Active Deception Framework: An Extensible Development Environment for Adaptive Cyber Deception

Md Mazharul Islam
Dept. of Software and Information Systems
University of North Carolina at Charlotte
North Carolina, United States
mislam7@uncc.edu

Ehab Al-Shaer
INI/CyLab
Carnegie Mellon University
Pittsburgh, United States
ehab@cmu.edu

***Abstract*—Cyber deception provides a proactive cyber defense that can reverse the asymmetry in cyber warfare through confusing, misleading, or diverting attackers to false goals. However, developing and deploying adaptive cyber deception techniques in real-life operational networks is an extremely complex and time-consuming task due to the extensive efforts required to implement the underlying network infrastructure configuration functions that are necessary to support active cyber deception operations, including observing, planning, and deploying honey resources at real-time. Therefore, developers in this field often spend significant time and effort building such infrastructural functions rather than focusing on developing sophisticated strategies for cyber deception applications.**

In this paper, we developed an active cyber deception framework (ADF) that provides an extensible rich API and synthesis engine for developing advanced cyber deception applications. The API can be used to observe adversary actions, compose multi-strategy deception plans, and ensure safe yet quick deployment of deception plans by automatically managing the network configuration and operational tasks. In addition, ADF provides deception as a service by automatic orchestration of deception planning and deployment with minimal human involvement. We implemented our deception framework using the OpenDaylight Software-defined networking controller. We evaluated ADF using various case studies that demonstrate the rapid and cost-effective deployment of advanced application of active deception on real networks within a few seconds.

***Index Terms*—Cyber Deception, SDN**

I. Introduction

In modern cyber warfare, the prevalence of cyber asymmetry between the adversary and the defender is that the defender needs to protect all susceptibilities into the infrastructure. In contrast, the adversary requires one vulnerability to exploit. Existing reactive cyber defense techniques response after the attack has been launched and usually approaches the target for specific, known attack descriptions or signatures [1]. This often lets the adversary remain stealthy enough to learn the system and discover further vulnerabilities and possible lateral movement. In addition, skilled attackers can easily avoid static signature-based detection through exhaustive reconnaissance, fingerprinting, and social engineering [2]. Therefore, a proactive approach must be used by defenders to break this asymmetry. Cyber deception is a promising technology to achieve this goal.

Cyber Deception is an intentional misrepresentation of real systems' ground truth to manipulate adversary's course of actions under the premises of the defenders' rules [3]. However, the goal of cyber deception is beyond just to mislead adversaries. Cyber deception can *deflect* adversary away from their target to false or no target, *distort* their perception of the infrastructure by adding ambiguity and decoys into the network. Deception can *deplete* adversary consuming their computational power, delaying attack propagation by storming the static ground truth to a probabilistic state, for instance, increasing the number of probing by mutating the static IP addresses of critical resources. Finally, deception can engage with the adversaries to stir down under defenders' premises to *discover* their hidden tactics and techniques, which leads the defender to protect future zero-day attacks. In summary, deception is a rising paradigm in cyber defense that works beyond traditional detect-then-prevent techniques while effectively discovering new adversary tactics, consuming their resources, slowing down attack propagation, and learn adversary intention for future defense.

The current state of art depicts the increasing adoption rate of cyber deception because of its evident success over the existing reactive defense [2]–[9]. By 2022, it is expected that the global cyber deception market's expense will grow up to $2.3 billion [10]. However, developing cyber deception techniques in real networks is a highly complex task. It requires significant effort in implementation and network configuration management. Efficient and adaptive cyber deception needs continuous network monitoring to observe adversary activities, optimal planning for feasible implementation, and safe deployment without breaking the integrity of the system. As a result, few deception frameworks are developed and validated in the real-life operational environment.

To overcome these challenges, we develop an Active Deception Framework (ADF) that has extensible rich API to build sophisticated cyber deception applications. The goal of ADF is to make deception infrastructure as services through

978-1-7281-8388-6/20/$31.00 ©2020 IEEE
DOI 10.1109/SecDev45635.2020.00023

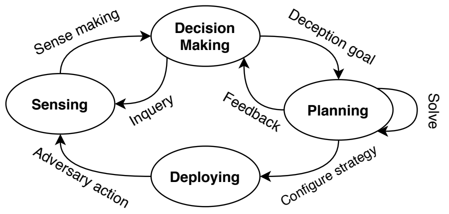


Fig. 1: Adversary adaptive deception deployment strategy.

API that shields deception architects from all intricate details about the low-level deception primitives implementation, orchestration and deployment, which eventually block them from developing novel and innovative deception applications. ADF provides an open environment for developing deception by 1) an extensible API that allows developers to build advanced deception application, 2) a decision-making synthesis engine (solver), that has satisfiability modulo theories (SMT) [11] for optimal correct by construction planning, ConfigChecker [12] for verification (reachability) analysis, etc., and 3) a controller for automated orchestration and dynamic composition of risk-aware deception planning deployment.

The ADF API creates a new dimension to make cyber deception as a service. It is extensible and available on GitHub; therefore, more functions can be added by the community based on requirements. The high-level deception planning API specifies the HoneyThings (honeypots, decoys, etc.) [3] configuration parameters and misrepresentation mechanisms of the system. While deployment, these parameter configurations can be done by low-level network management API. Thus, ADF has a comprehensive low-level network management API. The high-level deception planning API and low-level network configuration API make ADF an easy to develop multi-strategy deception planning and deployment in a timely and economical fashion.

We developed ADF over Software-defined networking (SDN). SDN provides a programmable environment over network configuration management through a centralized controller that enables comprehensive diagnosis of observations and quick deception action response. ADF supports a high-level deception API over the OpenDaylight [13] controller and a low-level network management API over OpenFlow protocol [14]. The framework has sensors to monitor adversary actions. ADF incorporates solvers such as SMT [11], ConfigChecker [12], etc., to optimize deception planning. We studied various deception planning deployments with different defense goals in real SDN testbed, showing the use case of ADF. Our evaluation results show that ADF can ensure proactive cyber deception by deploying multi-strategy deception policies within a few seconds.

II. Active Deception Architecture

ADF achieves adversary adaptive dynamic deception through multiple processes shown in figure 1. ADF has a set of sensors shown in table I, that observe adversary actions

TABLE I: ADF sensors, management and constraint API.

Sensor API	Management API	Constraints API
isHostScanning()	block()	getRouteRisk()
isLinkFlooding()	inspect()	overlap()
chekTrafficRate()	throttling()	isIncludeSwitch()
checkElephantTCP()	splitInspect()	getAvailableBandWidth()
getFlowStatistics()	priorityForwarding()	checkUniqueIP()
checkNewComers()	installFlowRule()	checkNonRepeateIP()
getCriticalLinks()	installNetworkPath()	checkSpatialCollision()
getAllFlowRules()	sendPacketOut()	getMinDetectionProb()
findNeighbors()	createTunnel()	getAttackUncertainity()
detectBot()	subscribeEvent()	canReach()
getPortID()	removeAllFlows()	getShortestPath()

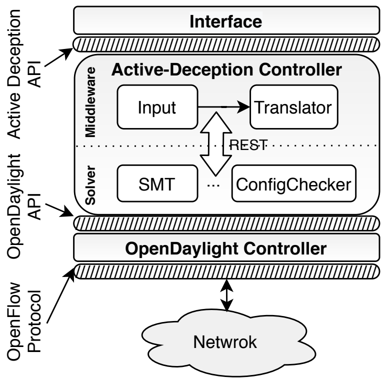


Fig. 2: Active deception framework.

to analyze the attack behavior, capabilities, and predict its intention. The sensors collect these sense-making observations from cyber resources such as host machines, IP addresses, OS, services, switches, links, and more. Using the sensors' report, the decision-making process deduces the current deception goals from deflection, depletion, distortion, or discovery. Then, in the planning process, an optimal set of defensive actions has been chosen to deploy.

ADF leverages multi-strategy deception planning solutions such as SMT, ConfigChecker, etc. to select the deception action policy optimally. These deception actions strategically change the low-level network or system configuration of the HoneyThings, e.g., decoy host creation, redirect traffic to proxy/decoy, IP mutation, a mix of true-false response generation for scanning request, etc. Finally, ADF deploys the deception policy into the network. The sensors observe the adversary response against the activated deception actions. The adversary may get engaged with the deception and run under the rule of the defender. However, he/she can detect the deception and adapt its technique as well. But in either case, the ADF sensor reports the current observation to plan new deception tactics. Therefore, ADF provides an adversary adaptive dynamic deception framework that is proactive, robust, and swift in deployment.

III. Active Deception Framework

Figure 2 shows the major components of ADF: the user interface to initiate cyber deception and the active deception

```python
class ActiveDeceptionApiHandler:
    def event_generator(self, request):
    ...
        deception_event = EventBuilderFactory
                          .build(request)
        output = ActiveDeceptionServiceImpl
                 .defenseByDeception(deception_event)
```

Listing 1: Middleware generates the OpenDaylight API deception event from user input (Python implementation).

```java
public class ActiveDeceptionServiceImpl implements
    ActiveDeceptionService
    ...
    public Future<RpcResult<DefenseByDeceptionOutput>>
        defenseByDeception(DefenseByDeceptionInput input){
        ...
        DeceptionPlan optimalConfiguration =
            Solver.solve(input.getConstraints())
        Future<RpcResult<CreateHoneyNetworkOutput>> status
            = createHoneyNetwork(optimalConfiguration);
```

Listing 2: The deception controller solves optimal deception planning configuration to create honey network (OpenDaylight implementation).

controller to deploy the deception along with the OpenDaylight controller over the SDN network.

A. Interface

The ADF interface provides an easy way to play with the deception API (ADF API) to build highly complex deception planning. For instance, the admin wants to initiate deception to proactively protect critical resources into the network while also wants to learn the network behavior for any future attacks. Therefore, she starts a multi-strategy deception technique, such as distort adversary, by creating anonymity and diversity into the network while diverting malicious traffic to a decoy machine for unknown attack tactics discovery. Such a multi-strategy deception can be launch swiftly and effortlessly using the ADF interface shown in listing 3. Listing 3 depicts a JSON request for launching deception by the API *createHoneyNetwrok()*. The details of the JSON and the API is discussed in section III-C. The interface delivers the deception triggering JSON request to the middleware through REST API.

B. Active Deception Controller

The Active deception controller (ADC) in the framework is the central orchestrator that handles the end-to-end processing of the cyber deception from initiation by the interface to the safe deployment in the network. ADC provides an open playground that enables prototyping or building advanced deception planning rapidly and safely using SDN. ADC leverages its facilities by providing a deception API that gives access to sophisticated cyber deception and OpenFlow management functions using the OpenDaylight controller. Besides, ADC incorporates with a decision-making synthesis engine called solver, that is capable of solving computationally hard problems using constraint satisfaction solvers (SMT Z3) [11], ConfigChecker [12], etc. to optimize deception policy actions. ADC composes the deception triggering by the interface,

```json
{"input": {
        "api": "createHoneyNetwork",
        "target": "r1",
        "impact": "high",
        "k": 2,
        "l": 3,
        "trigger": "activate"
    }
}
```

Listing 3: JSON request to launch deception by *k-anonymity* and *l-diversity* through API *createHoneyNetwork()*.

ensure safe low-level configuration changes, and deploy the planning into the network. Therefore, ADC makes cyber deception techniques as a service that users can access without taking any low-level configuration management headaches yet mitigate attacks proactively.

Middleware. The middleware translates the high-level deception API to OpdenDaylight API. It incorporates the solver through REST for solving constraint problems in order to optimize deception planning. Middleware creates a back-and-forth communication bridge to ADC with the user interface through ADF API and the SDN network through OpenDaylight API. We developed the middleware in python and run as a daemon server along with the ADF controller. The middleware uses a factory pattern to interpret user deception input and create a deception event correspondingly, shown in listing 1. Then it invokes the ADF service, which implements the OpenDaylight API to launch the deception. Listing 2 shows the OpenDaylight implementation of the deception deployment.

Solver. The solver in ADC is to optimize constraint problems to generate a feasible and practically deployable deception configuration. We designed the solver as a plug-in-play model in ADF architecture. Therefore, various deception and configuration optimization solution can be added with ADC as required. We integrated SMT solver to optimize anonymity and diversity of concealment configuration [2], ConfigChecker to solve reachability constraints [12]. ADC incorporates with the solver through middleware via REST API.

C. Active Deception API

ADF provides a comprehensive API for developing complex and multi-strategy deception plan. The API list is divided into four classes: a high-level public API for deception planning known as deception API (ADF API). The other three are low-level APIs. Sensor API for collecting network behavior to observe adversary actions. Management API to configure cyber resources such as switches, links, hosts, services, etc. The constraint API calculates risk, overlaps, reachability, availability while configuring honey networks. Table I shows the low-level API list, and appendix A describes their functionalities.

The novelty of ADF is the extensible deception API that can be used to build sophisticated multi-strategy deception planning in a cost-effective and timely fashion. It eliminates the challenges of deploying effective cyber deception policies that require frequent but complex low-level network configuration management. Because of the robust and expressive ADF

TABLE II: ADF Deception API

Name	Descriptions
createHoneyNetwork()	Dynamically creates a honey network with decoy/shadow hosts and services to analyze adversary for unknown TTP discover or distort them to delay attack propagation.
reDirect()	Redirect traffics to a given destination (can be a decoy or false target) and tunnel the packet to a proxy to generated trusted response.
reRoute()	Change the old path between a source and destination pair to a new path to avoid possible link flooding or other security measures.
routeMutate()	Change the route frequently of active flow(s) to another satisfiable route based on event or time.
hostMutate()	Randomizing real src/dst IP addresses to virtual src/dst IP addresses for depletion, so that real IP is used for routing but end hosts always uses virtual IP to communicate.
migrateService()	Create new machine with same services of the current target then migrates all benign traffic to the new machine.
spatioTemporalMutation()	Randomize the real IP of given hosts so that each host reach the same destination with a different IP address. Therefore, the view of the network is different for different host.
createShadow()	Creates an identical fingerprint (shadow) of a given host in the honeypot.
createDecoy()	Creates a decoy host. If the decoy is specified for a target host without specifying any services, then arbitrary but the same type of services will be created in the decoy, e.g., an FTP server but with a different vendors.

TABLE III: Deception API: *createHoneyNetwork()*

Param	Descriptions
target	The critical resources (hosts, services, links, etc.) to defend.
impact	Impact of the critical resources. (low, medium or high).
k	To anonymize fingerprinting, *k-anonymity* places $(k-1)$ shadow host with identical fingerprinting of the target host.
l	To anonymize configuration, *l-diversity* places $(l-1)$ fake services of same software type but different versions/vendors.
trigger	*activate*: Activate generated honey network.
	deactivate: Deactivate and remove honey network.

TABLE IV: ADF deflection API: *reDirect()* and *reRoute()*

	Param	Descriptions
reDirect()	***src***	Source host IP or flow ID.
	dst	Destination host IP or flow ID.
	to	The redirection destination, can be a switch, host, IDS or even the controller.
reRoute()	***src***	Source host IP or flow ID.
	dst	Destination host IP or flow ID.
	to	A new route consist of switches between ***src*** and ***dst*** e.g., s_1, s_2, s_4, s_9.

TABLE V: ADF depletion API: *spatioTemporalMutation()*

Param	Descriptions
h	Target host list for spatial mutation.
eIP	List of ephemeral IP addresses. (Optional)
$\boldsymbol{m_i}$	***eIP*** collision rate where $i \in h$
t	Lifespan of ***eIP*** (temporal period).
how	***eIP*** distribution fucntion, can be uniform or random

API, deception management is orchestrated automatically with minimal overhead. Table II describes a selective list of active deception API. The ADF API can be used to achieve various deception goal:

1) Anonymity and Diversity: Anonymity and diversity are effective concealments to hide the true identity of a host [2]. For instance, by randomizing the IP address of critical resources, agile defenses may fail because skilled attackers can identify their target by the static fingerprint of that host (e.g., OS, running services, and their versions). Therefore, deception by *k-anonymization* places $(k-1)$ shadow host with identical fingerprinting of the target host. Moreover, to defeat the static fingerprint problem, *l-diversity* places $(l-1)$ decoy host with fake services of the same software type but different versions/vendors.

The anonymity and diversity can easily be achieved using the *createHoneyNetwork()* API shown in table III. The API dynamically creates a honey network with anonymized shadow hosts and diverse decoy hosts. Listing 3 shows how to invoke *createHoneyNetwork()*. The *target* is the critical resource, *impact* defines the risk of being compromised (high means very critical resources), the k and l are integer value that defines the anonymity and diversity numbers.

2) Deflection by Redirection: The defender can deflect adversaries away from real host to a shadow/decoy host or an inspection environment to let them run in order to discover unknown attack techniques. ADF provides deflection API such as *reDirect()* and *reRoute()* to serve this purpose. Table IV shows the APIs. The *reDirect()* API deflects adversary traffic to an inspection host based on source IP or source flow ID. The *reRoute()* API is useful to deceive any man in the middle listener by changing the active route between two hosts.

3) Depletion by Spatio-Temporal Mutation: Depletion means the consumption of adversaries' resources by increasing its computation, confusing towards plausible target identification, and delaying in attack propagation. Therefore, the adversary requires to be more interactive with the system; eventually reveals its identity and hidden tactics. For instance, depleting scanning attacks can be quantified as the total number of probes required to make a hit to the target is higher than a certain threshold. Spatio-temporal mutation distorts the adversary views towards the network [15]. It randomizes the static IP binding to hosts periodically, so that collective reconnaissance information becomes obsolete after that period. Besides, the adversary requires to probe again if they make a further lateral movement. Therefore, Spatio-temporal mutation either makes the adversary hit the wrong target (decoy/shadow in ADF) or increase the total number of probes. The spatial mutation assigns a host multiple ephemeral IP (*eIP*) addresses so that each of the neighbors uses a different IP address to communicate with that host. Table V shows the ADF API for *spatioTemporalMutation()*. The parameter h is a list of target hosts of which IP will be mutated by the given list

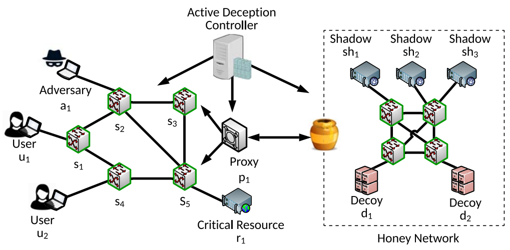


Fig. 3: Adversary adaptive deception by creating honey network with shadow and decoy hosts.

TABLE VI: ADF SMT solver creates honey configuration for critical host r_1 by 2-anonymity and 3-diversity.

Host	OS	Services		
r_1	Ubuntu	Vsftpd-2.3.5	Apache-2.2.22	MySQL-5.5.54
d_1	MacOS	CrushFTP 9.3.0	Nginx-1.17.8	Postgresql-10.2
d_2	Win7	FileZilla-0.9.58	IIS 7.5	SQL Server-13
sh_1	Ubuntu	Vsftpd-2.3.5	Apache-2.2.22	MySQL-5.5.54
sh_2	MacOS	CrushFTP 9.3.0	Nginx-1.17.8	Postgresql-10.2
sh_3	Win7	FileZilla-0.9.58	IIS 7.5	SQL Server-13

of ephemeral IP, *eIP*. If no *eIP* is given, ADF selects *eIP* randomly. Every target host receives a list of *eIP*; therefore, all neighbor reaches that target host by distinct *eIP*s. However, a target host may have the same *eIP* for multiple neighbors, which is defined by the collision rate m_i.

IV. Case Study

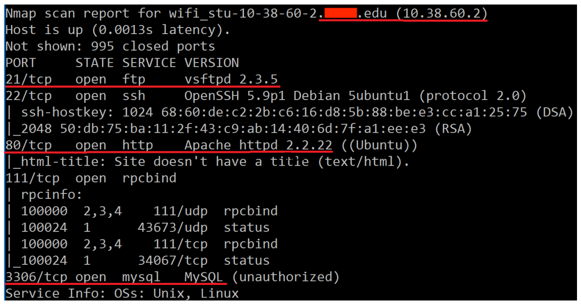
We extensively evaluated ADF by deploying various goal-oriented sophisticated deception plans into real networks. We measure that ADF incurs minimal overhead into the system.

```
Vagrant.configure("2") do |config|
    config.vm.define "shadow_1" do |shadow_1|
        shadow_1.vm.box = "hashicorp/precise64"
        shadow_1.vm.network "public_network", bridge: "Ethernet",
            ip: "10.38.60.2", netmask:"255.255.224.0"
        shadow_1.vm.provision "shell", inline: "sudo apt-get -y
            install vsftpd=2.3.5"
        shadow_1.vm.provision "shell", inline: "sudo apt-get -y
            install apache2=2.2.22"
        shadow_1.vm.provision "shell", inline: "sudo apt-get -y
            install mysql-server=5.5.54"
    ...
```

Listing 4: Automated shadow host (sh_1) configuration script in Vagrant.

A. Adversary Distortion by Anonymity and Diversity

We distorted adversary views towards the network by creating anonymity and diversity for critical resources. Figure 3 shows a network, where there is a probability that the adversary starts scanning to identify critical resource r_1. Therefore, the defender launches deception by distortion using ADF API *createHoneyNetwork()*. The API call is shown in listing 3. ADC incorporates ADF solver with the values of k and l to optimized the honey network configuration. Table VI portrays the optimized results. Real host r_1 has OS Ubuntu running on it with three different services. The solver generates two decoy d_1 and d_2 for diversity and three shadows sh_1, sh_2, and



```
Nmap scan report for wifi_stu-10-38-60-2.[redacted].edu (10.38.60.2)
Host is up (0.0013s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.5
22/tcp   open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1 (protocol 2.0)
| ssh-hostkey: 1024 68:60:de:c2:2b:c6:16:d8:5b:88:be:e3:cc:a1:25:75 (DSA)
|_2048 50:db:75:ba:11:2f:43:c9:ab:14:40:6d:7f:a1:ee:e3 (RSA)
80/tcp   open  http    Apache httpd 2.2.22 ((Ubuntu))
|_html-title: Site doesn't have a title (text/html).
111/tcp  open  rpcbind
| rpcinfo:
| 100000  2,3,4    111/udp  rpcbind
| 100024  1      43673/udp  status
| 100000  2,3,4    111/tcp  rpcbind
|_100024  1      34067/tcp  status
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OSs: Unix, Linux
```

Fig. 4: Nmap scanning finds new host (sh_1).

sh_3 for anonymity. After that, ADF generates a Vagrant [16] script shown in listing 4 to create the honey network with shadows and decoys. The honey network is connected with proxy p_1. ADF takes a few seconds to create honey networks with different sizes. Figure 6 shows that it takes around 3.7 seconds to create a honey network with twenty shadow and decoy hosts.

TABLE VII: Ephemeral IP assignment with real IP.

	Real IP	eIP	
u_1	10.0.0.1	10.0.0.10	10.0.0.11
u_2	10.0.0.2	10.0.0.8	10.0.0.9
r_1	10.0.0.3	10.0.0.6	10.0.0.7

Distortion then Discovery. After the deployment of the *createHoneyNetwork()*, adversary probing will yield a distorted view with many possible targets. A Nmap [17] scanning result is shown in figure 4, where the adversary finds a possible target sh_1, which is actually a shadow host. Therefore, adversaries either land in a decoy/shadow or increase probing to find its real target. Both lead them to engage with the defender, which makes them get detected.

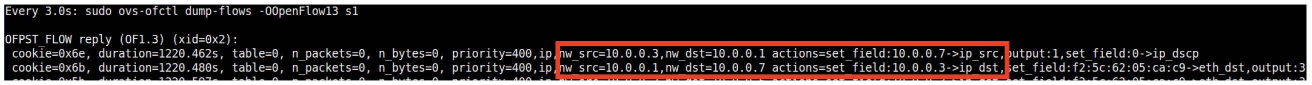
TABLE VIII: Forwarding entry mapping with real IP and *eIP*.

	u_1(10.0.0.1)	u_2(10.0.0.2)	r_1(10.0.0.3)
u_1(10.0.0.1)	-	10.0.0.10	10.0.0.11
u_2(10.0.0.2)	10.0.0.8	-	10.0.0.9
r_1(10.0.0.3)	10.0.0.7	10.0.0.6	-

B. Adversary Depletion using Spatio-temporal Mutation

The anonymity and diversity distorted the adversary view towards the network. However, the defender can deplete adversaries by calling the ADF API *spatioTemporalMutation()*. Assume that the defender wants spatial mutation for user u_1, u_2, and critical resource r_1. After the mutation API gets called, ADF assigns *eIP* for these hosts shown in table VII. Row 2 in table VII means, u_1 has real IP (10.0.0.1) and two ephemeral IP (10.0.0.10) and (10.0.0.11). Table VIII shows the forwarding rules of these host through *eIP*. For instance, Row 2 in table VIII means, u_1(10.0.0.1) communicates with u_2(10.0.0.2) and r_1(10.0.0.3) through *eIP* (10.0.0.10) and (10.0.0.11) respectively instead of its real IP (10.0.0.1).

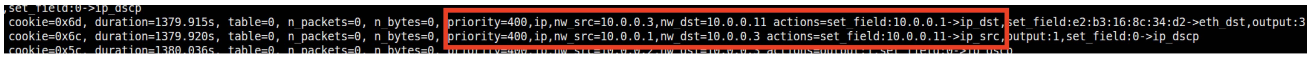
ADF installs the corresponding flow rules into the SDN switches shown in figure 5. The flow rules in figure 5a



```
Every 3.0s: sudo ovs-ofctl dump-flows -OOpenFlow13 s1

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x6e, duration=1220.462s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=set_field:10.0.0.7->ip_src,output:1,set_field:0->ip_dscp
 cookie=0x6b, duration=1220.480s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.7 actions=set_field:10.0.0.3->ip_dst,set_field:f2:5c:62:05:ca:c9->eth_dst,output:3
```

(a) $r_1(10.0.0.3)$ reaches $u_1(10.0.0.1)$ through eIP $(10.0.0.7)$ as source IP. Similarly, when $u_1(10.0.0.1)$ replies to eIP $(10.0.0.7)$, the destination changes back to $r_1(10.0.0.3)$ from $(10.0.0.7)$.



```
,set_field:0->ip_dscp
 cookie=0x6d, duration=1379.915s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.11 actions=set_field:10.0.0.1->ip_dst,set_field:e2:b3:16:8c:34:d2->eth_dst,output:3
 cookie=0x6c, duration=1379.920s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=set_field:10.0.0.11->ip_src,output:1,set_field:0->ip_dscp
```

(b) $u_1(10.0.0.1)$ reaches $r_1(10.0.0.3)$ through eIP $(10.0.0.11)$ as source IP. Similarly, when $r_1(10.0.0.3)$ replies to eIP $(10.0.0.11)$, the destination changes back to $u_1(10.0.0.1)$ from $(10.0.0.11)$.

Fig. 5: Flow rules for Spatio-temporal mutation.

shows, how r_1 sends packets to u_1 using *eIP* and gets reply. For instance, $r_1(10.0.0.3)$ reaches $u_1(10.0.0.1)$ through eIP $(10.0.0.7)$ as source IP instead of its real IP $(10.0.0.3)$. Similarly, when $u_1(10.0.0.1)$ replies to eIP $(10.0.0.7)$, the destination changes back to $(10.0.0.3)$ from $(10.0.0.7)$ and the replies delivered to the real recipient $r_1(10.0.0.3)$. The rules in figure 5b similarly shows how $u_1(10.0.0.1)$ reaches $r_1(10.0.0.3)$. After time interval t, the *eIP*s in table VII changes to a new sets of *eIP*s. ADF incurs limited overhead into the system for deploying spatial mutation. The cost depends on the total number of *eIP*. For a spatial mutation with fifty *eIP*, ADF requires 2.7 seconds to install all necessary flow rules into the network shown in figure 7.

Depletion then Discovery. Following the deployment of *spatioTemporalMutation()*, the real IPs for all hosts become obsolete to communicate with each other. Therefore, adversaries need to probe every after t times for reconnaissance. These raise the total number of probing significantly, which consumes the adversary resources, makes them less stealthy, and causes the attack costly. If the adversary directly probes the real IPs to any of the hosts, ADF marks that host as a potential scanner and redirect him to a shadow/decoy for further inspection.

C. Adversary Deflection by redirection

Distortion and depletion make stealthy attackers more interactive with the system, which increases their exposure. When adversaries frequently engage with the system, for instance, it increases the number of probes to identify targets, and the ADF sensor observes such unusual activities. Therefore, ADF can whitelist benign users and mark potential adversaries. ADF deflects such adversaries by redirection using API *reDirect()*. The *reDirect()* API installs the following rules:

1) *(src=*, dst=*IP_{r_1}*, set_dst:*IP_{p_1}*)*
2) *(src=*$IP_{attacker}$*, dst=*IP_{r_1}*)* $\rightarrow$ *(src=*IP_{p_1}*, dst=*IP_{d_1}*)*
3) *(src=*IP_{d_1}*, dst=*IP_{p_1}*)* $\rightarrow$ *(src=*IP_{r_1}*, dst=*$IP_{attacker}$*)*

The first rule redirects all traffic to the proxy that is direct probing to critical resource r_1, assuming there is a probability that the source is a potential attacker. Because, after mutation gets started, no benign user probes r_1 directly with its real IP. Therefore, the proxy redirects the traffic to a decoy to inspect the adversary activities. The next two rules are to make the adversary blind follower. For instance, second rules in the p_1 keep track of the source IP address while redirecting it to a decoy (d_1). When the decoy replies, p_1 reverse the source IP back to r_1 using the third rule. Therefore, the adversaries assume that they reach their target r_1, but the response is actually coming from a decoy (d_1).

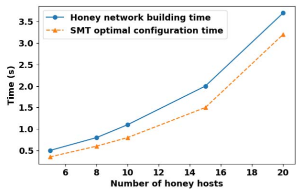


Fig. 6: Honey network creation overhead.

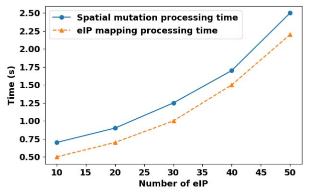


Fig. 7: Spatial mutation overhead.

V. Related Work

The current state of the art in cyber deception is enriched due to its effectiveness over existing reactive defense policies. Different theories and approaches are being used for optimized and powerful cyber deception planning. Game-theoretic approaches such as partially observable stochastic games [7], zero-sum Stackelberg game [18], Bayesian game [19], [20] provides adversary adaptive deception.

Reinforcement learning such as partially observable markov decision process (POMDP) [21], [22], game theory with Q-learning [23], and MDP [24] provide interactive cyber deception planning with the adversary in real-time. Probabilistic logic deception (PLD) [6], satisfiability modulo theories (SMT) [2], [5] provides optimized deception planning.

Deception framework like [2], [4], [25], [26] provides a practical implementation for optimizing complex deception plannings. However, these solutions do not focus on building a multi-strategy deception goal simultaneously. Software-defined networking (SDN) has been used for developing many cyber deception techniques for network securities against reconnaissance attacks [2], [4], [8], [15], [27]–[29].

VI. Conclusion

In this paper, we present an Active Deception Framework (ADF) that enables an open environment for developing sophisticated cyber deception applications. ADF facilitates swift, safe, and effective cyber deception deployment into the

SDN network. ADF leverages an extensive deception API that can be used to build multi-strategy deception policies. ADF provides sensors that observe adversary activities in real-time, helping to interact with adaptive adversaries to make active deception planning. In addition, the management API helps to conduct low-level network configurations without human intervention. ADF is flexible; its API can be extended based on requirements. We evaluated ADF in the SDN network showing different case studies by developing various goal-oriented deception strategies. ADF incurs very little system overhead while providing proactive defense by deception.

Acknowledgement

This research was supported in part by the United States Army Research Office (ARO) and Office of Naval Research (ONR). Any opinions, findings, conclusions or recommendations stated in this material are those of the authors and do not necessarily reflect the views of the funding sources.

References

[1] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 80–85, 2018.

[2] Q. Duan, E. Al-Shaer, M. Islam, and H. Jafarian, "Conceal: A strategy composition for resilient cyber deception-framework, metrics and deployment," in *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2018, pp. 1–9.

[3] E. Al-Shaer, J. Wei, W. Kevin, and C. Wang, *Autonomous Cyber Deception*. Springer, 2019.

[4] J. H. Jafarian, A. Niakanlahiji, E. Al-Shaer, and Q. Duan, "Multi-dimensional host identity anonymization for defeating skilled attackers," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. ACM, 2016, pp. 47–58.

[5] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang, *Cyber deception*. Springer, 2016.

[6] S. Jajodia, N. Park, F. Pierazzi, A. Pugliese, E. Serra, G. I. Simari, and V. Subrahmanian, "A probabilistic logic of cyber deception," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2532–2544, 2017.

[7] K. Horák, Q. Zhu, and B. Bošanskỳ, "Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security," in *International Conference on Decision and Game Theory for Security*. Springer, 2017, pp. 273–294.

[8] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using sdn-based virtual topologies," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1098–1112, 2017.

[9] K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major, "Game theory for adaptive defensive cyber deception," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 2019, pp. 1–8.

[10] Deception technology market research report- forecast 2022. [Online]. Available: https://www.marketresearchfuture.com/reports/deception-technology-market-2466

[11] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[12] E. Al-Shaer and M. N. Alsaleh, "Configchecker: A tool for comprehensive security configuration analytics," in *2011 4th Symposium on Configuration Analytics and Automation (SAFECONFIG)*. IEEE, 2011, pp. 1–2.

[13] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. IEEE, 2014, pp. 1–6.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proceedings of the First ACM Workshop on Moving Target Defense*, 2014, pp. 69–78.

[16] Vagrant. [Online]. Available: https://www.vagrantup.com/

[17] Namp. [Online]. Available: https://nmap.org/

[18] A. Schlenker, O. Thakoor, H. Xu, F. Fang, M. Tambe, L. Tran-Thanh, P. Vayanos, and Y. Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 892–900.

[19] T. E. Carroll and D. Grosu, "A game theoretic investigation of deception in network security," *Security and Communication Networks*, vol. 4, no. 10, pp. 1162–1172, 2011.

[20] E. Al-Shaer, J. Wei, K. W. Hamlen, and C. Wang, "Dynamic bayesian games for adversarial and defensive cyber deception," in *Autonomous Cyber Deception*. Springer, 2019, pp. 75–97.

[21] E. Miehling, M. Rasouli, and D. Teneketzis, "A pomdp approach to the dynamic defense of large-scale cyber networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2490–2505, 2018.

[22] M. Al Amin, S. Shetty, L. Njilla, D. Tosh, and C. Kamouha, "Attacker capability based dynamic deception model for large-scale networks," *EAI Endorsed Transactions on Security and Safety*, vol. 6, no. 21, 2019.

[23] K. Chung, C. A. Kamhoua, K. A. Kwiat, Z. T. Kalbarczyk, and R. K. Iyer, "Game theory with learning for cyber security monitoring," in *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 2016, pp. 1–8.

[24] O. Hayatle, H. Otrok, and A. Youssef, "A markov decision process model for high interaction honeypots," *Information Security Journal: A Global Perspective*, vol. 22, no. 4, pp. 159–170, 2013.

[25] J. H. Jafarian and A. Niakanlahiji, "A deception planning framework for cyber defense," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[26] M. M. Islam, E. Al-Shaer, and M. A. Basit-Ur-Rahim, "Email address mutation for proactive deterrence against lateral spear-phishing attacks," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2020.

[27] M. M. Islam, Q. Duan, and E. Al-Shaer, "Specification-driven moving target defense synthesis," in *Proceedings of the 6th ACM Workshop on Moving Target Defense*, 2019, pp. 13–24.

[28] C.-Y. J. Chiang, Y. M. Gottlieb, S. J. Sugrim, R. Chadha, C. Serban, A. Poylisher, L. M. Marvel, and J. Santos, "Acyds: An adaptive cyber deception system," in *MILCOM 2016-2016 IEEE Military Communications Conference*. IEEE, 2016, pp. 800–805.

[29] M. M. Islam, E. Al-Shaer, A. Dutta, and M. N. Alsaleh, "Clips/activesdn for automated and safe cybersecurity course-of-actions orchestration," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 2019, pp. 1–3.

Appendix

A. Active Deception Framework API descriptions

TABLE IX: Active Deception Framework Sensor, Management and Constraint API description.

	Name	Descriptions
Sensors	isHostScanning()	If any IP address sending SYN packets grater than the threshold th in a certain time window t, this function generates a true positive alarm.
	isLinkFlooding()	If the bandwidth consumed by the flows going to that link l, is greater than the the bandwidth threshold th, this will generate a true positive alarm.
	chekTrafficRate()	Calculate the average rate of a specific flow f of type (UDP and ICMP) flows.
	checkElephantTCP()	Calculate the percentage of large-size TCP traffic from a given flow list ¡f¿ .
	getFlowStatistics()	To get the complete information about a flow f such as: number of packets matched with f, bytes captured by f, time window for those packets, traffic type (ICMP, TCP, UDP) etc.
	checkNewComers()	Calculate the ratio of new IP source addresses from a given flow list ¡f¿ that has not been seen before recently in a given time window t.
	getCriticalLinks()	This function returns the critical links may generated in the topology based on the flow data path.
	getAllFlowRules()	This function retrieves all the flow rules available into a switch s.
	findNeighbors()	Returns all the neighbor hosts of the given IP address h that connected with the given switch s.
	detectBot()	If the signature of the examined packets satisfies the condition of bot traffic, return true.
Deception API	createHoneyNetwork()	Dynamically creates a honey network with decoy/shadow hosts and services to analyze adversary for unknown TTP discover or distort them to delay attack propagation.
	reDirect()	Redirect traffics to a given destination (can be a decoy or false target) and tunnel the packet to a proxy to generated trusted response.
	reRoute()	Change the old path between a source and destination pair to a new path to avoid possible link flooding or other security measures.
	pathMutate()	Change the path frequently of active flow(s) to another satisfiable path based on event or time.
	ipMutate()	Randomizing real src/dst IP addresses to virtual src/dst IP addresses for depletion, so that real IP is used for routing but end hosts always uses virtual IP to communicate.
	migrateService()	Create new machine with same services of the current target then migrates all benign traffic to the new machine.
	spatialMutation()	Randomize the real IP of given hosts so that each host reach the same destination with a different IP address. Therefore, the view of the network is different for different host.
Management API	block()	Block any incoming traffic from the given host IP address.
	inspect()	Inspect header and limited prefix data (application header), can be redirected to given IDS for advance inspection.
	throttling()	Policing the rate (traffic shaper) of the given traffic.
	splitInspect()	Divide the traffic and apply deep packet inspection.
	priorityForwarding()	Use priority queuing to forward traffic (gold, bronze and silver services)
	installFlowRule()	This function installs a single flow rule into a switch.
	installNetworkPath()	This function installs a complete data flow(s) path between source and destination host.
	sendPacketOut()	Forward the packets to the given destination (switch, host or controller).
	createTunnel()	Creates a tunnel where gateways at both ends only changes the source IP address to new IP address so that man in the middle get false information about source IP address.
	subscribeEvent()	Subscribes an event. An event can be a flow trace having type TCP, particular rate or given src-dst pair.
	removeAllFlows()	Removes all flow rules from a switch.
Constraints API	getRouteRisk()	Calculates the risk of a route based on the probability of each risk get attacked.
	overlap()	Calculates overlapping links between two route.
	isIncludeSwitch()	Checks if a particular switch available in a route.
	getAvailableBandWidth()	Checks the assgined bandwidth of a link.
	checkUniqueIP()	Checks if two sets of IPs are unique or not.
	checkNonRepeateIP()	Checks if the given IP is not used by other already assigned IP.
	checkSpatialCollision()	Checks the collisions in eIP in spatial mutation,
	getMinDetectionProb()	Calculates the minimum probability to detect an event based on given signature or specification.
	getAttackUncertainity()	Calculates the probability of a host, link or switch in a route that will be attacked with a given probability.
	canReach()	Find all reachable sources or destinations to/from a specific source s and destination d.
	getShortestPath()	Calculates the shortest path source s and destination d.