A Cyber Deception Method Based on Container Identity Information Anonymity

Lingshu LI†a), Jiangxing WU†, Wei ZENG†, Nonmembers, and Xiaotao CHENG†, Member

SUMMARY Existing cyber deception technologies (e.g., operating system obfuscation) can effectively disturb attackers’ network reconnaissance and hide fingerprint information of valuable cyber assets (e.g., containers). However, they exhibit ineffectiveness against skilled attackers. In this study, a proactive fingerprint deception method is proposed, termed as Continuously Anonymizing Containers’ Fingerprints (CACF), which modifies the container’s fingerprint in the cloud resource pool to satisfy the anonymization standard. As demonstrated by experimental results, the CACF can effectively increase the difficulty for attackers.

key words: active defense, network deception, dynamic fingerprints, data flow anonymity

1. Introduction

Network reconnaissance refers to a precursory step of advanced attacks[1]. Since cloud computing presents a convenient remote network access method, the cyber assets (e.g., containers) are more likely to be identified, enumerated, investigated, and fingerprinted[2], [3]. Accordingly, cyber deception has been a research hotspot, blocking reconnaissance by creating false information and responses to hide the true identity (e.g., operating system obfuscation technology, virtual address technology, or address randomization technology). Besides, cyber deception can deploy decoy hosts, network services, or system information to capture and analyze attack behavior (e.g., HIoTPOT [4]). The container cloud is characterized by rapid elastic scaling and on-demand configuration, providing an advantage for cyber deception.

However, a single deception technique is ineffective against skilled attackers. Thus, recent studies attempted to use a composition of multiple deception techniques to achieve superlinear strength than deploying them individually. Duan et al. [5] combine dynamic rotation, diversification, and honeypot technology to improve the effectiveness of cyber deception. Jafarian et al. [6] employ IP/MAC addresses, service ports, service names, exploitable vulnerabilities, and others as host identity fingerprints. Then they propose an anonymous active defense system HIDE to continuously change the host’s fingerprint. However, excessive fingerprint modification and honeypots deployment cause large overhead.

2. Deception Method

Fingerprint anonymity tries to hide the sensitive attribute (e.g., the service type). Fingerprint k-anonymity means that for each container, a group of k – 1 containers with identical fingerprints (called shadow containers) are placed in the same network. Container fingerprint modification is implemented through a deception gateway, implementing address/port hopping and message modification like a Network Address and Port Translation (NAPT) device. Figure 1 illustrates the fingerprint anonymization deception architecture in the container cloud. Each container holds a service (e.g., Mysql or Web) and covers two fingerprints: Real Fingerprint (RF) and Modified Fingerprint (MF). Fingerprint contains several quasi-label attributes (e.g., OS type, container type). The defender makes the attack unreachable or misleads the attacker to the wrong target or honeypot, increasing the difficulty of the network reconnaissance.

Fig. 1 Fingerprint anonymization deception method
3. Methodology

3.1 Deception Overhead

Fingerprint deception brings computing overhead, service delay, and even interruption. A classification tree is built based on the natural semantic relationship to evaluate the deception overhead. The leaf node represents the attribute value, and the non-leaf node represents the attribute generalization.

For instance, the semantic classification tree of the OS attribute is presented in Fig. 2. The deception overhead of modifying the quasi-label attributes from leaf node \(v\) to leaf node \(u\) is defined as:

\[
d(v, u) = \frac{|Dv - 1|}{|D| - 1} - 1
\]

where \(|D|\) denotes the number of nodes in the classification tree and \(|Dv|\) denotes the number of nodes in the subtree whose root is the nearest common ancestor of node \(v\) and \(u\). For instance, when modifying the attribute from Ubuntu to Centos, the nearest common ancestor is Unix-like. We obtain \(d(\text{Ubuntu}, \text{Centos}) = 3\).

The function \(\gamma(s^1, s^2, \ldots, s^n)\) denotes the fingerprint of container \(s^1\) which contains \(n\) attributes. Thus, the deception overhead of modifying the fingerprint from \(f_1(a_1^1, \ldots, a_n^1)\) to \(f_2(a_1^2, \ldots, a_n^2)\) is defined as:

\[
\gamma(s^1, s^2) = \frac{1}{n} \sum_{i=1}^{n} d(a_1^1, a_2^2)
\]

For a container cluster \(\{s^1, s^2, \ldots, s^m\}\), the deception overhead is defined as:

\[
\gamma(s^1, s^2, \ldots, s^m) = \min_{j \in [1, \ldots, m]} \sum_{i=1}^{m} \gamma(s^i, s^j)
\]

3.2 CACF Algorithm

Containers are created, migrated, and deleted in the cloud continuously. Existing studies face the challenge of timely modifying the changing containers’ fingerprints. Thus, the Continuously Anonymizing Containers’ Fingerprinting (CACF) algorithm is proposed based on the popular privacy-preserving publishing algorithm CASTLE [9].

The main idea of CACF is elucidated as follows.

1. Maintain two sets: candidate K-anonymous cluster and published K-anonymous cluster.
2. For a new arrival tuple \(s\), add \(s\) to a candidate K-anonymous cluster, or generate a new cluster according to the deception overhead and cluster number limit.
3. Publish the candidate K-anonymous cluster (conduct cluster merging when the anonymity rule is not satisfied).

The main algorithm refers to Algorithm 1. Since the container creation, deletion, and migration in cloud resource pool are ubiquitous, the current containers’ status has to be obtained through container orchestration management tools (e.g., Kubernetes). \(\psi_i(n) \in \Psi\) denotes that container cluster \(\psi_i\) has \(n\) different service types. To maintain the currently running container clusters to satisfy the anonymization standard, CACF first adds tuples to clusters whose anonymization is destroyed by container deletion and migration. Subsequently, the \(\text{best_selection}\) function is called to aggregates the tuples dynamically by complying with their distance and fingerprint modification overhead. If the \(\text{best_selection}\) function does not find a suitable cluster, a new cluster will be created. Since the newly initialized container should get online in time, the CACF’s timeliness is of high importance. CACF defines the maximum waiting time of a container’s fingerprint modification and adopts a delay control strategy by \(\text{delay_constraint}(t)\). Suppose the input stream is \(S_{in}\), and the data stream after the K-anonymous publishing is \(S_{out}\). When the new coming tuple arrives at time \(s.t\), all tuples whose arrive time is before \(s.t - \delta\) should have been published.

The function \(\text{best_selection}\) is to select the most suit-

---

**Algorithm 1**  
**CACF**\((S, k, \delta, \beta, \tau)\)

**Input**: data stream \(S_{in}\), anonymous indicator \(k\); delay constraint \(\delta\), the upper limit of cluster number \(\beta\), anonymization result in the previous stage \(\Psi\), deception overhead threshold \(\tau\);

**Output**: anonymized data stream \(S_{out}\);

1. initialize the set of un-anonymized clusters \(\Gamma\) to \(\emptyset\);
2.\(\text{While} \ S_{in} \neq \emptyset \ \text{do}\)
3.\(\text{let} \ s \in S_{in} \ \text{be the currently processing tuple}\)
4.\(\text{for each} \ \psi_i(n) \in \Psi \ \text{do}\)
5.\(\text{if} \ n < k \ \text{then}\)
6.\(\text{add} \ \psi_i(n) \ \text{to} \ \Gamma \ \text{and delete from} \ \Psi\)
7.\(C_f = \text{best_selection}(\Gamma, \beta, s)\)
8.\(\text{if} \ C_f = \text{NULL} \ \text{then}\)
9. create a new cluster, put \(s\) into it.
10.\(\text{else}\)
11.\(\text{add} \ s \ \text{to} \ C_f\)
12.\(\text{let} \ s' \ \text{be the tuple arrived at time} \ s't = s.t - \delta\)
13.\(\text{if} \ \text{the tuple} \ s' \ \text{has not been output} \ \text{then}\)
14. \(\text{delay_constraint}(s')\)

---

**Fig. 2** Semantic classification tree of OS attribute
able cluster for a container fingerprint tuple. First, the cluster $SetC_{\min}$ closest to $s$ in existing clusters is found according to $\gamma(s \cup C_i)$ defined in Eq. (2). Subsequently, let $\tau$ be the threshold to decide whether insetting $s$ to any existing cluster. $\gamma(s \cup C_i)$ is the deception overhead after $s$ is added to the cluster $C_i$. If the deception overhead is less than the threshold $\tau$, the cluster will be added to the candidate set $SetC_{ok}$. Lastly, a cluster with the smallest cluster size is returned from $SetC_{ok}$. If $SetC_{ok}$ is empty and the upper limit is reached, a cluster with the smallest cluster size is returned from $SetC_{\min}$. Besides, if $SetC_{ok}$ is empty and the upper limit is not reached, $NULL$ is returned.

Algorithm 2 best_selection($\Gamma$, $\beta$, $s$)

| Input: fingerprint tuple $s$, un-anonymized clusters $\Gamma$, the upper limit of cluster number $\beta$, deception overhead threshold $\tau$ |
| Output: the cluster closest to tuple $s$ |

1 initialize candidate set $E$ to $\emptyset$
2 for each $C_j \in \Gamma$ do
3 $e_j = \gamma(s \cup C_j)$
4 add $e_j$ to $E$
5 $e_{\min} = \min E$
6 let $SetC_{\min}$ be the cluster corresponding to $e_{\min}$
7 for each $C_j \in SetC_{\min}$ do
8 if $\gamma(s \cup C_j) \leq \tau$ then
9 add $C_j$ to $SetC_{ok}$
10 if $SetC_{ok} = \emptyset$ then
11 if $|\Gamma| \geq \beta$ then
12 return the smallest cluster from $SetC_{\min}$
13 else
14 return $NULL$
15 else
16 return the smallest cluster from $SetC_{ok}$

4. Simulation

The experimental data set is derived from the history logs of a self-built container cloud, which involves seven attributes (i.e., operating system, container type, container size, image version, IP segment, service port, application type). The sensitive attribute is the application type. There are more than 20,000 tuples and all attributes are category attributes. The proposed algorithm is compared with CASTLE[9] and MDAV[10] to illustrate its effectiveness. CASTLE is a representative privacy-preserving publishing algorithm, and MDAV is a commonly used batch processing micro-aggregation algorithm.

4.1 Security Analysis

This study presents a deception test framework adopting network address hopping and container fingerprint deception. The defend probability is expressed as:

$$CM = e^{-1} + (1 - e^{-1}) \left(1 - \frac{l}{k}\right)$$

where $l$ denotes the number of sensitive attributes in the $k$ anonymous cluster. The quantity $e^{-1}$ results from invisible hosts due to IP hopping[5].

Subsequently, the probability of hit after $n$ round of scans is $T = 1 - (CM)^n$. It is assumed that the required hitting threshold is $T_n$. The reconnaissance attack is considered successful if $T > T_n$. The attacker’s attack cost is proportional to the number of attacks. To simplify the calculation, let $C_a = n$. Accordingly, the cost of an attacker performing a successful attack is:

$$C_a = \log(1 - T_n)/\log CM$$

Figure 3 presents the attack cost under different hitting thresholds. There has been a steady rise in the attack cost. The attacker has to pay more for the higher hitting probability. CACF algorithm tends to be more effective in improving the attack cost than MDAV and CASTLE. The CASTLE algorithm has a defect that it does not consider that the $k$-anonymity cluster may be destroyed by container offline. The CACF algorithm remedies the defect by replenishing containers to the existing anonymous cluster. Though it may increase the fingerprint overhead, it effectively improves the system security by increasing the attacker’s overhead.

4.2 Performance Analysis

Figure 4 draws a comparison of deception overhead under different $k$-value constraints. We can observe that the deception overhead of fingerprint modification increases uniformly with the increase of value $k$. The overhead of MDAV is the largest under the identical service request strength. As
it cannot use the existing high-quality anonymous groups, some containers are not divided into the optimal fingerprint cluster and obtain anonymity at a low cost. Compared with CASTLE, the proposed CACF algorithm has a slightly higher overall overhead. Since the online service may be affected or interrupted by fingerprint modification, the CACF algorithm minimizes secondary fingerprint modifications of running anonymized containers. Therefore, some new containers have to take the sub-optimal choice, increasing the total fingerprint modification cost. Since the published containers’ fingerprints are not employed for statistics or analysis, CACF does not limit the anonymous cluster’s size or involve a cluster segmentation process, thereby decreasing the fingerprint modification overhead.

The algorithms’ running time under different δ-value constraints is shown in Fig. 5. It shows that the MDAV algorithm runs the fastest for its simple logic, no complicated cluster merging and splitting operations, and no need to maintain the published anonymous data set. The time cost of the CACF algorithm is less than that of the CASTLE algorithm since CACF discards the split operation of large anonymous clusters. The published container fingerprint data is not required to maintain the original statistical significance. Thus, there is no need to reduce the size of the cluster that slightly affects security.

5. Conclusion

In this study, a clustering-based Continuously Anonymizing Containers’ Fingerprints method (CACF) is proposed for cyber deception, which uses multiple deception techniques to defeat network reconnaissance. To be specific, the proposed algorithm performs real-time anonymity on the containers cloud. As revealed from the experimental results, CACF outperforms conventional methods in security capabilities, and the modification overhead is acceptable.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2018YFB0804004).

References

[1] K. Park, S. Woo, D. Moon, and H. Choi, “Secure cyber deception architecture and decoy injection to mitigate the insider threat,” Symmetry, vol.10, no.1, p.14, 2018.
[2] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, K. Jian, and J.P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” Journal of Systems Architecture, vol.98, pp.289–330, 2019.
[3] Z. Shu and G. Yan, “Ensuring deception consistency for ftp services hardened against advanced persistent threats,” Proc. 5th ACM Workshop on Moving Target Defense, pp.69–79, Jan. 2018.
[4] U.D. Gandhi, P.M. Kumar, R. Varatharajan, G. Manogaran, R. Sundarasekar, and S. Kadu, “HiotPOT: Surveillance on IoT Devices against Recent Threats,” Wireless Personal Communications, vol.103, no.2, pp.1179–1194, 2018.
[5] Q. Duan, E. Al-Shaer, M. Islam, and H. Jafarian, “Conceal: A strategy composition for resilient cyber deception-framework, metrics and deployment,” 2018 IEEE Conference on Communications and Network Security (CNS), pp.1–9, IEEE, May 2018.
[6] J.H. Jafarian, A. Niakanlahiji, E. Al-Shaer, and Q. Duan, “Multi-dimensional host identity anonymization for defeating skilled attackers,” Proc. 2016 ACM Workshop on Moving Target Defense, pp.47–58, Oct. 2016.
[7] Z. Zhao, F. Liu, and D. Gong, “An SDN-Based Fingerprint Hopping Method to Prevent Fingerprinting Attacks,” Security and Communication Networks, vol.2017, no.2, pp.1–12, 2017.
[8] B. Anderson and D. McGrew, “OS fingerprinting: New techniques and a study of information gain and obfuscation,” IEEE Conference on Communications and Network Security (CNS), pp.1–9, 2017.
[9] J. Cao, B. Carminati, E. Ferrari, and K.-L. Tan, “Castle: Continuously anonymizing data streams,” IEEE Transactions on Dependable and Secure Computing, vol.8, no.3, pp.337–352, 2010.
[10] A. Rodríguez-Hoyos, J. Estrada-Jiménez, D. Rebollo-Monedero, A.M. Mezher, J. Parra-Arnaud, and J. Forné, “The Fast Maximum Distance to Average Vector (F-MDAV): An algorithm for k-anonymous micro-aggregation in big data,” Engineering Applications of Artificial Intelligence, vol.90, 2020.