DeepC2: AI-powered Covert Botnet Command and Control on OSNs

Zhi Wang, Chaoge Liu, Xiang Cui, Jiaxi Liu, Di Wu, Jie Yin

Abstract—Command and control (C&C) is the essential component of a botnet. In previous C&C using online social networks (OSNs), the botmasters’ identifiers are reversible. After a bot is analyzed, the botmaster’s accounts can be predicted in advance. Additionally, abnormal content from explicit commands may expose botmasters and raise anomalies on OSNs. To overcome these deficiencies, we proposed DeepC2, an AI-powered covert C&C method on OSNs. By leveraging neural networks, bots can find botmasters by avatars, which are converted into feature vectors and built into bots. Defenders cannot predict the botmaster’s accounts from the vectors in advance. Commands are embedded into normal contents (e.g., tweets and comments) using easy data augmentation and hash collision. Experiments on Twitter show that command-embedded contents can be generated efficiently, and bots can find botmasters and obtain commands accurately. Security analysis on different scenarios show that it is hard to predict the botmaster’s avatars. By demonstrating how AI may help promote covert communication on OSNs, this work provides a new perspective on botnet detection and confrontation.

Index Terms—Online Social Networks, Command and Control, Botnet, Convert Communication, Neural Networks

I. INTRODUCTION

A botnet refers to a group of compromised devices that are remotely controlled by a botmaster via command and control (C&C) channels [1]. Multiple types of attacks can be launched based on botnets, such as DDoS, spam and crypto-mining. Compared to other malware, the major feature of a botnet is that it has a one-to-many C&C channel, which receives commands from the botmaster and forwards them to bots.

With the evolution of botnet, the construction of C&C channels has focused on concealment and has begun to utilize online web services [2], such as online social networks (OSNs), cloud drives, and online clipboards. For example, Hammertoss (APT-29) [3] used Twitter and GitHub to publish commands and hide communication traces. HeroRat [4] used Telegram for C&C communication on Android devices. Turla [5] utilized Gmail to receive commands and to exfiltrate information to the operators.

OSNs have some features to build a good C&C channel. It is nearly impossible for OSNs to go offline. Users have access to OSNs anytime with a networked device. Then, visiting OSNs is allowed by most anti-virus software. This ensures the availability of the commands. As many people use the OSNs, the botmaster’s accounts can hide among normal users. Also, it is easy to limit the accounts but not easy to shut down the OSNs. With the help of dynamic addressing, the bots can obtain commands from multiple accounts.

However, there are still some deficiencies of building C&C channels using OSNs. Firstly, to help bots addressing, the reversible botmaster identifiers (i.e., ids, links, tokens, and DGAs) have to hardcode into bots (see Table I). Once the bots are analyzed by defenders, with the C&C infrastructure being exposed, all the botmaster’s accounts can be calculated in advance. Additionally, in most cases, commands are published in plain, encoded or encrypted forms (see Fig. 1). These abnormal contents will expose C&C activities and raise anomalies on OSNs, triggering restrictions on botmaster’s accounts and interrupting C&C activities. When OSNs block all the accounts from the botmaster, it is difficult for the bots to retrieve new commands and recover the C&C channel.

To reduce the anomaly of C&C communication on OSNs, Pantic et al. [6] proposed a method that embeds commands into tweets using tweet metadata (length). The length of a tweet represents an ASCII code in decimal form. As that time a tweet had a maximum length of 140, 7 bits could be conveyed through one tweet. While commands can be issued stealthily, the system has a low capacity. It needs to post N tweets to publish a command of length N.

The deficiencies above can be solved by introducing AI technology. Neural network models can obtain excellent accu-
racy in different tasks, but they have poor explainability [15]. It is not clear how neural networks make decisions. Therefore, it is difficult to reverse the decision-making process. Neural network models also have good generalization ability and fault tolerance. It can help to build block-resistant C&C communication.

The main idea of DeepC2 is as follows. To overcome the deficiencies brought by reversible hardcoding, we use a neural network model for addressing. The bots find the botmaster's accounts on OSNs by the avatars' feature vectors. The feature vectors are extracted by a neural network model. Defenders cannot calculate and predict the avatars in advance through the model and vectors. Then, to eliminate the abnormal contents, we propose a new method to embed the commands into contextual and readable contents (we take Twitter and tweets as the example in this work). To achieve this, the botmaster crawls tweets about a certain topic, and uses data augmentation to generate a large number of related tweets. Then the botmaster uses hash collision to get the command-embedded tweets from the generated ones and posts them. We use Twitter trends as the rendezvous point for the bots and the botmaster. The botmaster crawls and posts tweets to a trending topic, and the bots can find the botmaster quickly among the Twitter users. After addressing, the commands can be parsed from the tweets posted by the botmaster's accounts.

The contributions of this paper are summarized as follows:

- We propose a method to build a more block-resistant C&C communication on OSNs.
- We introduce neural networks to solve the problem of reversible hardcoding in C&C addressing.
- We propose a method for embedding commands into natural semantic tweets to avoid anomalies caused by abnormal contents on OSNs.
- We also evaluate this method and discuss possible countermeasures to mitigate this kind of attack.

Ethical Considerations. The combination of AI and botnet is considered to be an upward trend. We cannot stop the evolution of cyberattacks, but we should draw attention to the defenses in advance. The goal of this work is not to inspire malware authors to write more efficient malware but to motivate security researchers and vendors to find solutions for an emerging threat. To this end, we intend to provide this method to build a possible scenario to help prevent this kind of attack in advance.

The remainder of this paper is structured as follows. Section II describes relevant backgrounds on the techniques in this work. Section III presents the methodology for building the covert C&C channel. Detailed implementations are demonstrated in Section IV. Section V is the evaluations on the experiments. Section VI discusses possible countermeasures. Section VII presents related works to this paper. Conclusions are summarized in Section VIII.

II. BACKGROUND

A. Botnet

A botnet is composed of a botmaster, bots and a C&C channel. By reverse-engineering a bot, analysts will know how the bot addressing and communicating with the C&C server [16]. Also, analysts will know all the botmaster’s accounts (domains, IDs, etc.) and commands by running a bot. However, when the analysts know the commands, the bots in the wild also know the commands. From the defence perspective, it’s a failure [17]. Defenders should find and limit the botmaster’s accounts in advance. When addressing using DGAs or IDs, the botmaster’s identifiers can be calculated in advance. It gives the defenders chances to defend the C&C. Therefore, the key issue in designing a botnet is to design a block-resistant C&C channel that even the analysts know the detailed information about the C&C infrastructure, it is hard to calculate the botmaster’s identifiers and limit the C&C in advance.

B. Siamese Neural Network

The Siamese neural network [18] is effective in measuring the similarity between two inputs. The two inputs accepted by the Siamese neural network will feed into two identical neural networks to generate two outputs. Like “Siamese” twins sharing the same organs, the identical neural networks share the same architecture and weights. By calculating the distance between two outputs, the similarity between two inputs can be measured.

Fig. 2 shows the architecture of the Siamese neural network. In this work, the two identical neural networks are CNNs [19]. They are used to extract feature vectors from avatars. Images input into CNN models are converted into vectors of the same length. Contrasting loss function [20] is used to help backpropagate error and train the model.

C. Easy Data Augmentation

Data augmentation is a technique to solve the insufficiency of training data. By applying data augmentation, researchers can enlarge the existing dataset to meet the needs of training
works and promote the normalized performances of neural network models.

In this work, the botmaster needs to generate numerous tweets for hash collisions. Wei J. and Zou K. [21] proposed easy data augmentation (EDA) techniques in 2019. They used synonym replacement (SR), random insertion (RI), random swap (RS) and random deletion (RD) to generate sentences with similar meaning to the given sentences. Examples of EDA are shown in Table II with an original sentence from Twitter [22].

The augmented sentences may not be grammatically and syntactically correct and may vary in meaning. However, due to differences in language, culture and education, there are many grammatically incorrect tweets on Twitter. The Internet is diverse and inclusive. The botmaster should ensure that the tweets have semantics but do not need them to be “correct”.

D. Online Social Networks

People share content in different forms on OSNs. Due to different policies and user privacy settings, content access permissions vary on different OSNs. Some OSNs’ contents are limited only to authenticated users, while some have no restrictions that everyone can access all contents in the platform. Table III shows the content access restrictions of Alexa top OSN sites. Attackers can utilize the nonrestricted parts to convey customized information to others (including bots). In this work, to demonstrate the feasibility of the methods, we choose Twitter to build the C&C channel. The commands are embedded in tweets and posted by the botmaster.

III. METHODOLOGY

A. Threat Model

In this work, the C&C channel is built from the perspective of attackers. There are three prominent roles to this work: attackers, OSNs, and defenders. i) Attackers. We consider adversaries to be attackers capable of neural network and artificial intelligence techniques and have the ability to use various system vulnerabilities to get into a system. ii) OSNs. OSNs offer platforms for information publishing and retrieval. OSNs have the ability to limit the abnormal content and accounts based on their term of services. They can also actively detect autonomous and abnormal behaviors and limit the specious accounts according to their regulations. iii) Defenders. We consider defenders to be third-party unrelated to attackers and OSNs. Defenders have access to the vectors from the prepared pictures and the structure, weights, implementation and other detailed information of the neural network model. Defenders also have the ability to reverse engineer the bot program to obtain the detailed implementation of the bot.

B. Approach Overview

1) Overall Workflow: We take Twitter as the OSN platform to demonstrate the method. The main workflow is as follows. The botmaster posts contextual command-embedded content on an OSN platform first. Then, bots find botmasters through avatars with the help of a neural network model and parse commands from botmaster’s tweets. To achieve this, the botmaster needs to train a neural network model, prepare some pictures as future avatars, and extract feature vectors of the pictures through the trained model. The vectors and model are distributed with bots. When publishing a command, a botmaster chooses some trending topics synchronously with bots and generates and posts contextual, readable and command-embedded tweets on the topic. Bots crawl tweets to the trending topic along with tweeters’ avatars and then identify botmasters by comparing the avatars and vectors through the neural network model. If bots find botmasters, commands can be parsed from botmaster’s tweets. Due to the poor explainability of neural network models, it is difficult for adversaries to find botmasters in advance if the models and vectors are leaked, which ensures the security of botmaster’s accounts. Additionally, contextual tweets can eliminate the anomalies caused by abnormal content and conceal the intent of botmasters when the tweets are exposed to the public.

   We divide the entire process into two stages, as shown in Fig. 3. The first stage is the preparations for the model and bots. The second stage is for the communication between the botmasters and bots.

2) Preparation: In the first stage, the botmaster gets bots ready for C&C activities. The botmaster needs to make the
following preparations: 1) the neural network model and the vectors, 2) the rules for addressing, and 3) the Twitter accounts.

**Model and Vectors.** The botmaster needs to train a neural network model and prepare some pictures as future Twitter avatars first. Then the botmaster extracts the feature vectors from the pictures using the trained model (see Fig. 4). Next, the botmaster publishes the bots with the model and the vectors.

**Rules.** As the Twitter trends are the rendezvous point, the botmaster needs to set rules for selecting Twitter trends. Then, as each avatar and vector are used only once to prevent replay attacks, the botmaster also needs to have an updating mechanism for bots to update the vectors when necessary. The bots are built with these pre-defined rules.

**Accounts.** After publishing a command, the current Twitter account is considered unsafe, and reusing the account is not recommended. Therefore, the botmaster also needs to maintain some Twitter accounts for publishing commands at different times.

![Fig. 4. Extract features using neural networks](image)

3) **Communication:** The second stage contains four steps. Step 1 is to select a trending topic. The botmaster and bots visit Twitter trends and select a topic according to the predefined rules. The bots can visit Twitter trends periodically according to the specific tasks, and the botmaster only needs to visit Twitter trends when there are commands to issue.

Step 2 is for the botmaster to embed the commands to tweets. The botmaster uses hash collision to embed the commands, so it needs to generate numerous tweets to perform the collision. To this end, the botmaster crawls tweets on the selected topic first, and then uses EDA to generate the new tweets. If multiple tweets collide for a command, the botmaster can choose one randomly or by preference.

Step 3 is for the botmaster to publish the commands-embedded tweets. As the collided tweets contain the trending topic, the bots can obtain them after the botmaster posted the tweets on Twitter by searching the keyword(s). At this point, the botmaster should have selected a prepared picture and set it as the avatar so that the bots could find the account among the tweeters.

Step 4 is for the bots to find the botmaster and get commands. To this end, the bots crawl the tweets and tweeters’ avatars to the selected trend. Then the bots calculate the distances between the crawled avatars and the built-in vectors. If a distance is below a threshold, it is considered that the botmaster’s account is found. Bots can obtain the commands by calculating the hashes of the tweets posted by the botmaster.

**C. Technical Design**

1) **Neural Network Model:** A neural network model is used to protect the botmaster accounts and conceal bots' intent. The use of neural network models has the following advantages: 1) The neural network models are irreversible. Convolution is a lossy operation. Combined with some intentionally introduced losses, it is hard for adversaries to calculate the botmaster identifiers in advance. 2) The neural network models are fault-tolerance that similar inputs will generate similar outputs. 3) The neural network models have good generalization ability. After the model is trained, it will perform well on different data. It can help bots identify botmasters accurately and does not mistakenly identify someone else as a botmaster.

The method of converting an image into a vector is similar to [image-hashing][23]. In image-hashing, similar inputs have similar output hashes. Although image-hashing is more light-weighted, it is not suitable for this task. The non-neural network-based image-hashing methods are reversible. Adversaries can build images that produce similar vectors according to the given hashes. For neural network-based image-hashing [24], the learning tasks are more complex than those in DeepC2. As for cryptographic hash algorithms, they are sensitive to changes. As pictures uploaded to OSNs are compressed or resized (Table III), avatars are different from

### Table III: Contents access restrictions of Alexa top OSN sites

| OSN       | Login | Area | All | Posts Login | Area | All | Comments Login | Post | All | Compress | Watermark | Resize | Login | Area | All |
|-----------|-------|------|-----|------------|------|-----|----------------|------|-----|----------|-----------|--------|-------|------|-----|
| facebook.com | N     | N    | C   | N           | N    | Y   | Y              | N    | Y   | Y        | N         | N      | N     | R    | Y   |
| twitter.com   | Y     | N    | Y   | N           | N    | C   | N              | Y    | N   | Y        | N         | N      | R     | Y    | Y   |
| instagram.com  | Y     | Y    | Y   | N           | N    | C   | Y              | Y    | N   | N        | O         | Y      | N     | Y    | Y   |
| weibo.com     | N     | N    | C   | N           | N    | C   | Y              | Y    | Y   | N        | N         | O      | Y     | N    | Y   |
| tumblr.com    | N     | N    | C   | N           | N    | C   | N              | Y    | Y   | N        | N         | N      | Y     | N    | Y   |
| imgur.com     | N     | Y    | Y   | N           | N    | Y   | Y              | Y    | N   | N        | Y         | N      | N     | N    | Y   |
| pixnet.net    | N     | N    | C   | N           | N    | C   | N              | Y    | Y   | Y        | N         | N      | N     | N    | Y   |
| pixnet.com    | N     | Y    | N   | Y           | N    | Y   | Y              | Y    | N   | N        | N         | N      | N     | N    | Y   |

*facebook.com does not provide trends.

Login = Login to view, Area = Area restrictions, All = All contents is available, Post = Login to post
Y = Yes, N = No, O = Occasionally, C = Customized by user, R = Restrictions that can be bypassed.
the original images, which will cause hashes change due to the avalanche effect \cite{25}. So cryptographic hashes are also not suitable for this task.

The model is used differently for botmasters and bots, as shown in Fig. 5. For the botmaster, the model is used to extract feature vectors from avatars. The botmaster feeds the model with a batch of pictures, and the model outputs a batch of vectors that represent the pictures. For bots, the model is used to calculate the distances between avatars from Twitter users and the vectors to identify botmasters. A selected vector and a crawled avatar are fed into the model, and then the model outputs the distance of the inputs.

2) Avatars and Vectors: To prevent replay and enhance security, it is recommended that each avatar and vector be used only once. Botmaster will change the current account and avatar when a command is issued, and bots will also delete the used vectors. Bots can get updates to the vectors, model and exploits from the C&C server. Therefore, bots can carry at a minimum one vector when being published. Due to various situations, the bots may not be able to run on time. To ensure that the bots can go online as expected, it is suggested that bots are published with more vectors.

Although the Twitter accounts should be used only once, we chose to reuse one account to publish all testing tweets during the experiments due to the limited resources. Because of the contextual tweets, there is no abnormal content remaining on Twitter.

3) Twitter Trends: Twitter trends have the following advantages: 1) While bots cannot find botmasters quickly among Twitter users without hard-coded rules, Twitter trends provide a rendezvous point for them; 2) Twitter trends change with the tweet volume under different topics and are updated every 5 minutes, which is difficult to predict; 3) Since normal users also discuss different topics, botmasters can hide among them. Therefore, Twitter Trends is an appropriate rendezvous point for DeepC2.

4) Hash Collision: The botmaster embeds commands into tweets by hash collision. In this work, we take publishing an IP address (of the C&C server) as an example to illustrate the process of publishing commands. As tweets posted by botmasters have natural semantics, information conveyed by a tweet is insufficient to launch an attack. Therefore, the botmaster can convey the address of a C&C server to bots. Detailed commands and payloads for a botnet campaign and updates of the model and vectors are delivered through the server. Also, the botmaster can convey a domain, shortening-URL or online-clipboard ID to bots in this way.

To convey an IP address to bots through tweets, the botmaster splits the IP address into 2 parts first, as shown in Fig. 6. Each part is expressed in hexadecimal. For each generated tweet, the botmaster calculates its hash and compares whether the first 16 bits of the hash are identical to one IP part. If two parts of an IP address collide, a successful hash collision occurs. The botmaster posts the collided tweets in order. When bots identify botmasters, bots calculate the hashes of tweets posted by the botmaster and concatenate the first 2 bytes of hashes to obtain the IP address. In this way, 16 bits can be conveyed in one tweet. We do not recommend conveying a whole IP address once because it needs too many tweets to launch a successful collision. Two 16 bits will reduce the calculation greatly.

5) Tweets Generation: To perform a successful collision, the botmaster needs numerous tweets. The new tweets are generated using EDA, as described in Sec. II-C. After the selection of a trend, the botmaster crawls the trend tweets to generate more sentences. We crawled 1 K tweets for each selected trend in the experiments. Before we used the crawled tweets to generate new sentences using EDA, we cleaned them first. As there are word deletions and swaps during augmentation, if a tweet is too short, the generated sentences may not contain the trending words. Thus, we filtered out tweets with fewer than 10 words. Additionally, there were retweeted tweets that did not contain the trending words, and we filtered them out and retained only the original tweets. Then, we removed emojis, links, tabs, line breaks and punctuations except for “.”, “!” and “?” in each tweet. Duplicate tweets were removed at last. Normally there were 400 to 900 tweets left. We used EDA to generate 50 sentences for each
remaining tweet, which obtained 20 K to 45 K new sentences. This is still insufficient for a hash collision. We converted all sentences to the upper case and add punctuation ("!", "!!", "!!!", "!", "!!" and "!!!") at the end of each sentence. This resulted in 140 K to more than 300 K sentences in total, which greatly increased the success for a hash collision (see Sec. V-B).

It is not deterministic for a successful hash collision. If a collision fails, the botmaster can add more noise (i.e. other punctuations, typos or form changes) to the sentences or wait until the next appointed time and try again. When more sentences collide for one part, the botmaster can select one randomly or by preferences. The botmaster needs to post the 2 final tweets in order so that the bots can correctly recover the IP address.

6) Authentication: There are different situations that bots cannot go online as expected. If the defenders put on a saved-avatar and post tweets with Sinkholed IP address, the bots may connect to the wrong C&C server. In case of this happening, authentication for bots and C&C servers is mandatory to ensure a secure C&C communication (as in [26]). If the botmaster publishes a domain, shortening-URL or online-clipboard ID, authentication can also be completed by digital signature to the command content.

IV. IMPLEMENTATION

In this section, we demonstrate the proposed convert C&C channel is feasible by presenting a proof-of-concept experiment on Twitter.

A. Data Preparation

1) Twitter Avatars: Twitter provides 4 different sizes of avatars: 48x48, 73x73, 200x200 and 400x400 (see Table IV). Links to these avatars of the same user are similar, and the only difference lies on the sizes. According to a preliminary experiment (described in Appendix A), of all 6 combinations of the 4 sizes, avatar pairs in larger sizes from the same user have a smaller distance. Therefore, we chose to use avatars of size 400x400 to identify botmasters. Bots can obtain links of 400x400 avatars by replacing the suffix in the links.

2) Twitter Trends: In the experiments, Twitter APIs are used to obtain trends, tweets and avatars for botmasters and bots. The Twitter trends API returns the top 50 topics in a chosen area specified by a location ID (WOEID, where on Earth ID). There are detailed tweet volumes if the volume is larger than 10 K over the past 24 hours. The botmaster can design a proper rule to select the trending topics synchronously with bots to obtain enough tweets for EDA and hash collision and hide among normal users. In the experiments, we obtain trends from Johannesburg, South Africa (whose WOEID is 1582504) by the Twitter API and select the last trend above size 400x400 avatars by replacing the suffix in the links.

B. Siamese Neural Network

1) Architecture: As mentioned above, CNN is used in the Siamese network to extract features from images. Fig. 7 shows the architecture of the CNN used in this work. It consists of 4 convolutional layers and 3 fully connected layers. Activation functions between convolutional layers are tanh, and those between fully connected layers are ReLU. To increase the uncertainty of vector generation, we introduce a compression process of the avatars. The avatars are 3-channel JPG or JPEG pictures and are resized to 128x128 before feeding into the model. The CNN model accepts a 3-channel 128-pixel image as the input and generates 128 outputs to make up a feature vector.

![Diagram of CNN](image)

We use the contrastive loss function [20] for the Siamese neural network. For two image inputs of the identical CNNs, $Y$ is a binary label assigned to the pair, where $Y = 0$ represents

| Size    | Link                                                                 |
|---------|----------------------------------------------------------------------|
| 400x400 | https://pbs.twimg.com/profile_images/13544798/9072884255/AaUbc7ao_400x400.jpg |
| 200x200 | https://pbs.twimg.com/profile_images/13544798/9072884255/AaUbc7ao_200x200.jpg |
| 73x73   | https://pbs.twimg.com/profile_images/13544798/9072884255/AaUbc7ao_bigger.jpg |
| 48x48   | https://pbs.twimg.com/profile_images/13544798/9072884255/AaUbc7ao_normal.jpg |
the images to be similar, and \( Y = 1 \) represents that the images are different. \( G_1 \) and \( G_2 \) are two vectors generated by identical CNNs. Let \( D_w = \| G_1 - G_2 \| \) be the Euclidean distance between the vectors, \( w \) be the weights of the network, and \( m > 0 \) be a margin (radius around \( G \)). The loss function is:

\[
L = (1 - Y) \frac{1}{2} (D_w)^2 + Y \frac{1}{2} (\max(0, m - D_w))^2
\]

2) Training: The model was implemented with Python 3.6 and PyTorch 1.5. To train the model, we crawled avatars of different sizes from 115,887 Twitter users and randomly selected 19,137 sets of avatars to build the dataset. We chose avatars of size 400x400 randomly to make up input pairs with label 1. Due to the lack of original pictures of the avatars, we used avatars with sizes of 200x200 and 400x400 from the same user to make up input pairs with label 0. The ratio of input pairs marked as 0 and 1 is 1:2. Finally, we obtained 19,137 “same” image pairs and 38,274 different image pairs. We used 75% of them for training and 25% for testing. The threshold for Euclidean distance was set to 0.02 (see Appendix B). If the distance was lower than 0.02, two inputs were considered to be similar; if higher, they were considered to be different.

3) Performance: To test the performance, we conducted the training process several times. This model converged rapidly during training. After 10-20 epochs, 100% accuracy on the test set was obtained. The size of a trained Siamese neural network model is 2.42 MB. We used avatars from all 115,887 users to make up the validation set, for a total of 463,544 pairs (115,887 pairs with label 0 and 347,657 pairs with label 1, 1:3 in ratio). Evaluations show that the model reached an accuracy of more than 99.999%, with only 2-4 mislabeled pairs. Different from traditional machine learning works, we need to avoid hijacking the botmaster’s accounts, which means mislabeling from not the same to the same (false positive) is forbidden, while some mislabeling from the same to not the same (false negative) can be allowed. The original labels of the mislabeled pairs were all 0, which means no avatar collision occurred with the trained models and ensured security for the botmaster’s accounts.

C. Experiments on Twitter

1) Environments: To simulate bots around the world, we used 7 virtual servers. They were Ubuntu 18.04 x64 servers with 1 GB ROM and 1 vCPU, and were located in Bangalore, Toronto, Amsterdam, Sydney, Tokyo, Dubai, and Virginia, respectively. The code for botmaster was run on another virtual server with the same configuration in San Francisco. Both codes for bots and botmaster were implemented with Python 3.6.

2) Commands and Avatars: We prepared 40 photos taken with mobile phones as avatars for the botmaster’s accounts. The photos were cut to a size of 400x400 and converted into vectors by a trained model. The bots were installed with the model and a vector. In this experiment, the bots and botmaster selected a trend once an hour. Then, the botmaster posted the tweets in 5 minutes, and bots crawled related tweets 5 minutes after the selection. Commands consist of IP addresses from VPSs, private IPs and a threat report [28]. The vector was updated after a successful recognition. In this experiment, each time the botmaster completed a hash collision, it was recorded to a log file. Each time the bots crawled a batch of tweets and started and finished comparisons, they were also recorded in a log file. Afterward, the logs were collected to compare the post time with the retrieval time and to match the original commands with the decoded commands from the bots. Due to the time zone difference between the botmaster and bots, the recorded time was set to UTC time.

3) Results: We sent 47 commands using the 40 avatars. Due to frequent visits to Twitter trends, the selected trends are sometimes the same as the previous ones. Although this does not affect botmaster in real scenarios, in order to more objectively evaluate the success rate of hash collisions, we chose to wait for the next trend instead of replacing the command. All commands in the experiments were received and parsed correctly by the 7 bots. During the tests, the botmaster completed the tweet collection, sentence generation and hash calculation in 13.8 s on average and reached a success rate of 90.28% for hash collisions. After the selection of a trend, bots attempted to crawl 1 K tweets and usually obtained 800-900 nonrepeated tweets (only original tweets were saved for retweeted tweets). Bots needed to crawl the avatars of the tweeters and compared the avatars with the vector to calculate the distances and determine whether the botmaster was found. Due to different network and device conditions, the time this process required varied. The time costs for the bots to find the botmaster are shown in Fig. 8. It takes 5 s to 4.45 min to find the botmaster after crawling the tweets. During the experiments, some of our tweets received several “likes” from ordinary Twitter users, which showed that the sentences generated by EDA did not cause anomalies and were acceptable. After the bots obtained the IPs, the botmaster deleted the posted tweets.

V. Evaluation

In this section, we evaluate the performance of different parts in DeepC2. Environment: The evaluation was conducted
TABLE V
GUIDELINE FOR EFFICIENCY OF TWEETS GENERATION

| Time/s | 1  | 2  | 3  | 5  | 10 | 15 | 20 |
|--------|----|----|----|----|----|----|----|
| Quantity | 10262 | 14232 | 18202 | 26142 | 45993 | 65843 | 85694 |
| Quantity | 10K | 20K | 30K | 50K | 100K | 150K | 200K |
| Time/s | 0.93 | 3.45 | 5.97 | 11.01 | 23.60 | 36.20 | 48.79 |

on an Ubuntu 18.04 x64 virtual server with 1 GB ROM and 1 vCPU on San Francisco, and the code was implemented with Python 3.6.

A. Tweets Generation

To test the efficiency of tweet generation for the botmaster, we selected 79 trends from 4 randomly selected English-speaking areas around the world (San Francisco, London, Sydney, and Johannesburg). One-thousand tweets were crawled for each trend. Additionally, we cleaned the crawled tweets using the same method described in Sec. III-C5 and generated 50 new sentences using EDA for each remaining tweet. As keywords for trends, trending topics may contain one or more words, and with random deletion and random swap adopted in EDA, keywords in topics may be deleted or position changed in the newly generated sentences. If the botmaster posts sentences without accurate keywords, the bots cannot find the botmaster’s accounts from the crawled tweets with the trends. Therefore, the number of sentences with keywords contained was also recorded along with the quantity of all generated sentences. In the 79 selected trends, 55 trends contained only one word, and 24 contained more than one word. The results showed that with the percentage of words in each sentence to be changed set to 0.1, 89.54% of the newly generated sentences contained accurate keywords for the 55 one-word trends, and 77.55% contained accurate keywords for the 24 multiword trends.

The efficiency and the number of sentences are linearly related, as shown in Fig. 9. Table V shows samples of how many sentences can be generated in a given time and the time cost for generating sentences of a given quantity. As mentioned in Sec. III-C5, EDA obtains 20 K to 45 K sentences in this experiment. According to this test, it costs 3 to 10 seconds to generate the sentences. It is acceptable for the botmaster to prepare sentences for a hash collision.

B. Hash Collision

We use the sentences generated above to test the efficiency of hash collisions. To prepare different numbers of sentences, we also follow the method in Sec. III-C5 which converts cases and adds punctuation at the end of each sentence. For each trend, we obtain 4 batches of new sentences incrementally by adding 2 conversions at a time. We also collected 100 C&C server addresses as commands from the threat report [23]. We call a batch of sentences “hit” an IP if the batch succeeds in a hash collision for the IP. We use these new sentences and hashlib in Python 3.6.9 to calculate SHA-256 hashes on the virtual server with a single thread and record the time costs and hit rate of hash collisions with different quantities of sentences.

As shown in Fig. 10, it takes less than 1 second to calculate the hashes. In theory, 65,536 (216) sentences will hit an IP, which is ideal, as a hash collision is probabilistic. The experiment shows that there should be at least 200 K sentences to obtain a 90% hit rate and more than 330 K for a nearly 100% hit rate. As mentioned in Sec. III-C5 there are usually 140 K to more than 300 K sentences for the botmaster to perform a hash collision. This will result in a hit rate of over 75%. During the experiments on Twitter, the botmaster obtains an average of 219,335 sentences for hash collision and reaches a hit rate of 90.28%, which is also acceptable for practical purposes. Moreover, the botmaster can crawl more trend tweets and generate more sentences for hash collisions in real scenarios.

C. Avatar Recognition

To test the efficiency of avatar recognition by bots, we use the 40 vectors above, and 1,000 crawled avatars of size 400x400 to calculate the distances on the virtual server. The average time cost of extracting features from 1 K avatars and calculating 1 K distances is 11.92 s. This is also acceptable for bots in such hardware conditions. In real scenarios, this process may take longer as bots should have avatars crawled first, which varies due to different network conditions. Compared
with the experiments on Twitter (Fig. 8), crawling the avatars is the most time-consuming process during the addressing.

D. Crawling Tweets

In this experiment, bots crawl 1 K tweets 5 minutes after the selection of the trending topic every hour. In real scenarios, attackers can customize the waiting time, crawling volume and frequency to meet their needs. In this experiment, we’ll show how the attackers determine the appropriate parameters.

We also collected trends in Johannesburg and selected the last trend above 10K discussions as the keyword. Then, we use the botmaster’s account to post tweets that contained the keywords. The bots started to look for the botmaster’s account using the keywords by the Twitter API after waiting for 5, 10, 20, 30, 45, 60, 90, 120, 150 and 180 minutes. The bots recorded how many tweets were crawled to find the botmaster. We collected 56 groups of data. Fig. 11 shows the relation between the crawled tweet volume and waiting time. After waiting for 5 minutes, the bots found the botmaster within 1 K tweets in all cases. After waiting for 1 hour, in 88% of cases, the bots found the botmaster within 1 K tweets and 98% within 3 K tweets. Even 3 hours after the botmaster posted the tweets, the bots still found the botmaster within 1 K tweets in 68% of cases and within 3K tweets in 89% of cases. As in the experiments on Twitter, the waiting time of bots is 5 minutes, it is appropriate to crawl 1,000 tweets to find botmaster.

If the attackers choose topics from larger cities such as New York and Los Angeles, tweets may be more frequently updated, requiring the bots to crawl more tweets to find botmasters with the same waiting time. Additionally, if attackers choose top-ranked topics from the trending list, the bots also need to crawl more tweets if the waiting time remains the same. Nevertheless, if attackers choose to publish commands at midnight in the selected city, it is also different from the daytime. The parameters should be customized with different rules and needs when replaying this scenario.

E. Security Analysis

In this section, we discuss the security risks from the perspective of defenders. Defenders can 1) save and reuse avatars from the botmaster’s accounts, 2) train a GAN with the saved avatars to generate similar images, 3) train a corresponding decoder to derive an image that can produce a similar vector, 4) collide an avatar using avatars from real Twitter users, 5) attack the model to let the bot make incorrect decisions on the botmaster, and 6) generate adversarial samples to deceive the model and bots.

1) Save and Reuse Avatars from the Botmaster: Although it is difficult to guess the avatars used by botmasters, adversaries can monitor the behaviors of bots to identify botmasters’ accounts. Adversaries can save the avatars used by botmasters and reuse the avatars when the next appointed time arrives. They can also select a trend and post tweets that contain fake commands. This scenario will not work for bots that are always online because each avatar, vector, and account are used only once. After a command is issued, the bots will delete the stored corresponding vector. Therefore, if adversaries put on botmaster’s avatars, they still cannot be recognized by bots. However, for bots that go online after being offline and missed a command, they will recognize the adversaries’ accounts as botmasters and obtain an incorrect IP. Therefore, authentication for bots and C&C servers is mandatory to ensure secure C&C communication, as stated in Sec. III-C6.

2) Train a GAN: Defenders may try to train a GAN with saved avatars to generate similar images. From the perspective of computational costs in this situation, this is not feasible. The botmaster avatars are diverse and not just human faces. They can be animals, scenery, cars, plants, or art. Additionally, training a GAN needs numerous data. Botmaster avatars are seriously insufficient for building a training dataset. Attacking the C&C in this way is not recommended.

3) Train a Decoder: As adversaries have access to vectors and neural network models, structure and implementation, adversaries can attempt to recover and derive a similar image to cheat the bots. CNN makes the protection possible. CNN learns abstract features from raw images. Each convolution layer generates a higher degree of abstraction from the previous layer. As layers become deeper, much of the information in the original image is lost. This makes it difficult to recover the original image or derive a similar image based on the vectors.

We also simulated such an attack. We assume adversaries treat the neural network as an encoder and build a corresponding decoder to generate related images. Adversaries can also crawl numerous avatars from Twitter and extract feature vectors using the given model. The avatars and vectors make up the training data for the decoder. We trained such a decoder to generate images from vectors. The input of the decoder is a vector of length 128, and the output is an 3-channel PNG image of size 128x128. Then we calculated the distance between the original image and the generated image. Due to the losses introduced by CNN and image conversion, the lowest distance we got is higher than 0.05. As avatars retrieved by bots are in size of 400x400, more conversion and compression will be introduced for the generated 128x128 PNG images. It’s also difficult to attack the C&C in this way.
4) Collide an Avatar: Adversaries can try to collide an avatar. It sounds feasible but is hard practically. We analyzed the composition of the vectors. The 40 vectors mentioned above contain 5,120 numbers. The numbers are sorted incrementally and put into the coordinate system, as shown in Fig. 12. The numbers follow a normal distribution and constitute a continuous interval from -0.350 to 0.264. Each value in the vector is taken from the continuous interval, which is a large space and is hard to enumerate or collide. This ensures the security of the botmaster’s avatars and vectors. However, we still attempted a collision for avatars. We made more than 0.6 billion calculations on the distances between 115,887 pairs of crawled avatars using a trained model. There are 2,050 pairs that have a distance below 0.02 (0.00031%), of which 81 pairs are below 0.01 (0.000012%). By analyzing these pictures, we found that they share similar styles in that they all have a large solid color background, especially a white background (mainly logos) (see Fig. 13). As avatars are prepared by attackers, they can avoid this type of picture. It is also recommended that attackers use colorful pictures taken by their cameras instead of pictures from the Internet.

5) Attack the Model: Adversaries can attack the neural network model to let bots make incorrect decisions on botmaster accounts. There are some works on neural network trojan attacks [29], which make this attack possible. As the target of this attack is a bot (more specific, a neural network model), not a botnet, it may affect some bots but does not influence the whole botnet. Other unaffected bots can still make correct decisions on the botmaster’s accounts.

6) Generate Adversarial Samples: As the model and feature vectors are known to adversaries, it is a white-box nontargeted adversarial attack in this scenario [30]. Adversaries can generate an adversarial sample to fool the model. Adversarial attacks aim at misclassifying the original target. Although the CNN has 128 outputs, they do not represent 128 classes. Each output is a value in the feature vector. A slight perturbation of the feature vector will result in a distance higher than the threshold. Therefore, it is not applicable to attack the botnet in this way.

Even though adversaries happen to obtain an image that produces a similar vector, the size of the botnet can be seized, but the botnet cannot be taken over, as there are authentications between the botmaster and bots on the C&C server (as in [26]). Cryptography authentication is mandatory to ensure the security of C&C communication. All connections, commands and payloads between bots and the C&C server are protected by advanced symmetric keys and asymmetric key pairs. This is hard to break with the current computing capacity.

VI. POSSIBLE COUNTERMEASURES

There are ways to enhance the security of DeepC2, and we discuss the enhancement in Appendix [3]. In this section, we mainly discuss the possible countermeasures.

Behavior Analysis. Traditional malware detection methods such as behavior analysis and traffic analysis can be applied in this case to detect bots [31]. There are periodical behaviors of bots. According to the rules set by attackers, bots need to visit Twitter trends periodically. After the selection of a trending topic, bots need to crawl tweets and avatars to find botmasters. This series of operations can make up a behavioral pattern of bots. In addition, the periodic net flow is also a noticeable feature. In these ways, the bots can be detected.

Collaboration. Taking down a botnet needs cooperation across organizations or even nations. In this scenario, it is recommended that security analysts share the bots samples to the communities and the related OSNs once they appear so that every party can contribute to the mitigating works. OSNs can detect botmasters in real-time by running the samples and actively monitoring activities related to bots and botmasters. They can calculate the distances between the uploaded avatars and vectors, and block the botmasters as soon as the corresponding avatars are detected. This may need a large-scale calculation but is an effective way to mitigate this attack. Meanwhile, OSNs can also help to trace the attackers behind the accounts. Therefore, we believe the cooperation between OSNs and security communities is essential to mitigate this attack.

Improvement on OSNs. As there are many ways to utilize OSNs, OSNs should take measures to avoid being abused. The attackers should maintain some Twitter accounts for C&C. These accounts can be stolen from ordinary users, registered in bulk using automated programs [32], or brought from underground markets [33]. Therefore, we suggest that OSNs apply more complex human-machine verification during the registration, and manage the misbehaved social bots under
the ToS. Cracking down on underground account transactions is also necessary. While working on this work, we found some websites that sell Twitter accounts in bulk. We cannot predict how they got the accounts and how the buyers use the accounts. Since it violates Twitter ToS, related parties should limit illegal account transactions. We have reported it to Twitter.

As AI can be used to launch cyberattacks, security vendors should also consider the malicious use of AI so that the attacks can be detected when they are applied in real scenarios in the future.

VII. RELATED WORKS

A. Building C&C on OSNs

Stegobot [34] uses the images shared by OSN users to build the C&C channel. The social network is regarded as a peer-to-peer network to connect the bots and the botmaster. Information is hiding in images using steganography when the victims are uploading the images to the OSNs. The experiments on Fackbook show that the Facebook uploading process is equivalent to the JPEG compression over the image.

Sebastian et al. [9] proposed to build a covert C&C channel on Twitter. The commands are encrypted tweets with a keyword, for example, #walmart AZEF, where #walmart is the keyword and AZEF is the command cipher. However, this method also has the problem of abnormal contents on OSNs.

Kwak et al. [35] proposed a video steganography-based C&C channel on Telegram, which can transfer large-sized secret files. 500MB of secret data can be hiding in one video file.

Pantic et al. [6] proposed an anomaly-resistant C&C on Twitter. They used tweet-length as a command character and encoded each symbol in commands into numbers from 1 to 140. They collected tweets at different lengths from Twitter. When publishing a command, they chose tweets at specified lengths and posted them. The bots can get the commands by calculating the lengths of the tweets.

B. Malicious use of AI

This work provides a new scenario on the malicious use of AI. There are some works that discuss the combination of AI and attacks.

For covertly delivering malicious payloads, Liu et al. [36] proposed StegoNet to hide malware inside the neural network models. The parameters are replaced by the payloads. With resilience training, value-mapping and sign-mapping, the malware-embedded model’s testing accuracy does not have many decreases.

DeepLocker [37] was proposed to carry out targeted attacks stealthily. DeepLocker trains the target attributes into an AI model and uses the model’s outputs as a symmetric key to encrypt malicious code. The encrypted payload and the AI model are attached to benign applications. Target detection is conducted by the AI model. When the input attributes match target attributes, the secret key will be derived from the model to decrypt the payload and launch attacks on the target.

For detection evasion, MalGAN [38] was proposed to generate adversarial malware that could bypass black-box machine learning-based detection models. A generative network is trained to minimize the malicious probabilities of the generated adversarial examples predicted by the black-box malware detector. More detection evasion methods [39–41] were also proposed after MalGAN.

VIII. CONCLUSION

This paper discussed a novel covert command and control scenario, DeepC2, on OSNs by introducing AI technologies. By utilizing the poor explainability of neural network models, the addressing process can be concealed in AI models rather than exposed as reversible hardcoding. As it is hard to calculate the botmaster’s identifiers in advance, this method is block-resistant to the restrictions on OSNs. For issuing commands covertly, we use easy data augmentation and hash collision to generate contextual and readable command-embedded tweets to avoid abnormal content on OSNs. We have conducted experiments on Twitter to show the feasibility and efficiency. Furthermore, we have analyzed the security of the avatars and discussed ways to enhance the security. We also discussed possible countermeasures to mitigate this kind of attack.

AI is also capable of cyberattacks. With the popularity of AI, AI-powered attacks will emerge and bring new challenges for cybersecurity. Cyberattack and defense are interdependent. We believe countermeasures against AI attacks will be applied in future computer systems, and protection for computer systems will be more intelligent. We hope the proposed scenario will contribute to future protection efforts.

REFERENCES

[1] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, “A survey of botnet technology and defenses,” in 2009 Cybersecurity Applications Technology Conference for Homeland Security, 2009, pp. 299–304.
[2] J. Yin, H. Lv, F. Zhang, Z. Tian, and X. Cui, “Study on advanced botnet based on publicly available resources,” in Information and Communications Security. Cham: Springer International Publishing, 2018, pp. 57–74.
[3] FireEye, “Uncovering a malware backdoor that uses twitter,” FireEye, Tech. Rep., 2015.
[4] L. Stefanko. (2018, June) New telegram-abusing android rat discovered in the wild. [Online]. Available: https://www.welivesecurity.com/2018/06/18/new-telegram-abusing-android-rat/.
[5] M. Faou, “From agent.biz to comrat v4: A ten-year journey,” ESET, Tech. Rep., May 2020.
[6] N. Pantic and M. I. Husain, “Covert botnet command and control using twitter,” in Proceedings of the 31st Annual Computer Security Applications Conference, ser. ACSAC 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 171–180.
[7] A. Moscaritolo. (2009, Aug) Twitter used as botnet command-and-control hub. [Online]. Available: https://www.itnews.com.au/news/twitter-used-as-botnet-command-and-control-hub-153144.
[8] A. Singh, “Social networking for botnet command and control,” 2012.
[9] S. Sebastian, S. Aryapun, and P. Vinod, “Framework for design of graybot in social network,” in 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2014, pp. 2331–2336.
[10] F-Secure, “The dukes: 7 years of russian cyberespionage,” F-Secure, Tech. Rep., 2015.
[11] W. Mercer, P. Rasagneres, and M. Molyett. (2017, Apr) Introducing rokrat. [Online]. Available: https://blog.talosintelligence.com/201704/introducing-rokrat.html
| layer | size-in | size-out | kernel | param |
|-------|---------|----------|--------|-------|
| conv1 | 32×32×3 | 28×28×6  | 5×5×6  | 1 0.5K |
| ReLU  | 28×28×6  | 14×14×6  | 2×2×1  | 0     |
| pool1 | 14×14×6  | 10×10×16 | 5×5×16 | 1 2.4K |
| conv2 | 10×10×16 | 5×5×16   | 2×2×1  | 0     |
| Sigmoid | 5×5×16 | 1×1×1    | 0      |
| pool2 | 5×5×16   | 1×1×1    | 0      |
| fc1   | 1×400×1  | 1×256×1  | 1×1×1  | 0 102.7K |
| Sigmoid | 1×256×1 | 1×1×1    | 0      |
| fc2   | 1×192×1  | 1×1×1    | 0      |
| output | 1×192×1  | 1×1×1    | 0      |
| total  | 192×1    | 1×1      | 0      | 179.6K |

**TABLE VI** ARCHITECTURE OF CNN

[12] T. Lancaster and E. Idrizovic. (2017, June) Paranoid plugx. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-paranoid-plugx/.

[13] J. Grunzweig. (2018, Jan) Comnie continues to target organizations in east asia. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-comnie-continues-target-organizations-east-asia/.

[14] F. LeCun, K. Gade, S. C. Geyik, K. Kenthapadi, V. Mithal, A. Taly, R. Guidotti, and P. Minervini. (2020, Feb) Explainable ai: Foundations, industrial applications, practical challenges, and lessons learned. [Online]. Available: https://xaitutorial2020.github.io/

[15] S. Qiu, Q. Liu, S. Zhou, and C. Wu, “Review of artificial intelligence and adversarial attack and defense technologies,” in 25th USENIX Security Symposium (USENIX Security 16). Austin, TX: USENIX Association, Aug. 2016, pp. 263–278.

[16] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, “A comprehensive measurement study of domain generating malware,” in 25th USENIX Security Symposium (USENIX Security 16). Austin, TX: USENIX Association, Aug. 2016, pp. 263–278.

[17] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, “Signature verification using A “siamese” time delay neural network,” Int. J. Pattern Recognit. Artif. Intell., vol. 7, no. 4, pp. 669–688, 1993.

[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

[19] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06), vol. 2, 2006, pp. 1735–1742.

[20] J. W. Wei and K. Zou, “EDA: easy data augmentation techniques for boosting performance on text classification tasks,” in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, 2019, pp. 6381–6387.

[21] ATT&CK. (2020). Online. Available: https://twitter.com/MITREAttack/status/1267815021301633024

[22] ATT&CK. (2020). Online. Available: https://www.google.com/search?q=buy+twitter+accounts

[23] Google. (2021) Google search. [Online]. Available: https://www.google.com/search?q=buy+twitter+accounts

[24] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov, “Stegobot: A covert social network botnet,” in Information Hiding, T. Filler, T. Pevny, S. Craver, and A. Ker, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 299–313.

[25] M. Kwak and Y. Cho, “A novel video steganography-based botnet communication model in telegram sns messenger,” Symmetry, vol. 13, no. 1, 2021.

[26] T. Liu, Z. Liu, Q. Liu, W. Wen, W. Xu, and M. Li, “Stegonet: Turn deep neural network into a stegomallware,” in Annual Computer Security Applications Conference, ser. ACSAC ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 928–938. [Online]. Available: https://doi.org/10.1145/3427228.3427268

[27] D. Kirat, J. Jang, and M. P. Stoecklin, “Deeplockerr: concealing targeted attacks with ai locksmithing,” IBM Research, Tech. Rep., 2018.

[28] W. Hu and Y. Tan, “Generating adversarial malware examples for black-box attacks based on GAN,” CoRR, vol. abs/1702.05983, 2017. [Online]. Available: http://arxiv.org/abs/1702.05983

[29] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, “Learning to evade static PE machine learning malware models via reinforcement learning,” CoRR, vol. abs/1801.08917, 2018. [Online]. Available: http://arxiv.org/abs/1801.08917

[30] J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, “Black-box adversarial attacks against deep learning based malware binaries detection with GAN,” in ECAI 2020 – 24th European Conference on Artificial Intelligence, Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), ser. Frontiers in Artificial Intelligence and Applications, vol. 325. IOS Press, 2020, pp. 2536–2542. [Online]. Available: https://doi.org/10.3233/FAIA200388

[31] J. Wang, Q. Liu, D. Wu, Y. Dong, and X. Cui, “Crafting Adversarial Example to Bypass Flow-&ML- based Botnet Detector via RL,” in To be appear in RAID ’21: International Symposium on Research in Attacks, Intrusions and Defenses. ACM, 2021.

[32] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, “The rise of social bots,” Communications of the ACM, vol. 59, no. 7, pp. 96–104, 2016.

**APPENDIX**

### A. Selection on Avatar Size

As mentioned in [IV-A1], Twitter offers 4 different sizes of avatars (48x48, 73x73, 200x200 and 400x400). We need to use avatars from the same user to make up the training set with label 0. There are 6 combinations of the 4 sizes. To better serve the training set, an avatar combination should have a shorter distance as they come from the same user. Therefore, 256 avatars from 64 users are selected to calculate the distances using a CNN modified on LeNet (see Table VI).
A total of 256 vectors are extracted from the avatars by the CNN with random weights. For each user, there are 4 vectors that represent 4 different sizes of avatars. Then, Euclidean distances of different vectors from the same user are calculated. Now there are 6 distances for each user. For simplicity, we use $D_{n,m}$ to represent the distance between avatars of size $n \times m$ and $m \times m$. The experiment was repeated 14 times with 14 different random weights for the CNN.

Fig. 14 shows the 14 average distances of the 6 size combinations. This indicates that when one avatar is fixed in size, a smaller size gap gains smaller distances (e.g., $[D_{400,73}, D_{200,73}, D_{73,48}]$), and avatars with a larger size gain smaller distances (e.g., $[D_{400,200}, D_{73,48}]$). Therefore, avatars with sizes of 200x200 and 400x400 are used in the training set because they have the shortest distance of all the combinations, and avatars with sizes of 400x400 are used by bots to find botmasters.

B. Threshold for Distance

A threshold is needed to determine whether two avatars share the same source. We use a trained model to calculate the distances on the validation set, which contains 115,887 pairs with label 0 and 347,657 pairs with label 1. We record the distances of every comparison, sort them by value and label, and count their frequencies to learn the boundary between the "same" avatars and different avatars. As shown in Fig. 15, the distances of all pairs with label 1 and only 4 pairs with label 0 are larger than 0.02, and the remaining pairs with label 0 are less than 0.02. It shows that 0.02 is a proper threshold for the determination. In real scenarios, attackers can choose a threshold less than 0.02, as the undistributed avatars and distances are within the authority of attackers.

C. Enhancement

As proof of concept, the parameters in this work are conservative. There are ways to enhance the security of DeepC2.

In the design of bots, the vectors can be longer than 128. It makes analysis and collisions for avatars even more difficult. The threshold of distances can also be lower than 0.02, as the undistributed avatars and the distances are within the authority of attackers. They can balance the efficiency and accuracy according to the needs. Additionally, more losses can be introduced during the processing of avatars, like compression, deformation, format conversion, etc. This makes it harder to recover the avatars.

For addressing, the botmaster can select more topics. Botmasters can publish commands on the topics, and bots can choose one randomly to find botmasters. As shown in Table III, attackers can also use other fields in OSNs to convey customized contents. For instance, attackers could comment on a tweet, and bots would identify botmasters and obtain commands from botmasters’ profiles. Other platforms, like Weibo and Tumblr, can also be utilized.

As stated before, attackers should maintain some accounts to publish different commands. To reduce the specious behaviors of accounts, attackers can maintain them by imitating normal users or social bots [32]. This work can be done manually or automatically [32]. When attackers need to publish a command, attackers can select one account and maintain other accounts as usual.