state by itself, but they take the system a step closer to the undesired state. As such, these actions can have an overlap with the domain actions defined by SAs in Phase-1, and that is acceptable and expected. Consider for example, actions such as *deleteFile*, *uninstallSoftware* or *stopService*. They can be viewed as both valid system management actions as well as malicious ones initiated by a bot/virus or by a user who has gained unlawful access to the system. Apart from overlapping action definitions, the predicates used in both the phases can also be part of a shared vocabulary to maximize reuse.

V. IMPLEMENTATION

In this section, we describe the implementation of planner and the subsequent computation of vulnerability scores. We also describe the execution component of our framework based on Puppet.

A. Planning

We use the preference-based planner SGPlan v5.2.2 [6] for our experiments. SGPlan was the winner of the 1^{st} prize in the Satisficing Planning track of the IPC-5 competition. We use SGPlan for Phase 2 of our experiments as well, though any classical planner that does not consume PDDL-based preferences would also have sufficed.

Without applying any preferences, if we were to work out a solution for the initial state from Figure 2 and the goal state from Figure 4, it is clear that there are a total of 18 unique plans considering that there are two VMs (*vm1*, *vm2*), three web servers (*ws1*, *ws2*, *ws3*) and three protocols to transfer files over the network (*scp*, *sftp*, *telnet*). However, with the two preferences from Figure 4 applied, the number of preferable plans reduce to a total of 6 (the option for using *vm1* and *telnet* gets eliminated because the former is not defined as firewall-enabled in the initial state and the latter is not preferred for file transfers due to preference *p2*). Figure 6 shows one of the preferred plans that the Phase-1 planning outputs. It generates ssh keys for *vm2* server, uploads public key to repository for password-less communication, downloads *ws2* software from repository, installs *ws2* web server, copies the web service bundle file via *sftp*, starts the corresponding *ws2* service and migrates the service.

```
0: (GENSSHKEYS VM2)
1: (UPLOADSSHKEYS VM2)
2: (DOWNLOADWS2 VM2 WS2)
3: (INSTALLSOFTWARE WS2)
4: (COPYUSINGSFTP OSGI)
5: (STARTSERVICE WS2SVC WS2)
6: (MIGRATE OSGI WS2 WS2SVC)
```

Figure 6. Preferred plan

We defined a series of malicious actions that a hacker or a trojan might perform to exploit vulnerabilities. One vulnerability occurs when *ws2* web server is used. It only allows the unsecure HTTP type of connections to the web service and this can lead to potential data sniffing during man-in-the-middle type of attack. This privacy breach can even lead to actions such as shutting down of web server! Another vulnerability modeled in the PDDL file concerns SFTP. Sending a file with overly long name over SFTP results in a heap-based buffer overflow which can then be exploited to allow remote execution of malicious code[7].

```
(:goal (not(isServiceStarted ws1svc)))
(:goal (not(isServiceStarted ws2svc)))
(:goal (not(isServiceStarted ws3svc)))
(:goal (canExecuteRemoteCode))
```

Figure 7. Undesired goal states

Figure 7 shows the undesired goal states resulting from malicious actions like *shutdownWebserver* and *executeRemoteCode* (not shown). The initial state, as described earlier, is the union of initial state from Figure 2 and the parameterized effects of the actions in Figure 6. We evaluate the plans against each of the undesired goal states and compute a vulnerability score (note that for the purpose of this experiment, we assigned equal weightage to the

[6]http://wah.cse.cuhk.edu.hk/wah/programs/SGPlan

[7]http://www.securelist.com/en/advisories/39921

goals). For example, the preferred plan of Figure 6 results in a vulnerability score of 2, as the planner can reach two undesired goal states: *(not (isServiceStarted ws2svc))* and *(canExecuteRemoteCode)*. This is because the preferred plan uses *ws2* web server and also uses *sftp* to transfer files, both of which enable malicious actions.

Table III shows the vulnerability scores of 6 preferred plans of our use-case. It can be seen that there are two plans having a vulnerability score of zero. Both these plans use a secure HTTPS-enabled web server, and use *scp* to transfer files. Hence, either of them is equally recommended for execution. Figure 8 shows one of these plans which is better (i.e. less vulnerable) than the earlier plan of Figure 6.

Table III
VARIOUS PREFERRED PLANS & THEIR VULNERABILITY SCORES

	Web server	Protocol	Vulnerability Score
vm2	*ws1*	*sftp*	1
vm2	*ws1*	*scp*	0
vm2	*ws2*	*sftp*	2
vm2	*ws2*	*scp*	1
vm2	*ws3*	*sftp*	1
vm2	*ws3*	*scp*	0

```
0: (GENSSHKEYS VM2)
1: (UPLOADSSHKEYS VM2)
2: (DOWNLOADWS1 VM2 WS1)
3: (INSTALLSOFTWARE WS1)
4: (COPYUSINGSCP OSGI)
5: (STARTSERVICE WS1SVC WS1)
6: (MIGRATE OSGI WS1 WS1SVC)
```

Figure 8. Preferred plan with zero vulnerability

B. Execution

Modeling a large domain such as IT service delivery operations in a planning language like PDDL, translating the resulting plan into real-world actions and executing them on actual systems is a significant engineering challenge. To understand the feasibility of building such a risk-aware planning & execution framework in a real-world scenario and to test its scalability aspects, we decided to create a working proof-of-concept. We chose Puppet as the execution platform as it is known to scale to large installations, being used to configure servers at Oracle, Rackspace, NYSE [8] etc. However, we found that the challenge of implementing such an end-to-end system posed several smaller challenges. Some of them were as follows:

1) Determining the right granularity for modeling the actions
2) Abstracting over differences between operating systems in a heterogeneous environment
3) Translating the plan to an executable script

Domain modeling: The domain modeler needs to ensure that each action in the domain is well-defined. For example, meta-action like `check-for-consistency` should be broken down into implementable actions. Further, each action must be independently executable if all its preconditions are satisfied. Finally, it is useful to have actions of comparable granularities. For instance, having a very high level directive such as at network layer protocol or a very low level function such as for network packets makes the final plan tough to comprehend and implement.

Abstracting differences across OS: Different OS have different system calls for performing the same task and hence the underlying implementation can be very different from each other. One major reason to use Puppet as an execution engine is its ability to use a declarative approach that abstracts out such differences.

Translating actions: A plan has actions that are parameterized. They need to be translated to the corresponding executable modules. We implemented a mapping utility that identifies the Puppet resource corresponding to the given action and instantiates it. By parsing the plan output, we extract the list of actions in the plan and translate them into a sequence of Puppet actions using the templates. We then copy these manifests to the corresponding nodes and issue *puppet apply* command to execute them.

Figure 9 shows how a template looks like for package installation. The mapping utility instantiates this template by replacing the parameter *<webserver>* with its actual value present in the plan's actions (ex: *ws1*). The value of *<version>* parameter is populated by referring another lookup table.

```
$<webserver>_pkg_name=$operating_system ?
{Debian =>
"/tmp/packages/<webserver>-<version>.deb",
Ubuntu =>
"/tmp/packages/<webserver>-<version>.deb",
Fedora =>
"/tmp/packages/<webserver>-<version>.rpm"}
```

Figure 9. A Puppet Template

Experimental Setup: The experimental setup is shown in Figure 10. We use a separate VM solely for planning. It performs the Phase-1 and Phase-2 planning for each Puppet node, computes the vulnerability scores for the different preferred plans, selects a least vulnerable plan and then composes the Puppet manifest using the mapping utility. It then sends the Puppet manifest file to the respective Puppet node, instructing it to apply the manifest.

[8]http://puppetlabs.com/about/customers

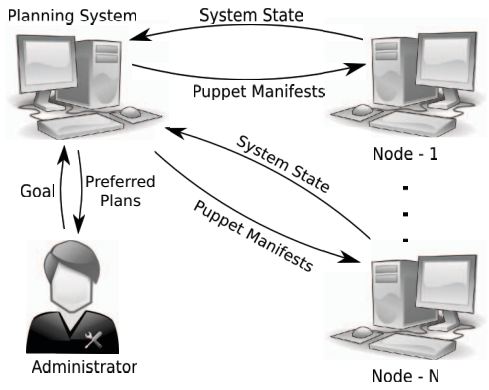

Figure 10. Experimental Setup

For validating our proof-of-concept, we created a test bed with 3 VMs one each for Ubuntu, Debian and Fedora and each acting as a Puppet node. We configured the nodes such that each of them has a slightly different initial state. For example, we made available both SCP and SFTP on one of the nodes; whereas we only enabled Telnet on another. Similarly, we tied the prerequisite for installation of web servers to specific OS. For example, for the Ubuntu node, only *ws1* and *ws2* could be installed but not *ws3* and so on. This allowed the planner to generate different plans of different lengths and complexity for each of the nodes. Ideally, we would like each node to report its initial state just before planning commences. Indeed, Puppet provides a 'Facter' component to do just this. However for the purposes of this experiment, we hand-crafted the initial states in the problem PDDL files for each node.

Performance: Table IV gives an indication of the performance of the various steps in the end-to-end scenario. While the planner normally runs in a fraction of a second for a given problem PDDL, the computation of vulnerability scores and the translation of the plan to Puppet actions is fast too. Iterative planning takes slightly more time as it involves finding multiple derived preferred plans from the first plan by removing and re-inserting its actions iteratively. Most of the time though, is spent in the execution of the Puppet actions, with some actions like *download* and *install* consuming a larger fraction of time relative to other actions. Note that we did not include domain modeling time in the table, as it is a one-time activity and it becomes faster with experience and usage of modeling tools [24]. While all our experiments were on a test-bed of 3 VMs, techniques to further scale both the planner and the Puppet configuration are well-known.

VI. Applications

There is inherent risk in many of the service delivery workflow operations that SAs perform on a day-to-day basis. Consider activities such as server build and software integration, which involves installing components such as operating systems, middleware applications, databases, security software such as anti-virus tools etc. Risk assessment is often a major part of the decision-making process in selecting the vendor or the version of these components. Other activities that SAs perform include user ID management (involving actions like adding, modifying or removing user IDs), storage space management (freeing up disk space, archiving the files, truncation or compression of log files, provisioning extra storage space etc.), application management (monitoring, starting or stopping of processes and services), network monitoring, system resource monitoring (CPU, memory, IO monitoring etc.), security compliance of servers etc. Our proposed solution can be potentially applied to many of these scenarios to identify the risk exposure of the system.

Table IV
Performance of various steps

S.No.	Item	Time	Comment
1	Phase-1 planning	< 1 sec.	per problem
2	Phase-2 planning	< 1 sec.	per problem
3	Iterative planning	< 5 sec.	per plan
4	Translation	< 2 sec.	per plan
5	Execution	depends on actions	per manifest

The generality of our approach can easily extend to other domains as well:

- Personalized dietary menu planning aware of risks such as allergies
- Financial investment planning aware of risks such as over-exposure to asset class
- Planning movement of high-value targets such as nuclear warheads, dignitaries etc.

VII. Conclusion & Future Work

In this paper, we described a novel two-phase approach for *risk-aware planning* applied to IT Service Delivery operations. We demonstrated the advantages of a risk-aware planner - it is the ability to compute vulnerability scores and compare multiple plans, so that a plan with a low vulnerability score can get discovered and selected, thus ultimately reducing the probability of the system getting exposed to security threats by threat agents. Our approach introduced a novel look-ahead strategy that allowed evaluating the risk factor of plans before their execution. This approach is generic and applicable to multiple domains. We showed the feasibility of implementing our approach and scaling it up using the combination of an AI planner and a configuration management tool like Puppet.

There are several directions of future work which we are currently investigating. There could be scenarios where intermediate states of a plan are not desirable, for example providing a user with privileged access which gets revoked later in the plan. We are investigating the use of trajectory-based constraints to handle such scenarios and a way to

compute vulnerability scores in such a scenario. Another area of future work concerns the specification and coverage of both domain and malicious actions. Full-scale domain modeling is hard, especially when the domain is huge such as Service delivery ops, or when the domain is not very well-formalized and keeps evolving such as IT security. While it is important to model as many actions as possible so that a variety of goals can be achieved, it is equally vital that they are specified at the right granularity level so that the executable script for the actions can be reused in different plans. We plan to extend our study to more complex use-cases surrounding network monitoring, user ID management, storage management etc. with security implications for the cloud, databases, network, web, devices etc. Recently, Munindar [25] described the Science of Security (SoS) and the theoretical model & formal constructs for security, and we would like to see similar constructs being applied to workflow automation domain. We believe our work is a first-of-a-kind on planning IT workflows that considers vulnerability modeling and quantification. There is wide scope for future work in defining a robust risk-based framework in this domain.

REFERENCES

[1] R. Shankar, *Process Improvement Using Six Sigma: A DMAIC Guide*. ASQ Quality Press, 2009.

[2] M. Burgess and A. Frisch, *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine*, ser. Short Topics in System Administration. USENIX Association, 2007, vol. 16.

[3] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[4] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Tech. Rep. TR-98-003, 1998.

[5] C. W. Hsu and B. W. Wah, "New features in SGPlan for handling preferences and constraints in PDDL3.0," in *Proceedings of the Fifth International Planning Competition*, 2006, pp. 39–42.

[6] A. Jorge and S. A. McIlraith, "Planning with preferences," *AI Magazine*, vol. 29, no. 4, p. 25, 2009.

[7] B. Srivastava, J. P. Bigus, and D. A. Schlosnagle, "Bringing planning to autonomic applications with ABLE," in *Proc. International Conference on Autonomic Computing*, 2004.

[8] I. Weber, H. Wada, A. Fekete, A. Liu, and L. Bass, "Supporting undoability in systems operations," in *27th Large Installation System Administration Conference*, ser. LISA'13. USENIX, 2013, pp. 75–88.

[9] S. Hagen and A. Kemper, "Model-Based Planning for State-Related Changes to Infrastructure and Software as a Service Instances in Large Data Centers," in *IEEE CLOUD*, 2010.

[10] A. Keller, J. L. Hellerstein, J. L. Wolf, K.-L. Wu, and V. Krishnan, "The CHAMPS system: Change management with planning and scheduling," in *NOMS (1)*, 2004.

[11] K. E. Maghraoui, A. Meghranjani, T. Eilam, M. H. Kalantar, and A. V. Konstantinou, "Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools," in *Middleware*, 2006.

[12] H. Herry, P. Anderson, and G. Wickler, "Automated Planning for Configuration Changes," in *Proceedings of the 2011 LISA Conference*. Usenix Association, 2011.

[13] H. Herry and P. Anderson, "Planning with Global Constraints for Computing Infrastructure Reconfiguration," in *Workshop on Problem Solving using Classical Planners at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[14] J. Peer, "Web service composition as AI planning - a survey," *University of St. Gallen*, 2005.

[15] M. Chen and Y. Yan, "QoS-aware Service Composition over Graphplan through Graph Reachability," in *Proceedings of the 2014 IEEE International Conference on Services Computing*, ser. SCC '14, June 2014, pp. 544–551.

[16] H. Liu, W. Zhang, K. Ren, C. Liu, and Z. Zhang, "A Risk-Driven Selection Approach for Transactional Web Service Composition," in *Grid and Cooperative Computing, 2009. Eighth International Conference on*, Aug 2009, pp. 391–397.

[17] R. Lippmann and K. Ingols, "An Annotated Review of Past Papers on Attack Graphs," Lincoln Laboratory, MIT, Tech. Rep., 2005.

[18] N. Ghosh and S. K. Ghosh, "A Planner-based Approach to Generate and Analyze Minimal Attack Graph," *Applied Intelligence*, vol. 36, no. 2, pp. 369–390, Mar. 2012.

[19] M. Boddy, J. Gohde, T. Haigh, and S. Harp, "Course of action generation for cyber security using classical planning," in *Proc. ICAPS 2005*. AAAI Press, 2005, pp. 12–21.

[20] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration Testing == POMDP Solving?" in *CoRR abs/1306.4714*, 2013.

[21] J. L. Obes, C. Sarraute, and G. Richarte, "Attack Planning in the Real World," in *AAAI Workshop on Intelligent Security*, 2010.

[22] Jakoubi, Stefan and Tjoa, Simon and Goluch, Sigrun and Kitzler, Gerhard, "A Formal Approach Towards Risk-Aware Service Level Analysis and Planning," in *ARES*. IEEE Computer Society, 2010, pp. 180–187.

[23] Tjoa, Simon and Jakoubi, Stefan and Goluch, Gernot and Kitzler, Gerhard and Goluch, Sigrun and Quirchmayr, Gerald, "A Formal Approach Enabling Risk-Aware Business Process Modeling and Simulation," *IEEE T. Services Computing*, vol. 4, pp. 153–166, 2011.

[24] T. L. Mccluskey, D. Liu, and R. M. Simpson, "GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment," in *ICAPS 2003*. AAAI press.

[25] M. Singh, "Keynote: Toward a Science of Security (and Governance) in Service Systems, SCC '14," 2014.