Chatbot Security and Privacy in the Age of Personal Assistants

Winson Ye
Computer Science Department
College of William and Mary
Williamsburg, VA
wye@email.wm.edu

Qun Li
Computer Science Department
College of William and Mary
Williamsburg, VA
liqun@cs.wm.edu

***Abstract*—The rise of personal asistants serves as a testament to the growing popularity of chatbots. However, as the field advances, it is important for the conversational AI community to keep in mind any potential vulnerabilities in existing architectures and how attackers could take advantantage of them. Towards this end, we present a survey of existing dialogue system vulnerabilities in security and privacy. We define chatbot security and give some background regarding the state of the art in the field. This analysis features a comprehensive description of potential attacks of each module in a typical chatbot architecture: the client module, communication module, response generation module, and database module.**

***Index Terms*—dialogue system, chatbot, personal assistants, conversational response generation, conversational AI, chatbot security, NLP security, adversarial text generation, IoT security**

I. Introduction

Chatbots are achieving impressive feats in modern times. Everyday, millions of people use personal assistants like Siri in order to complete tasks such as booking flights or finding good restaurants. They have fundamentally changed the way that humans interact with computers for the better. However, all this progress in developing human like dialogue systems must also be met with optimistic caution. Indeed, members of the chatbot community must be vigilant about possible security vulnerabilities in these chatbots, especially as they are tasked with more and more critical tasks as time goes on. People who interact with chatbots perceive them to be almost human, so they may be more open to giving them personally identifiable information. This presents possible privacy risks.

In order to present the community with a detailed overview of chatbot vulnerabilities, we survey possible attacks as well as their potential solutions. First, we must define what a chatbot is. Quite simply, a chatbot is any computer application designed to hold natural conversations with human users. This definition is broadly construed so as to capture the wide variety of chatbots available today, including personal assistants like Siri or customer service chatbots on the front page of many prominent companies like Amazon. Next, we must define the typical chatbot architecture:

1) Client Module: the part of the chatbot that the user interacts with along with all the applications the chatbot can control.
2) Communication Module: the infrastructure that transmits user messages from the client module to the response generation module and from the response generation module to the database module.
3) Response Generation Module: the program responsible for actually understanding the input message and generating an appropriate response for the user.
4) Database Module: the place where all the data relevant to a conversation is stored, such as message history, photos, and user preferences.

To better understand how this architecture works, we can follow the path that a message takes from the client module all the way to the database module. First, the user logs on to the platform the chatbot is hosted on. This could be an app on their phone, a website, or a smart device. These are the client side interfaces that the user will craft their messages on. Other important functions on the client side include user authentication and voice recognition as appropriate. Once the message is sent, the communication module is responsible for transporting the text from the client side to the response generation module. Here, developers focus on encrypting and authenticating communications by using secure protocols like HTTPS. This module is also responsible for monitoring traffic to detect suspicious activity such as an imminent DDoS attack. After arriving at the server side, the chatbot must interpret the message and generate a reply. Interpreting the message either implies that NLP algorithms will be used to parse the sentences into useful fields, or a neural network will convert the message into some abstract vector representation using word embeddings. To generate the appropriate message, either a neural network will directly synthesize one using a seq2seq model that learns to map an input sentence into the most appropriate output sentence, or the optimal response is selected from a repository of candidate responses. Once the response generation module has produced the reply, the communication module transports it back to the client module. At the same time, the database module stores information related to the current chatbot interaction. For example, the response generation module can make use of a knowledge graph stored in the backend. User preferences may be stored here as well, such as which conversational topics the current user responds most positively to.

II. Client Module

The client module is the user experience side of the chatbot architecture. It is primarily responsible for executing the chatbot's agenda and receiving user input. Other functions embedded in this module include authentication, companion apps, and voice recognition for personal assistants. For example, a customer service chatbot may use a website as its client module.

A. Unintended Activation Attacks

Personal assistants are always collecting voice data. In theory, however, they usually listen for a wake-up command such as "Hey Siri" before they record any actual conversations. The conversations that are recorded can be sent up to the cloud to improve personalized advertising or machine learning algorithms. In the ideal situation, the user has complete control over the data that personal assistants collect.

However, there are a number of ways things could go wrong. Sometimes the wake-up phrases that these personal assistants use can be confused with other words in regular conversation. In this case, the user could have their conversations recorded unintentionally. Moreover, other people in the same room but who are not the user can also have their voices recorded during a personal assistant session [1]. Humans do not even have to be present in the room to trigger a personal assistant's recording capabilities. An adversary could play a recording or talk to the personal assistant through another infected ioT device.

All of these attacks exploit the personal assistant's method of turning on and recording conversations. If successful, these attacks could seriously violate user privacy. Fortunately, there are countermeasures. To prevent remote activation, Lei et al. [2] have developed tools to determine whether there is an actual human nearby by using Wifi signals within the home to detect human motion. Microsoft's Xiaoice uses a human-to-bot classifier to determine whether a human is talking to the chatbot. Otherwise, Xiaoice may pick up conversations the user is having with other people. [3].

B. Faked Response

Many users have preconceived notions about how Alexa and Google Home work. For example, Zhang et al. [4] found that almost half of the 156 users in their user study tried to switch from one skill to another in the middle of an interaction. Furthermore, 30% had trouble turning off their personal assisants and 78% did not use the LED indicator on the personal assistant to check for proper termination. As such, an adversary can develop malicious skills that fake responses to take advantage of these user misunderstandings.

Let us consider possible attacks in action. Imagine that the user has downloaded a malicious skill that is advertised as an airplane ticket reservation system. On its own, this malicious skill could only collect the user's travel details without arousing suspicion. However, once the user decides to switch to a workout skill, the malicious app could pretend to hand control over by uttering a fake response, such as "Sure, here is Workout App." At this point, the adversary is free to collect the user's health information.

Another attack is to fake termination. As long as the skill says a response like "goodbye" to the user, everything will seem okay and safe on the outside. With the user now tricked into thinking the app is closed, the skill can continue recording the user. It is important to note that the personal assistant will forcefully terminate an app at some point without audio input, but Zhang et al. bypass this by including a silent audio file in their attack to feed to the personal assistant.

Developers can develop countermeasures against these attacks by developing a blacklist of suspicious personal assistant responses. Zhang et al. take such an approach. They perform fuzzy matching to determine the similarity between a given personal assistant response and the ones on the blacklist. If the similarity exceeds a certain threshold, then the personal assistant is notified and verification procedures are executed to ensure that the skill in question is legitimate.

C. Access Control Attacks

Malicious apps on personal assistants could take advantage of the loopholes in the permissions system in an IoT network to control other devices during a coordinated attack. Since personal assistants usually lie at the command center of many IoT networks, this kind of attack could be particularly devastating.

For example, the malicious app may be disguised as a home monitoring app. As such, it may be given several coarse grained permissions such as the ability to disable security cameras. The attacker can take advantage of this and disable the cameras when the user has left home [5]. At that time, a robber could infiltrate and steal valuable belongings. Another example is a temperature monitoring app that can open the windows when the temperature reaches a certain point. Similarly, a robber can take advantage of this by instructing the app to open the windows when the user is sleeping.

One solution to these access control attacks is simply to adopt defensive coding strategies. Permissions should be granted very carefully and narrowly so that no one app can wreck havoc on someone's home if they turn out to be malicious. However, human error is inevitable and this defense may not always work. As such, there are automated approaches to defense. For example, Jia et al. develop ContexIoT [5], a program that automatically profiles IoT apps for suspicious behavior. The main idea behind their work is that each security sensitive action executed by the app must get approval from the user first. For example, in the aforementioned temperature app scenario, ContexIoT will notice that the user has not specifically authorized for the windows to be open when they are asleep before. As such, the user will be prompted to authorize this action, thus revealing the adversary's motives.

D. Adversarial Voice Samples

Many personal assistants rely on a voice recognition feature embedded in the client module to function properly. However, these voice recognition features are not always perfect. There have been incidents in the past where hackers could fool personal assistants into doing their bidding.

To fool the voice recognition module embedded in the client program, one can craft adversarial voice samples. Overall, there are two kinds of attacks one can pull off: white box attacks and black box attacks. White box attacks rely on some knowledge from the model, whereas black box attacks assume nothing about the model. Either way, the main goal is to disguise the adversarial voice command that the attacker is trying to evoke. One way of accomplishing this is to introduce a perturbation to the original voice sample such that it is misinterpreted by the voice recognition module but imperceptible to human beings. More sophisticated attacks may embed voice commands into seemingly innocent audio sources like songs [6]. Here, the authors perturb the innocuous song just enough so that the voice recognition module can make out the adversarial voice commands but human beings cannot.

Countermeasures against adversarial samples in the voice domain have been proposed. White box attacks are quite easy to foil simply by keeping the model secret. However, adversarial samples can still be crafted even in black box settings, so overreliance on the secrecy of the model is bad practice. Other measures focus on increasing the amount of work needed to pull off a successful attack. If the adversary has to query the model many times, the attack may become unfeasible. Additionally, the developers can retrain the model. Adversarial training can correct the voice recognition module's misunderstandings.

III. Communication Module

Next, we will discuss the communication module. On the whole, this module serves two functions: 1) transporting messages from the client program to the response generation module and 2) fulfilling data requests sent from the response generation module to the database module. All the layers of the typical networking stack are in play here, but we will focus primarily on the top level layers, such as the application layer and transport layer.

A. Wiretapping

Even if traffic in the communication module is fully encrypted, an adversary may still be able to extract information from seemingly harmless metadata.

Consider Amazon Alexa. Attackers are still able to deduce with confidence significantly greater than chance what kind of voice command is used even if the data is encrypted before being sent to the cloud. Indeed, Kennedy et al. [7] describe one such attack. Features such as the packet size and number of bytes transmitted can be used to determine the corresponding voice command for a given traffic trace. This generic information is easily obtained through a packet sniffer like Wireshark. The classification algorithm itself can be formulated as a machine learning problem, and the authors demonstrated that naive bayes classifiers or support vector machines are reasonable choices.

There are a number of countermeasures in place to make sure that packet sniffing does not reveal sensitive information to hackers. For example, Buffered Fixed-Length Obfuscation [8] aims to send packets of a fixed size at fixed intervals. This way, hackers cannot discern any kind of discriminative pattern from the network traffic. However, the overhead for this approach in the personal assistant domain is high. As such, finding an efficient way to prevent hackers from extracting important information from encrypted communications is still an open research problem.

B. MitM Attacks

Man in the Middle (MitM) attacks can intercept messages between client A and client B and replace them with the adversary's own malicious messages. These attacks could incite violence by provoking the human users, change political opinion through social engineering attacks, or spam both clients.

The key challenge behind these attacks is to develop a believable modification of the original messages. Lauinger et al. [9] describe how their work can accomplish this using their man in the middle, Honeybot. Honeybot makes the conversation seem natural by engaging the users in a topic before making any modifications to messages. For example, Honeybot may ask certain questions such as "What are your favorite movies?" Once the conversation has developed enough, Honeybot can begin its spamming attacks by inserting malicious links. This will seem natural because the users are sharing links to their favorite movies anyway. Honeybot can delete and insert messages of its own as well.

Detecting these attacks can be challenging. In cases where there is a sophisticated agent acting as a man in the middle, the conversation may seem very natural. In these scenarios, it is best to adopt encryption and authentication protocols to secure the traffic between the client module and the response generation module. Despite the risks, many chatbots still do not adopt these best practices.

C. DDos Attacks

DDos attacks aim to prevent the chatbot from interacting with users by flooding the server with requests. For companies that rely on chatbots to serve customers, a few hours of downtime may cause catastrophic damage. In order to conduct such an attack, attackers usually have to gather a large number of compute resources, usually by infecting computers with a virus and forcing them to join a malicious network known as a botnet.

There are several ways hackers can launch DDos attacks. One method is to simply clog the network with meaningless traffic, thus dramatically slowing down the server's response time. Another method is to craft adversarial packets that contain the same IP address for both source and destination fields. Yet another technique is to ask the dialogue agent to generate a very long and verbose response to some kind of query. For example, the hacker could ask the response generation module to tell it a story. If multiple malicious computers make this same request and they trick the server into sending all of these replies back to an unsuspecting user, the resulting traffic will completely clog all communication channels.

Since DDos attacks are not unique to chatbots, researchers have had a long time to come up with a solution to this problem that plagues any application that requires networking in general. One technique is to measure the statistical properties of the packets being sent over the network [10]. The crucial intuition here is that administrators can use past information to guide their threat detection systems. For example, one can capture the distribution of source IP addresses and compare this to the past distribution. If the match is high, then it is likely that the current traffic is legitimate. Otherwise, someone may be performing a DDos attack.

IV. Response Generation Module

Let us now discuss the response generation module. This module is primarily responsible for interpreting the user message and generating an appropriate reply. Incorporated in these modules could be policy planning algorithms, domain specific dialogue models and affective computing modules that give the chatbot emotional intelligence.

A. Out of Domain Attacks

Consider a chatbot that is trained very well on a few domains but lacks authoritative knowledge on other domains. An adversary could systematically find these weak points in the chatbot by either brute force attacks or even by developing another ”probing neural network” that can estimate the confidence the dialogue model has in its response. We will refer to this kind of attack as an ”out of domain” attack.

If hackers can exploit out of domain attacks successfully, major damage could be done as the chatbot’s behavior for these edge cases may be highly unpredictable. For example, if an adversary finds out that a customer service chatbot designed for flight booking cannot handle car rentals well, then hackers could trick users into divulging personal information to rent a car only to have their information stolen by the hackers.

To defend against these attacks, there must be some way for the chatbot to monitor its own confidence of a response. For example, a detector could be trained to classify certain requests as in domain or out of domain. However, the research challenge here is in coming up with the training data as manual labeling is time consuming and expensive. Zheng et al. [11] tackle the problem by automatically generating pseudo out of domain requests from normal in domain requests. Alternatively, Liu et al. [12] propose a technique to improve how deep neural networks can quantify uncertainty by improving the model’s ability to quantify how far a given test example is from other training points in the input space.

B. Adversarial Text Samples

Adversaries can directly attack the response generation module itself by crafting clever input messages. These input messages may cause the chatbot to respond with false information or use offensive language.

Adversarial attacks directed at dialogue systems usually craft an input sentence that can break the chatbot. Liu et al. [13] use a reinforcement learning approach to tackle this research challenge. In their work, the authors create a ”reverse dialogue generator.” This agent is responsible for predicting the correct input sentence that triggered the corresponding output sentence. After training, this agent should be able to produce an input sentence that will lead to the dialogue model responding with the desired output sentence specified by the attacker.

To address adversarial inputs, researchers have been proposing a variety of solutions. To prevent attackers from coercing the chatbot to use offensive language, one simple approach is to employ a hate speech detector. Here we define ”hate speech” broadly as any negative language that fosters animosity between users. Any sentences that contain hateful words will automatically be filtered out. In practice, however, hate speech detectors are not perfect and have well known vulnerabilities. For example, one does not have to use swear words in order to craft a hateful message. As such, simple keyword matching is insufficient and never exhaustive. A more sophisticated approach is proposed by Dinan et al [14]. In their work, they propose a neural network classifier that can distinguish between hate speech and safe speech. It is based on BERT, a powerful general purpose language model. The model was fine tuned on the Wikipedia Toxic Comments dataset. To make the detector robust against clever messages that may change the spelling of swear words or use unconventional insults, the authors ask crowdworkers to break the detector, and then they feed these adversarial examples back to the model to improve its robustness.

C. Language Model Attacks

The state of the art in NLP systems calls for the use of pretrained language models. Recently, these large models have achieved impressive feats in areas such as neural machine translation, sentiment classification, and toxicity detection. However, their ubiquitous presence makes it all the more important that these language models are secure enough for live chatbots to use.

One way attackers could exploit the chatbot community’s current dependence on language models is by crafting adversarial models that can cause the NLP system to malfunction in very specific ways [15]. These malicious language models can make their way into the chatbot in the development process if the programmer does not take the time to vet each model. This is a very sophisticated attack because the language model will have no perceivable effect on normal text messages. However, for certain triggers, the chatbot will exhibit suspicious behavior. For example, given certain input sentences, the chatbot may respond with offensive language.

To protect against adversarial language models, the simplest solution would be to adopt best practices in chatbot development. Before integrating any language model into the chatbot itself, extensive verification procedures must take place to ensure that the language model has been disseminated by a legitimate organization. In addition, future updates to this language model have to be verified as well. Another more sophisticated approach would be to develop a self defense algorithm. For example, this algorithm would have to be able to search for trigger words that may coerce the chatbot into using offensive language. Once these trigger words are detected, the chatbot can either be retrained to cleanse itself of the triggers or the language model can be thrown out entirely.

D. Adversarial Reprogramming

A dedicated adversary can repurpose the response generation module to perform another task without changing the model parameters at all. For example, the module may contain a hate speech detector. If an attacker can convert this hate speech detector into a social vulnerability detector, then a terrorist can use this detector to target people who will be vulnerable to radicalization.

Neekhara et al. [16] describe one approach to apply adversarial reprogramming to the text classification domain. In their black box attack, they define a reinforcement learning agent that learns to feed carefully crafted input sequences to the model. The output of the classifier is then remapped according to some user defined function, and the final output is used as the answer for the adversarial task.

It is difficult to mount a defense against adversarial reprogramming attacks because they are fundamentally different from standard adversarial attacks. For example, the goal in adversarial reprogramming is not to coerce the neural network to make a mistake, but rather to repurpose the model for an adversarial task. Nevertheless, the adversary still has to put in some effort in order to pull off the attack successfully. One avenue of defense is to increase the number of queries needed for the hacker to learn the classification patterns of the model.

E. Feedback Engineering Attacks

Feedback engineering attacks take advantage of the response generation module's ability to learn from user feedback. Many chatbots are designed such that they can hold more engaging conversations with users as time progresses because they will learn user preferences when it comes to conversation topics. However, a hacker could take advantage of this by generating feedback that pushes the chatbot in the wrong direction, such as towards generating hate speech.

There are two primary ways chatbots are designed to improve from user interaction: 1) through retraining or 2) through reinforcement learning. Let us consider attacks aimed at chatbots that retrain themselves first. Chen et al. [17] detail one such approach. Their aim is to retrain NLP systems such that normal queries can be answered as usual, but input sentences with certain trigger words will cause suspicious behavior. The key challenge here is to ensure that these triggers do not alert the user to any suspicious activity. Other attacks focus on reinforcement learning models. Zhang et al. [18] describe one possible approach for such an attack. Essentially, the hacker is assumed to have white box access to the chatbot. They can then add a perturbation to the reward signal propagated by the environment. This selective perturbation can push the agent towards learning a policy of the hacker's choosing.

To prevent the chatbot from being poisoned by malicious training examples, researchers could adopt a variety of solutions. For example, instead of directly retraining on the messages themselves, the developers can separate the response generation module and the training examples located in the database module with a layer of abstraction [19]. To prevent attacks against a reinforcement learning agent, developers must note that in practice there are limitations to how long the adversary can perturb the rewards received by the dialogue agent. If the environment changes or the agent is taken down for updates, the adversary will have failed to carry out the poisoning attack.

V. Database Module

We will now discuss the database module. Here, the chatbot can look up any relevant information to the conversation. For example, it can query a knowledge graph in order to generate a well informed response. Attacks to the database module could compromise the privacy of millions of users and may change the behavior of the chatbot catastrophically. For example, a modification of the credit scores in the database module may cause a banking chatbot to reject a customer's application for credit immediately.

A. SQL Injection Attacks

One key vulnerability in many applications that use SQL as a datastore is an injection attack. These attacks rely on carefully crafted inputs to coerce the database into performing unintended operations such as modifying information or returning sensitive information.

Halfond et al. [20] explain that injections can occur through server variables or even cookies. For example, an attacker could hide an injection attack in a server variable that gets triggered whenever the database is ordered to log the contents of that variable.

To combat SQL injection attacks, a variety of solutions can be adopted. However, the main cause of this vulnerability is lack of input validation. Thus, any robust solution to these attacks requires at the very least that the developer spend sufficient time cleaning and validating data. More sophisticated prevention techniques employ static and dynamic analyses or machine learning algorithms. For example, Huang et al. [21] propose WAVES, an algorithm that searches the application for any places an injection attack can occur and then builds possible adversarial inputs based on known attack patterns. Before the final application is deployed, the developers must be able to withstand the attacks generated by WAVES.

B. Knowledge Graph Attacks

Some chatbots use a special database known as a knowledge graph in order to reason about the world around them. Knowledge graphs represent the relationships between real world entities, with the most straightforward example being a social network graph. Sometimes developers embed these knowledge graphs into a vector space in order to reason about the relationships between entities in a more mathematically sound way.

To perform this embedding, a neural network is trained to project the raw knowledge graph values into reasonable vector representations. Zhang et al. [22] take advantage of this fact to develop an attack wherein they poison the training data of the embedding such that certain facts can be added or forgotten. For example, imagine that the chatbot in question is a customer service chatbot for Amazon. The attacker is from Company X. This chatbot uses a knowledge graph in order to make recommendations to the user. Suppose the knowledge graph originally shows a strong match between the products the user likes and Company Y's products. To sabotage the competition, the attacker could delete any facts about Company Y's products in the knowledge graph and replace them with positive facts about Company X.

Standard poisoning prevention algorithms can help prevent some knowledge graph attacks as well. For example, one technique is to apply data sanitization. On a high level, this simply means analyzing trends in the clean data and removing any outliers that are indicative of poisoned training examples. However, this approach is not perfect and a clever adversary can circumvent this solution by placing poisoned data points as close as possible to the clean data.

VI. Conclusion

This survey has focused on analyzing possible security and privacy vulnerabilities in the chatbot architecture. Unfortunately, the chatbot community has not yet developed comprehensive standards for securing chatbots. Our hope is that this survey can be used to further the objective of developing secure next generation chatbots. Future work can focus on discovering attacks that span multiple modules of the chatbot architecture as well as following up on the proposed solutions to each security vulnerability detailed in this survey. Additionally, a more thorough analysis of the security of state of the art NLP mechanisms such as the attention mechanism used in transformers needs to be conducted. In the future, it is clear that chatbots will only take on an increasingly important role in society, serving in roles such as personal assistants, companions, or even therapists. It is imperative that developers conduct a full analysis of the security of their chatbot before deployment in order to avoid significant damage. Ultimately, we are optimistic that as long as the lessons laid out in this survey are followed that the benefits of chatbot technology will vastly outweigh the cons.

VII. Acknowledgement

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grant CNS-1816399. This work was also supported in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation and workforce development. For more information about CCI, visit cyberinitiative.org.

References

[1] H. Chung, M. Iorga, J. Voas, and S. Lee, "Alexa, can i trust you?" *Computer*, vol. 50, no. 9, pp. 100–104, 2017.

[2] X. Lei, G.-H. Tu, A. X. Liu, C.-Y. Li, and T. Xie, "The insecurity of home digital voice assistants-vulnerabilities, attacks and countermeasures," in *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2018, pp. 1–9.

[3] L. Zhou, J. Gao, D. Li, and H.-Y. Shum, "The design and implementation of xiaoice, an empathetic social chatbot," *Computational Linguistics*, vol. 46, no. 1, pp. 53–93, 2020.

[4] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Dangerous skills: Understanding and mitigating security risks of voice-controlled third-party functions on virtual personal assistant systems," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1381–1396.

[5] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms." in *2017 NDSS Symposium*.

[6] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 49–64.

[7] S. Kennedy, H. Li, C. Wang, H. Liu, B. Wang, and W. Sun, "I can hear your alexa: Voice command fingerprinting on smart home speakers," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 232–240.

[8] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE symposium on security and privacy*, pp. 332–346.

[9] T. Lauinger, V. Pankakoski, D. Balzarotti, and E. Kirda, "Honeybot, your man in the middle for automated social engineering." in *USENIX*.

[10] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to ddos attack detection and response," in *Proceedings DARPA information survivability conference and exposition*, vol. 1. IEEE, 2003, pp. 303–314.

[11] Y. Zheng, G. Chen, and M. Huang, "Out-of-domain detection for natural language understanding in dialog systems," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 1198–1209, 2020.

[12] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan, "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness," in *ICML*, 2020.

[13] H. Liu, T. Derr, Z. Liu, and J. Tang, "Say what i want: Towards the dark side of neural dialogue models," *arXiv preprint arXiv:1909.06044*, 2019.

[14] E. Dinan, S. Humeau, B. Chintagunta, and J. Weston, "Build it break it fix it for dialogue safety: Robustness from adversarial human attack," in *ACL 2019*.

[15] X. Zhang, Z. Zhang, and T. Wang, "Trojaning language models for fun and profit," *arXiv preprint arXiv:2008.00312*, 2020.

[16] P. Neekhara, S. Hussain, S. Dubnov, and F. Koushanfar, "Adversarial reprogramming of text classification neural networks," in *ACL 2019*.

[17] X. Chen, A. Salem, M. Backes, S. Ma, and Y. Zhang, "Badnl: Backdoor attacks against nlp models," *arXiv preprint arXiv:2006.01043*, 2020.

[18] X. Zhang, Y. Ma, A. Singla, and X. Zhu, "Adaptive reward-poisoning attacks against reinforcement learning," *arXiv preprint arXiv:2003.12613*, 2020.

[19] B. Hancock, A. Bordes, P.-E. Mazare, and J. Weston, "Learning from dialogue after deployment: Feed yourself, chatbot!" in *ACL 2019*.

[20] W. G. Halfond, J. Viegas, A. Orso *et al.*, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE international symposium on secure software engineering*, vol. 1. IEEE, 2006, pp. 13–15.

[21] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai, "Web application security assessment by fault injection and behavior monitoring," in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 148–159.

[22] H. Zhang, T. Zheng, J. Gao, C. Miao, L. Su, Y. Li, and K. Ren, "Data poisoning attack against knowledge graph embedding," in *IJCAI 2019*.