CHIMERA: Autonomous Planning and Orchestration for Malware Deception

Md Mazharul Islam*, Ashutosh Dutta*, Md Sajidul Islam Sajid*,
Ehab Al-Shaer†, Jinpeng Wei* and Sadegh Farhang†
*Department of Software and Information Systems, University of North Carolina at Charlotte, NC, USA
†Software Research Institute, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
*{mislam7, adutta6, msajid, jwei8}@uncc.edu, †{ehabalshaer, sfarhang}@cmu.edu

***Abstract*—Cyber deception is a promising defense that can proactively mislead adversaries and enables a unique opportunity to engage with them to learn new attack tactics and techniques. Although cyber deception has been around for more than a decade, static configurations and the lack of automation made many of the existing deception techniques easily discoverable by attackers and too expensive to manage, which diminishes the value of this technology. Sophisticated Advanced Persistent Threats (APTs) are highly dynamic and thereby require a highly adaptive and embedded deception that can dynamically create honey resources and orchestrate the deception environment appropriately according to the adversary behavior in real-time. This paper presents a theoretical framework and implementation for an autonomous goal-oriented cyber deception planner, called CHIMERA, that optimizes deception decision-making. CHIMERA agents can reside in any production machine/server and automatically create and orchestrate the deception ploys to steer and mislead the malware or APT to the desired goal without human interaction. The deception ploys are dynamically composed based on the deception planning while ensuring safe yet fast deployment and orchestration of deceptive course-of-actions. We evaluated our deception framework with real APT attacks for information stealing, ransomware, Remote Access Trojans (RAT), and others. In these case studies with 4,578 real malware samples, we showed that CHIMERA's adversary-aware dynamic deception strategies were able to effectively accomplish the deception goals within a few seconds and with minimum cost.**

I. Introduction

Cyber deception is a paradigm that aims to work beyond traditional detect-then-prevent approaches. In cyber deception, the defender intentionally conceals or falsifies the real configuration of the system's parameters (e.g., network topology, IP addresses, hardware IDs, registry keys, and so on) to create uncertainty and confusion for the adversary to mislead their perceptions and decision processes. The state of the art of cyber deception mainly focuses on developing high fidelity decoy systems that work as a standalone sandbox or virtual machine (VM) [1], [2], [3]. These decoys have fake files, user accounts, credentials, and more. If the adversary interacts or exfiltrates such honey resources, the defender gets alerts.

However, the goal of cyber deception is beyond just to catch attackers into VMs by setting up several traps. For instance, if the adversary has already penetrated the real system or fingerprinted the VMs to avoid, then static and decoy VM based deception techniques become ineffective [4], [5], [6]. Therefore, it is necessary to understand the distinct goals of deception to design an augmented reality that can be embedded with real systems. Firstly, cyber deception can be used to *divert* the adversary away from the real target to a false or no target when the adversary is already in the system—e.g., providing honey files [7] while the attacker searches for sensitive files. Secondly, the defender can *distort* the adversary's perception of the infrastructure by adding ambiguity into the system, e.g., running fake services with obvious vulnerabilities (honey-patches [8]). Thirdly, cyber deception can *deplete* adversary's computational power and resources to delay the attack propagation—for example, honey encryption [9] of the credential files, which the adversary needs to decrypt. Finally, the defender can *discover* new attack tactics, techniques, and procedures (TTPs) by letting them execute different attack actions in contained honey resources.

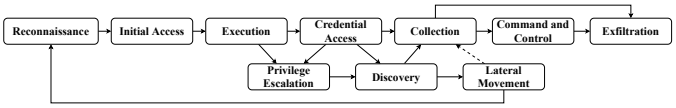

Fig. 1: APT *kill-chain* model.

Existing deception techniques are mainly developed to stop the attacker in a specific kill-chain phase. For instance, network-level deception techniques like redirecting malicious traffic to decoys [10], or generating a mystified response against probing [11] can protect reconnaissance. However, very few deception frameworks provide a deception composition strategy to defend APT actors in every kill-chain phase [12], [10]. Nevertheless, most honeypots and decoy systems are left behind with static deployment and configurations [13], [10], which skilled attackers can easily evade.

To address these limitations, we introduced CHIMERA, an autonomous framework that computes optimal cyber deception planning in real-time. CHIMERA relies on existing attack behavior from the MITRE ATT&CK [14] framework that classifies adversary actions into tactics, techniques, and procedures (TTP), which form a chain called TTP kill-chain. An example of a TTP kill-chain is shown in Fig. 1. Utilizing the knowledge base of existing APT behaviors, CHIMERA designs a deceptive environment through composing deception techniques, that achieves 4D deception goals (*divert*, *distort*, *deplete* and *discover*). CHIMERA deceives APT actors at every kill-chain phase. Although CHIMERA can be used to compose deception strategies for sandboxes or VMs, the

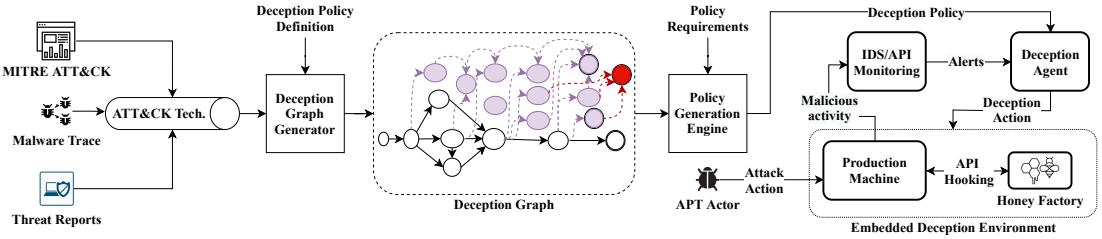

Fig. 2: CHIMERA Architecture.

novelty of the framework is to design deception environments that can be embedded with production machines to deceive APT attacks in real-time.

Developing such an embedded deception framework is challenging. First, APT actors are adaptive, who can detect and evade existing static deception techniques. Therefore, there is a risk associated with the failure of defense actions in an embedded deception environment, leading to a potential compromise of the system. Second, APT attackers are strategic, who can follow different attack paths to achieve their goal. Therefore, deception planning needs to have the capability to react to the current adopted attack approach. Third, the planning should be consistent, which means, if we lie at the reconnaissance phase (e.g., filename f), we must lie believably until the end of the exfiltration phase (e.g., honey content on that filename f). Finally, the deception planning needs to minimize the cost and system overhead while maximizing the achievement of deception goals.

Consequently, a deception action also depends on the attack action. Therefore, CHIMERA considers the strategic reasoning between attacker and defender by integrating the uncertain attack behavior into decision-making. To achieve that, we introduced a new attack propagation graph, called the deception graph (Fig. 3). The deception graph models the dependency between the attacker's action and the defender's deception by embedding two sets of state spaces named attack states (states where attacker lands by executing a successful attack action) and deception states (states where attacker lands if the deception action is successful). However, a deception graph for a particular APT group such as information stealer can have many states and transitions because of different attack techniques. Therefore, the defender needs to choose the optimal deception actions regarding effectiveness, costs, and overhead from an enormous set of choices to achieve his goal. We formulate the problem of selecting the optimal deception action using Partially Observable Markov Decision Process (POMDP), that optimizes deception planning by considering the dynamic attack and environment behaviors.

We implemented the embedded deception environment by hooking system-level APIs for Windows Operating System. First, we mapped 50 ATT&CK techniques with 191 distinct Windows APIs. Then, we created hooks for all of these APIs to perform the deception actions (e.g., generating a deceptive response) when the adversary calls them [15]. We evaluated CHIMERA in real-time with three different classes of APTs: Information Stealer, Ransomware, and Remote Access Trojan (RAT). We showed that CHIMERA has high efficiency in deceiving malware in embedded systems and incurs a very low system overhead. We also showed that the deception plan generated by CHIMERA is better in achieving distinct deception goals compared with state of the art such as Cuckoo sandbox or Any.run online malware analysis tools.

II. Threat Model and Scope of our Work

CHIMERA aims to design a dynamic deception environment that triggers appropriate deception actions in real-time to deceive the attacker's activity in every phase of the kill chain. For instance, CHIMERA can be used to set up a production machine against information-stealing APT so that defenders can analyze malware (e.g., discovering new TTPs) that exfiltrates sensitive information. CHIMERA has two significant aspects: 1) it can deceive malware already running into the production system because of the hooking deception techniques that work at the system API level. 2) APTs with decoy/VM detection capabilities are ineffective as CHIMERA provides embedded deception integrated with the real system.

It is important to note that the objective of CHIMERA is to deceive malware in a particular way such that defenders can achieve certain deception goals. Therefore, CHIMERA does not safelist processes between benign or malicious but deceives a given process based on the goal. For that, CHIMERA provides optimal planning, a sequence of deception actions based on adversary activity, and honey resources which can be pre-created honey files, credentials, and more. A myriad amount of research has been done on creating high-fidelity honey resources (such as decoy files [16], [7], network traffic [11], credentials [17], and systems [13], [10]). Therefore, in this work, we do not describe honey resources' creation; but focused on when and how to use them efficiently.

III. CHIMERA System Design

The architecture of CHIMERA is illustrated in Fig. 2. CHIMERA takes input as APT techniques from MITRE ATT&CK, threat reports, and malware API traces from sandbox (Cuckoo), and other analyzing tools like Hybrid Analysis, Any.run, and more. CHIMERA maps API call traces to ATT&CK techniques which are later used to generate a deception graph. The graph generator requires specification, such as deception action state space that can defeat certain attack actions. The deception graph is then used to generate optimal deception action planning, where policy definition is given as input, e.g., APT type (Information Stealer or Ransomware). CHIMERA has a deception agent in the production

MITRE ATT&CK Technique	Windows API Call Sequence	Adversary Action	Deception Action
Command and Scripting Interpreter	*CreatePipe, CreateProcess, CreateFile, ReadFile, CloseHandle*	execute	migrateInHE
Modify Registry	*RegCreateKeyA, RegSetKeyValueA, RegCloseKey*	addToRegistryRunKeys	doNothing
Query Registry	*RegOpenKey, RegQueryValue, RegCloseKey*	queryRegistry	honeyRegistry
Process Discovery	*1) CreateToolhelp32Snapshot, Process32First, Process32Next* *2) EnumProcesses*	tasklist	honeySwList
File and Directory Discovery	*GetCurrentDirectory, CreateFile, ReadFile, CloseHandle, FindFirstFile, FindNextFile, FindClose*	listDir	redirectToHoneyDir
Clipboard Data	*OpenClipboard, GetClipboardData*	copy	honeyCopy
Input Capture: Keylogging	*1) GetAsyncKeyState, GetKeyState, GetKeyboardState* *2) SetWindowsHookEx, GetKeyState, GetKeyNameText*	copy	honeyCopy

TABLE I: ATT&CK technique to Windows API mapping. Columns *Adversary Action* and *Deception Action* are names given by us to represent corresponding technique. We use these action names to describe state transitions in the Deception Graph.

machine and monitors (IDS/API monitoring) to observe APT actor's activity. The sensor alert triggers the agent to choose an optimal deception action to deceive the attacker's next action. The deception actions in CHIMERA are implemented in system-level API hooking that fetches honey resources from a Honey Factory to deceive the attacker. In the following sections, we will describe each component in detail.

A. API Sequence to MITRE ATT&CK Technique Mapping

MITRE ATT&CK [14] illustrates the APT lifecycle regarding tactic, technique, and procedure (TTP). A tactic (T) defines attack objective/goal, whereas, techniques (T) are actions to accomplish that goal. An attack technique describes the high-level context explaining why it is executed, what adversary gains from it, and how it is being performed. Such context helps the defender to design necessary deception actions. However, the APT actor interacts with the system using low-level procedures (P), a sequence of system API calls, to perform an attack technique. For instance, attack techniques such as sensitive files and directories search can be done by following shell commands: `dir`, `tree`, `ls`, `find`, `locate`, etc.

The defender can collect system log events from an ongoing APT campaign. However, to understand the attack context, it is necessary to map the API call traces to high-level attack techniques. Unfortunately, very few works have been done in mapping API calls to ATT&CK techniques and they are very limited [18], [19]. Therefore, we mapped the most essential 50 ATT&CK techniques frequently used in Information Stealer, Ransomware, and RAT with 191 windows API. From 4,578 malware samples, we collected more than 37000 API traces from Cuckoo sandbox [20] and their high-level behavior from tools like Any.run [21], Hybrid-analysis [22] and Malware Behavior Catalog (MBC) [18]. Finally, We go through the MITRE ATT&CK techniques description and manually map them with the API call sequences. Table I shows a subset of the mapping. We published the complete mapping in GitHub [23] to stimulate future research in this area.

B. Deception Graph

The deception graph in CHIMERA is a dependency graph to model the adversary propagation in the system over time. It is similar to attack graphs, where each node represents a distinct adversary position, and each edge represents the transition of attack propagation. A fragment of a deception graph generated for Information Stealer is shown in Fig. 3. Unlike the attack graph, the state transition in the deception graph depends on the interaction between attack and deception action. The attacker moves to a state by successfully executing a specific sequence of actions or getting deceived due to deception actions. For instance, in Fig. 3, the attacker has to successfully act *execute*, *setFileAttribute*, and *queryRegistry* actions to move from initial position *Phishing* to position *Credentials in Registry*. This research considers a distinct adversary position as a distinct *state* that comprises the overall state space S in our deception action planning (see section IV). The deception graph has two different types of states: (1) honey state (grey in color) where the attacker reaches after being deceived, and (2) real state (white in color) where the attacker reaches after successful execution of an attack technique. For example, in Fig. 3, the defender may redirect the attacker's traffic to a fake (decoy) C2 server, which takes the attacker to a honey state named *Deceived Exfiltration*. Whereas, by successfully exfiltrating data towards real C2 server, the attacker moves to real state *Exfiltration Over C2*.

The state transitions are represented as $(a_{d_1}/a_{d_2}/.../a_{d_n}, a_{v_1}/a_{v_2}/.../a_{v_m})$, where a_{d_i} is a deception action and a_{v_j} is an attack action separated by ",". Each pair (a_{d_i}, a_{v_j}) represents a distinct transition. Multiple states inside a dotted box represent a superstate. A transition to a superstate delegated for its appropriate sub-state only. For instance, the *(doNothing, tasklist)* transition means if the defender has no deception and the adversary does a *tasklist*, it will jump to the real state *Software Discovery*. Similarly, a *(doNothing, queryRegistry)* will lead the attacker to *Credentials in Registry* state.

C. Honey Factory

CHIMERA presents all the deceived responses to the attacker in an Honey Factory (HF). The HF consist of honey files, credentials, passwords, decoy user accounts, email accounts, web pages, software with honey patches, decoy process lists, registry files, honey traffic, decoy servers, VMs, and more. In CHIMERA, the honey resources in HF can be created offline in remote machines. In our evaluation, we have three remote VMs as part of HF. Therefore, the deception agent can delegate specific attack API calls (e.g., download a secondary piece of malware code command) from the production

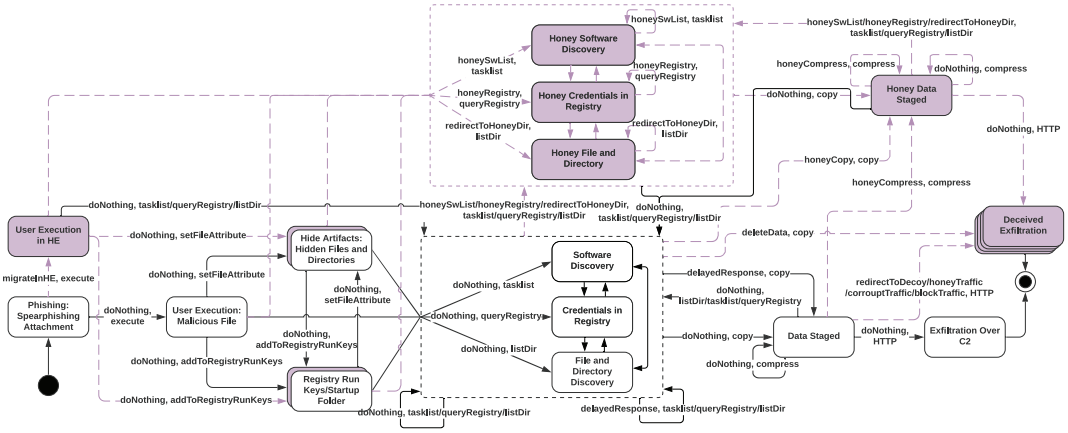

Fig. 3: Deception Graph mapping with MITRE ATT&CK techniques.

machine to one of the VMs through API hooking. However, HF resources can be created online inside the production machine. For instance, a decoy process list (honeySwList) can be shown in response to the adversary's tasklist command through hooking the APIs related to the process discovery technique (see Table I for deceiving process discovery API lists).

IV. Deception Planning

A. Deception Decision-making

To achieve the 4D goals cost-effectively, CHIMERA computes a policy that recommends an optimal deception action considering the current attack position and behavior.

Formulating Deception Decision-making: The optimal deception strategy at the current time-sequence t depends on the current adversary position. However, it is hard to infer the exact sequence of adversary positions from the beginning (at $t = 0$) due to dynamic environment and adaptive attack behavior. Therefore, we formulate the decision-optimization problem as Sequential Decision Process (SDP) [24]. The environment is stochastic due to non-deterministic deception consequences induced because of uncertain attack behavior.

Due to uncertain attack behavior, CHIMERA cannot certainly know the next attack action/move from a particular adversary position. For example, from the *Data Staged* position (in Fig. 3), the attacker may try to discover more credentials or exfiltrate data. Without knowing the next attack action certainly, there is a possibility that the deployed deception may be irrelevant to defend the current attack action. For example, let assume that CHIMERA decides to execute *honeyCompress* (i.e., compressing garbage data instead of real discovered data) to take the attacker to *Honey Data Staged* position in Fig. 3. Now, if the attacker *exfiltrates* data instead of *Compress*, *honeyCompress* is irrelevant. However, against unknown attack processes, CHIMERA can only probabilistically know the next attack move. Besides, sophisticated attackers may adapt action plans based on their observations about previous attack consequences, making any static action planning ineffective. CHIMERA formulates the SDP environment considering such deception failure probability.

In the SDP, CHIMERA executes a deception action and analyzes its consequences based on recent observations. Each observation specifies a distinct set of APIs called by the subjected process, based on which, CHIMERA also infers the current adversary position. However, the monitored set of APIs cannot be certainly mapped to a specific adversary position due to constrained monitoring of limited APIs. Hence, the environment is only partially observable with incomplete and imperfect information. To address the partial observability, CHIMERA formulates the decision-making problem as Partially Observable Markov Decision Process (POMDP) [24]. POMDP is a sequential decision process of an agent who acts and receives feedback from the environment synchronously, through addressing uncertainties related to partial observability. It is a tuple of $< S, A, T, \Omega, O, R, \gamma >$ where:

- S, A, and Ω are the state space, deception action space, and observation space, respectively,
- T and O represent the state transition function and observation function, respectively,
- R is the reward function,
- γ is the discount factor.

By solving the POMDP model, CHIMERA computes an optimal policy that recommends the optimal deception action for the current belief (i.e., probabilistic adversary position). Among these parameters, state space S consists of MITRE ATT&CK techniques represented as real state and honey state in the deception graph (section III-B), and deception action space, A, consists of unique deception actions considered for this research. Discount factor $\gamma \in (0, 1)$ regulates how far in future CHIMERA looks to understand the current action consequences into future, which is static in this paper.

B. State Transition Matrix

State transition matrix is a POMDP parameter that contains transitional probabilities among states for all considered deception actions. To clarify, the probability of transition from current state (i.e., adversary position) s to next state s' for the deception action a_d, for all possible $s \in S$, $s' \in S$, and $a_d \in A$ is denoted by $p(s'|s, a_d)$. While optimizing policy, POMDP solution approaches consider this parameter

mainly to understand action consequence on the environment [24]. CHIMERA determines $p(s'|s, a_d)$ using the following equation:

$$p(s'|s, a_d) = \sum_{a_v \in V} p(s'|s, a_d, a_v) \times p(a_v|s) \tag{1}$$

where, V is the attack space, and $p(s'|s, a_d, a_v)$ is the system behavior that defines the probability of transition from s to s' when the attacker executes a_v in response to a_d. Notably, V consists of unique attack actions. Fig. 3 shows some examples of attack actions by the second parameter across edges.

In Eqn. 1, CHIMERA integrates the expected attack behavior into its decision-model, in order to formulate the environment from defender's perspective [25]. This reduces the problem from Partially Observable Stochastic Game (POSG) to POMDP. Thus, CHIMERA addresses the limitation of POSG in approximating a solution for two players with different payoffs, which improves the scalability significantly.

To determine the system behavior, the agent uses the following equation:

$$p(s'|s, a_d, a_v) = \begin{cases} 1, & if \quad (s', s, a_d, a_v) \quad is \quad valid \\ 0, & Otherwise \end{cases} \tag{2}$$

In Eqn. 2, the first factor (s', s, a_d, a_v) is valid if, according to deception graph, the attacker transits to s' from s for attack action a_v while defense action is a_d. For example, in Fig. 3, *honeySwList* deceives the adversary action *tasklist*; hence, *honeySwList* is relevant to *tasklist*. The combination (s' =*Honey Software Discovery*, s =*User Execution: Malicious File*) is relevant to (a_d =*honeySwList*, a_v =*tasklist*), because the adversary moves to *Honey Software Discovery* from *User Execution: Malicious File* for executing *tasklist* against *honeySwList*. Importantly, the attacker always remains at the same state for doing nothing regardless of other factors.

In Eqn. 1, the second factor, $p(a_v|s)$ specifies the probability of executing action a_v at the current state s. This values come from the deception graph.

C. Observation & Observation Matrix

An observation is a distinct set of API calls that CHIMERA monitors to infer the current adversary position (state) in the attack chain. However, it cannot certainly specify the underlying state due to partial observability. Here, the observation space Ω is same as state space S. Each observation $o \in \Omega$ is highly correlated with one state $s \in S$ while having low correlations with other states. For example, based on recent observed set of API calls, there is high likelihood that the attacker performed *Software Discovery*. However, due to not monitoring all API calls, CHIMERA is not certain that the attacker has not performed other actions. We compute probabilities $p(s|o)$ defining the likelihood of s for the recent observation o, based on historical data of malware reports.

Composing Observation Matrix: Observation matrix O is a POMDP parameter that specifies correlations among states and observations. CHIMERA composes O to understand the current state from recent observation $o \in \Omega$. It contains probabilities $p(o|s)$ that specifies the probability of observing o when the state is s, for all possible $o \in \Omega$ and state $s \in S$.

CHIMERA determines $p(o|s)$ based on recent observation $o \in \Omega$ and prior probabilities $p(s|o)$, using the following equation:

$$p(o|s) = \frac{p(s|o) \times p(o)}{\sum_{x \in \Omega} p(s|x) \times p(x)} = \frac{p(s|o)}{\sum_{x \in \Omega} p(s|x)} \tag{3}$$

where, the probability, $p(o)$, of observing $o \in \Omega$ is same for all observations.

D. Reward

Alongside state transition matrix T, POMDP considers reward to optimize the policy through understanding the expected payoffs of deception actions for all possible scenarios. For all possible current state $s \in S$, next state $s' \in S$, and deception action $a_d \in A$, CHIMERA quantifies $R(s', s, a_d)$ that defines the payoff of a_d for the state transition from s to s'. Higher reward due to a_d motivates policy to execute a_d. CHIMERA formulates $R(s', s, a_d)$ using the following equation:

$$R(s', s, a_d) = -q(s') + \sum_{i \in \mathcal{G}} w_i \times Z_i^d - C_d \tag{4}$$

where, $q(s')$ is the risk of data exfiltration if the attacker reaches at state s', and $\mathcal{G}$ is the 4D deception goal consisting of *Diversion*, *Distortion*, *Depletion*, and *Discovery*. Additionally, Z_i^d defines whether a_d achieve the goal $i \in \mathcal{G}$ or not, and C_d is the installment cost of a_d.

In Eqn. 4, $q(s')$ at a state $s' \in S$ depends on two factors: (1) $\rho(s')$ that defines the probability of reaching to *Exfiltration Over C2* from s', and (2) L that is the loss (in dollars) due to data exfiltration. Notabaly, *Exfiltration Over C2* is actually the attack goal state s_g. The first factor, $\rho(s')$, depends on available paths to reach s_g from s', which are obtained from the deception graph at Fig. 3 (without considering the defense action). For example, from *Software Discovery*, the attacker can reach to s_g by the path with actions: (*copy*, *HTTP*) or by the path with actions: (*tasklist*, *copy*, *HTTP*). The second factor, L, is user-given and same for all possible scenarios.

There are multiple available paths to reach s_g from s, and the attacker reaches s_g if he successfully executes all actions of any of available paths. This intuition is formulated using the following equation:

$$\rho(s) = 1 - \prod_{l_j \in \mathcal{L}} (1 - \rho_j(s)) \tag{5}$$

where, $\mathcal{L}$ are available paths to reach to s_g from s, and $\rho_j(s)$ is the likelihood of reaching s_g through the path l_j. $\rho_j(s)$ depends on probabilities of executing attack actions following the sequence in l_j. Understandably, $\rho_j(s)$ gets lower with more required actions to reach s_g, that makes the reward higher. To clarify, from a honey state, he has to repeat previous actions

or execute more actions; thus, it reduces the risk of exfiltration and provides incentive to CHIMERA.

The second term in Eqn. 4 provides incentives to a_d for achieving specific deception goals using w_i. Notably, each deception action offers diversified benefits regarding 4D goals. For instance, *redirectToHoneyDir* provides diversion and discovery but not distortion and depletion, whereas, *Honey Data Staged* helps to discover attack behavior. Therefore, by w_i, the user can emphasize on actions that are more inclined to his objectives or mission. The last term, C_d, defines deployment cost of a_d due to required configurations, operations, and others. This paper considers w_i and C_d as user-inputs.

E. Belief

Belief b_t is the probabilistic distribution across all states $s \in S$, that probabilistically specifies the current state at time-sequence t through addressing the imperfect and incomplete environment observability. For instance, b_t(*Software Discovery*) defines the likelihood of *Software Discovery* as the current state. To address uncertainties related to observations, CHIMERA determines b_t considering the recent observation o, probable state transitions (state transition matrix), and correlations among states and recent observation (observation matrix). Using the traditional belief update approach [24], CHIMERA formulates current belief b_t using the following equation:

$$b_t(s) = \frac{p(o|s)\sum_{s'' \in S} p(s|s'', a) b_{t-1}(s'')}{\sum_{w \in S} p(o|w)\sum_{s'' \in S} p(w|s'', a) b_{t-1}(s'')} \quad (6)$$

where, $p(o|s)$ is the probability of observing o at state s, and $b_{t-1}(s'')$ is the belief about previous state s''.

F. POMDP Policy Generation

CHIMERA computes the optimal deception policy by solving the composed POMDP model. It recommends the optimal deception action a_d^* for current belief b_t. To address uncertainties associated with the environment, the composed POMDP model considers not only the probable attack behavior (by state transition matrix), but also correlations of observations (API traces) with probable attack positions (by observation matrix). CHIMERA applies Heuristic Search Value Iteration (HSVI) to solve the POMDP model, which approximates the policy with a bounded (user-given) regret rate [26]. Regret rate defines the precision of HSVI, and the approximated policy moves closer to the optimal policy for lowering its value.

HSVI takes an initial belief as the initial probabilistic attack position, in order to prune irrelevant belief space unreachable from initial attack position. The optimal deception action recommended by the computed policy maximizes the accumulated reward for the current belief b_t, considering not only the current attack position but also the probable attack propagation in future. The following equation formulates the intuition:

$$V^{\pi}(b_t) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s', s, a_t) | b_t, \pi\right]$$
$$\pi^* = \arg\max_{\pi} V^{\pi}(b_t)$$

where, $V^{\pi}(b_t)$ is the expected reward value considering future decision-horizon (defined by $\gamma \in (0, 1)$), π^* is the computed policy maximizing $V^{\pi}(b_t)$, and $R(s', s, a_t)$ is the expected reward for transition from s to s' by defense a_t at time t.

V. Implementation

We solve POMDP model by ZMDP (POMDP solver) [26]. We use python to create the deception graph from MITRE ATT&CK APT reports. We use API-to-MITRE (section III-A) mapping to include techniques in deception graphs from malware traces. For implementing the embedded Honey Factory (HF), we use system-level API hooking. To scale the deception actions with HF resources, we classified all deception strategies into four categories; 1) *Fake Failure*: this strategy always denies any API calls indicating a failure status, e.g., malware wants to compress a file but will get denied stating compression mechanism not found. 2) *Fake Success*: always return a successful response without performing the task. For instance, Ransomware wants to encrypt a file; it will return encryption successfully without any encryption. 3) *Honey Execute*: this strategy delegates the API calls to remote VM that executes the command, and the VM's response goes back to the attacker. 4) *Native Allow*: always allow any API calls.

Embedded Deception-API Hooking: We implemented the idea of deception strategy using EasyHook, a free, open-source hooking library for 32-bit and 64-bit Windows processes released under the MIT license. EasyHook provides a generic template for APIs hooking. In addition, EasyHook ensures thread safety by using a thread deadlock barrier. When the deception agent in CHIMERA decides to deceive the attacker's next movement (technique), the agent chooses a deception strategy. Let us assume the attack technique is File and Directory Discovery, and the defender's strategy is Honey Execute, which delegates every relevant API call to remote HF. From the API-to-MITRE mapping, the agent identifies the malware will invoke the following sequence of API: *GetCurrentDirectory - send - recv*. After identifying the relevant APIs, the agent uses Easyhook to inject a Dynamic Link Library (DLL) into the malware process including the strategy Honey Execute. According to our example, when the malware calls *GetCurrentDirectory* API, the current working directory was supposed to be copied into the parameter *lpBuffer*. However, the hook forwards the call to HF, and HF reports back a deceptive directory list, which gets copied back into the *lpBuffer*. Eventually, the malware receives the deceptive directory listing instead of the real one.

VI. Evaluation

We evaluated CHIMERA with 4,578 real malware samples from three different families: Information Stealer, Ransomeware, and Remote Acces Trojan (RAT). Our key evaluation

Family	InfoStealer													Ransomware				RAT		
Sub Family	LokiBot	Pony	Khalesi	Kegotip	Ramnit	Grozlex	Occamy	FareIt	Floxif	Emotet	Raccon	Generic.PWS	Trojan.PWS	WannaCry	Ryuk	Cerber	GandCrab	Gh0st	Pupy	Quasar
Samples	673	294	207	119	385	77	65	42	106	486	54	552	336	256	161	139	159	53	64	35

TABLE II: Datasets. 4,578 malware samples in total: 3,396 Information Stealers, 1,030 Ransomware, and 152 RATs.

Family	Malware Family	Discovery	Cuckoo	Any.run	CHIMERA
InfoStealer	Fareit	T	8	7	8
		P	39	126	149
	LokiBot	T	7	2	11
		P	21	173	243
	Pony	T	8	4	17
		P	231	191	582
	Racoon	T	7	8	16
		P	45	23	51
Ransomware	Ryuk	T	6	3	6
		P	27	32	102
	GandCrab	T	8	10	10
		P	192	109	245
RAT	Gh0st	T	2	2	6
		P	4	57	69
	VanilaRat	T	1	0	12
		P	1	0	12
	Quasar	T	5	2	13
		P	14	4	16

TABLE III: Techniques (T) and procedures (P) discovered by CHIMERA compared to Cuckoo sandbox and Any.run.

criteria were how efficiently CHIMERA deceives malware to obtain distinct deception goals in real-time. Further, we evaluate the quality of deception (in terms of discovery, depletion, and diversion), optimal policy generation, and overhead due to deception action deployments. Finally, we run CHIMERA in a production machine and discover new TTPs compared to existing APT analysis tools, such as Cuckoo [20] and Any.run [21]. We published our findings in GitHub [23] to incite future works in cyber deception.

A. Dataset

Table II summarizes the datasets we used in our evaluation. We collected a total of 4,578 malware samples from VirusTotal and MalShare. We choose 3,396 samples of Information Stealer from 13 families, 1,030 samples of Ransomware from 4 families, and 152 samples of RAT from 3 different families. We collected more than 37,000 API traces from Cuckoo sandbox [20] and DogeTron [13] from these malware samples. Furthermore, we collected the lifecycle of 27 Information stealing APTs, 17 Ransowmare APTs, and 19 RAT APTs from MITRE ATT&CK [14].

B. Experiment setup

We used 80% of our malware sample data randomly to build the model for deception action planning, and the rest is used for testing. We collected the API traces of the training malware samples from Cuckoo and DodgeTron. Using these traces and API-to-MITRE mapping (Table I), we built three different deception graphs for Information Stealer, Ransomware, and RAT. Existing works such as [27], [19] give us insight into calculating the likelihood of adversary transition from a given state to the next state. Further, we quantified the deception action cost and effectiveness as low, medium, and high. We then set up CHIMERA into a production machine with vulnerable software. We created a remote Honey Factory with three VMs. We used honey resources such as honey files, credentials, registry keys, passwords, decoy user accounts, process list, honey traffics, etc., in the factory VMs.

C. Deception Efficiency

We randomly chose 916 malware samples from our dataset to assess CHIMERA's efficiency in deceiving attackers. The results are illustrated in Fig. 4. We successfully deceived 879 malware samples (95.93%) that run to completion (e.g., exfiltration). Out of them, 781 malware samples took our baits and exfiltrated the honey resources, which we observe by inspecting the plain Command and Control (C2) communication. However, 98 malware samples sent encrypted traffic to C2, which we redirected to the decoy server because we could not verify whether they took honey resources or not. We failed to run 8 malware samples (0.89%) because of either not having the C2 server or the malware was unable to comply with system requirements. For instance, we could not run *VanillaStub.exe* (MD5: 185526401b0a3a083c797cac3598051a) RAT client for not having the master. The remaining malware samples did not run to completion due to expecting specific C2 commands/responses encrypted with specific keys.

D. Quality of Deception

We quantify deception quality by discovering new TTPs, unique API calls, distinct API call traces, and malware depletion to delay the attack propagation. We maintain the key criteria of successful deception: running malware to completion to reach its goal. We compare our outcomes with existing popular APT analyzers such as Cuckoo sandbox and Any.run.

TTP Discovery: Table III shows the performance of CHIMERA in discovering technique (T) and procedure (P). Clearly, CHIMERA discovers more attack techniques and procedures than Cuckoo and Any.run. For instance, when we run the Lokibot information stealer sample (MD5: 5ce9945d6999c9636c1f49e270382d6b) in CHIMERA, we observed it searches for files "C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\qldyz51w.default\pkcs11.txt" by calling the following APIs: *CreateFile*, *ReadFile*, and *CloseHandle*. This procedure resembles the "File and Directory Discovery" technique. Later, we discover another technique "Application Layer Protocol" due to calling the following APIs: *InternetOpen*, *InternetConnect*, *HttpOpenRequest*, and *HttpSendRequest*. Cuckoo or Any.run cannot detect the later technique and corresponding procedures due to not having a C2 server setup. With decoy C2 server, CHIMERA discovers more techniques and procedures. For the same reason, CHIMERA outperforms the other tools in discovering RATs.

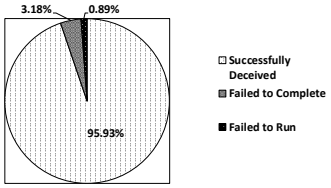

Fig. 4: CHIMERA deception efficiency.

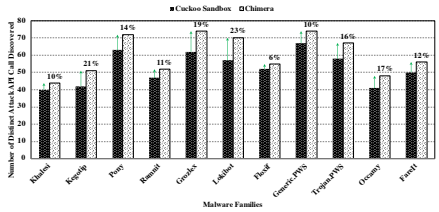

Fig. 5: Unique attack API discovered.

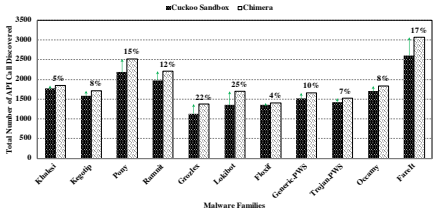

Fig. 6: Total trace API coverage.

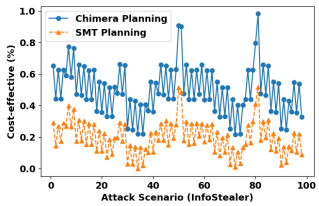

Fig. 7: Optimal policy over cost effectiveness.

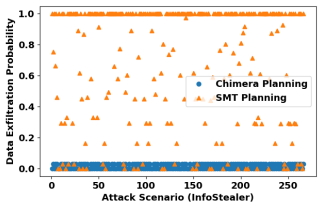

Fig. 8: Optimal policy over data exfiltration probability.

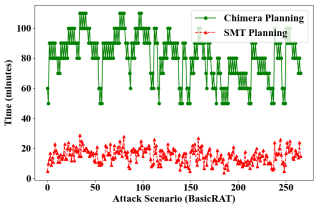

Fig. 9: Depletion time against BasicRAT.

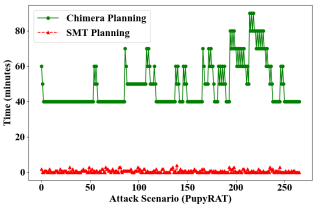

Fig. 10: Depletion time against PupyRAT.

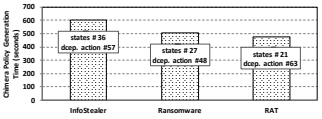

Fig. 11: Policy generation overhead for different APT.

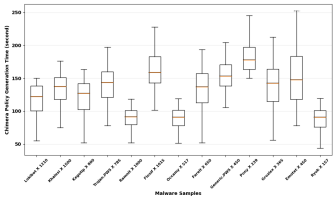

Fig. 12: Policy generation overhead for different malware.

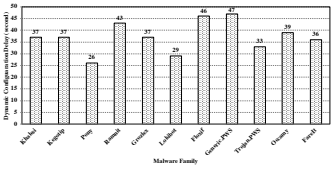

Fig. 13: Deception delay.

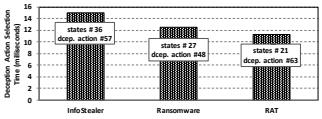

Fig. 14: Optimal deception action selection time (real-time).

Unique API Call and Trace Discovery: Fig. 5 shows the number of unique APIs discovered by CHIMERA compared to Cuckoo. On average, CHIMERA discovered 14.45% more unique API calls than Cuckoo. In total, CHIMERA found 78 distinct API calls that are not even monitored by Cuckoo at all. Fig. 6 shows distinct API trace coverage comparison. CHIMERA is ahead of 12.09% average trace coverage than Cuckoo. We consider a trace is distinct by taking the longest common subsequence from both CHIMERA and Cuckoo.

E. Optimal Policy

To measure how optimal CHIMERA's policy is, we created static planning using the same attack and defense action spaces over the deception graph and compare it with CHIMERA generated policy. We used Satisfiability Modulo Theories (SMT) to create the static planning. Fig. 7 shows that, while running different attack scenarios of an Information Stealer, CHIMERA always best performed choosing the most cost-effective actions, meaning CHIMERA chooses low cost actions with high efficiency in deceiving the attacker. Similarly, Fig. 8 shows that, the SMT planning mostly fails to prevent data exfiltration, whereas in CHIMERA planning, data exfiltration probability is nearly zero.

We engaged (to deplete the attacker to delay its progression) with two different malware, BasicRat that we called naive RAT, as it does not employ defense evasion techniques, and PupyRat, which can check the system to discover container environments (sandbox/VM). Fig. 9 shows that the SMT planning is capable of engaging with the attacker for 10 minutes on average before the BasicRat client quits because of not receiving the appropriate (deceptive) response. On the other hand, PupyRat detects the deception environment because of static planning in the minute mark (Fig. 10). In both cases, CHIMERA deception planning was able to engage with the attacker for more than an hour.

F. Policy generation overhead

We measured the overhead of CHIMERA in two aspects, 1) deception policy generation overhead and 2) delay due to deception action orchestration and deployment.

Offline Policy Generation Delay: From our datasets, we created three deception graphs for Information Stealer, Ransomware, and RAT, each having 36, 27 and 21 states respectively. Fig. 11 shows that the maximum policy generation overhead is 600 seconds for Information Stealer, having 37 states. The results also shows that the number of actions in deception graphs has little influence on the overhead. In Fig. 12, we individually created a deception policy for each sub-family and found that the average policy creation delay is around 250 seconds.

Online Action Selection Delay: CHIMERA deception agent selects an optimal deception action in run-time. Fig. 14 shows that the maximum delay to chose a deception action is 15 milliseconds for Information Stealer.

Dynamic Deception Delay: We calculated the dynamic deception delay by running a malware sample to comple-

tion into a machine without CHIMERA. Then we deployed CHIMERA into that machine and run the same malware. We calculated the time difference as system overhead due to deception action orchestration and deployment. Fig. 13 shows that the maximum orchestration delay is 47 seconds, which is insignificant compared to an APT campaign running time.

VII. Related Work

The state of the art of cyber deception focuses on designing virtual machines (VM) and sandboxes as a decoy [1], [2], [3]. These decoys have fake resources regarding files, software, user accounts, passwords, credentials, web pages, services, processes, servers, email accounts, and more [16], [8], [17], [7]. Alerts are sent to defenders if any activity is observed on those honey traps. However, skilled adversaries found unique ways to uncover such decoy VMs [4], [5], [6]. Usually, adversaries realize that isolated decoy machines do not initiate traffic, often responding late because they create complicated deception responses, making them easy to identify [28].

Research has been done to create network-level deception by malicious packet redirection [6] and falsified probe responses [11] to defend reconnaissance and lateral movement. System and data level deception such as honey password creation, decoy file generation, vulnerable software patching, etc., protects the system from internal attacks [1], [9]. Many research has been done to measure the efficacy of cyber deception [12]. Static planning and deception framework composing various deception actions is used to deceive APT actors [13], [29], [10]. To make the attack-defense battle dynamic, game-theoretic and probabilistic deception model has been introduced [30].

VIII. Conclusion

In this paper, we introduced a framework named CHIMERA that provides an optimal deception planning to deceive APT attacks and achieve the 4D deception goals: *diversion*, *deplетion*, *distortion*, and *discovery* in real-time. We use POMDP to obtain the optimal deception action planning. We developed a deception environment using Windows API hooking, where malicious API calls can be redirected to honey VM or responded with crafted honey content. Because of API level deception, we can use CHIMERA embedding with the production machine. We evaluated CHIMERA with 4,578 malware samples from three different families: Information Stealer, Ransomware, and RAT. We found CHIMERA deceives those malware samples with high efficiency (95.93%) and low system overhead (47s). The limitation of CHIMERA is that the deception is done through system level API hooking. If the malware can bypass API hooking or detect it, CHIMERA cannot deceive them.

References

[1] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[2] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, p. 102288, 2021.

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

[4] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale," in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[5] J. Rrushi, "Honeypot evader: Activity-guided propagation versus counter-evasion via decoy os activity," in *Proceedings of the 14th IEEE International Conference on Malicious and Unwanted Software*, 2019.

[6] L. Alt, R. Beverly, and A. Dainotti, "Uncovering network tarpits with degreaser," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014, pp. 156–165.

[7] P. Karuna, H. Purohit, S. Jajodia, R. Ganesan, and O. Uzuner, "Fake document generation for cyber deception by manipulating text comprehensibility," *IEEE Systems Journal*, 2020.

[8] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 942–953.

[9] A. Juels and T. Ristenpart, "Honey encryption: Security beyond the brute-force bound," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2014.

[10] 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.

[11] 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*, 2017.

[12] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman, "Examining the efficacy of decoy-based and psychological cyber deception," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[13] M. S. I. Sajid, J. Wei, M. R. Alam, E. Aghaei, and E. Al-Shaer, "Dodgetron: Towards autonomous cyber deception using dynamic hybrid analysis of malware," in *2020 IEEE CNS*, 2020, pp. 1–9.

[14] Mitre att&ck. [Online]. Available: https://attack.mitre.org/

[15] M. S. I. Sajid, J. Wei, B. Abdeen, E. Al-Shaer, M. M. Islam, W. Diong, and L. Khan, "Soda: A system for cyber deception orchestration and automation," in *Annual Computer Security Applications Conference*, 2021.

[16] J. Lee, J. Choi, G. Lee, S.-W. Shim, and T. Kim, "Phantomfs: file-based deception technology for thwarting malicious users," *IEEE Access*, vol. 8, pp. 32 203–32 214, 2020.

[17] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 145–160.

[18] Malware behavior catalog. [Online]. Available: https://github.com/MBCProject

[19] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1137–1152.

[20] Cuckoo sandbox. [Online]. Available: https://cuckoosandbox.org/

[21] Any.run. [Online]. Available: https://any.run/

[22] Hybrid analysis. [Online]. Available: https://www.hybrid-analysis.com/

[23] M. M. Islam. Chimera results. [Online]. Available: https://github.com/rakeb/Chimera

[24] D. Braziunas, "Pomdp solution methods," *University of Toronto*, 2003.

[25] R. Tipireddy, S. Chatterjee, P. Paulson, M. Oster, and M. Halappanavar, "Agent-centric approach for cybersecurity decision-support with partial observability," in *2017 IEEE HST*, 2017, pp. 1–6.

[26] T. Smith, "Zmdp software for pomdp and mdp planning," 2013.

[27] R. Al-Shaer, J. M. Spring, and E. Christou, "Learning the associations of mitre att & ck adversarial techniques," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–9.

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

[29] M. M. Islam and E. Al-Shaer, "Active deception framework: an extensible development environment for adaptive cyber deception," in *2020 IEEE Secure Development (SecDev)*. IEEE, 2020, pp. 41–48.

[30] 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.