Advanced Cyber Deception Framework (ACDF): A Comprehensive Study

Mark Maldonado† Caleb Johnson Manbir Gulati Todd Jaspers Paul F. Roysdon*

***Abstract*—Deception frameworks provide an effective environment for data collection on cyber criminals. Using deception techniques these frameworks help security professionals identify and deceive attackers. Information security is an increasingly complex problem as cyber attacks evolve and attackers become more competent in exploitation. Honeypots or honey networks provide an opportunity for counter-intelligence collection. The current State-of-the-Art (SotA) honeypot deployments are easily identified and cataloged by adversaries. Using machine learning, specifically a Tabular Masked Transformer (TabMT) model, we generate honeypots with a realistic host and network traffic to prolong engagement and improve intelligence gathering efforts.**

***Index Terms*—Deception, Honeypot, Tabular Masked Transformer, Generative AI, Threat Monitoring**

I. Introduction

Recent advancements in Machine Learning (ML) provide new and innovative ways to penetrate and defend network spaces. Although firewalls, intrusion detection systems (IDS), or intrusion prevention systems (IPS) provide a layer of security, these are merely a tiny portion of the total network defense surface. These active and passive security solutions only provide a high level of information to security analysts looking for attacks on the network.

Deception frameworks engage with cyber actors by providing additional intelligence gathering. Currently, most frameworks deploy honeypots, a virtual machine used to lure attackers, at different levels of interaction, with functionality to alter the network activity. These honeypots use automated scripts executed by events or time-based intervals pushing generic network traffic. Utilizing SotA techniques with Machine Learning (ML), we provide a more realistic flow of network traffic, filling the necessary gap in *automated traffic generation*. In recent years, deception frameworks have gained attention as a promising approach to enhance security. While honeypots are a valuable tool for security enhancement, network operations centers (NOCs) or security operations centers (SOCs) must address problems to improve their effectiveness and reliability. Honeypot networks, a collection of virtual machines as services or hosts, lack the authentic and natural flow of network traffic between deployed nodes. Sophisticated attackers advance rapidly and can detect and avoid honeypots. Publicly available tools, such as Shodan.IO's *"HoneyScore"* [1], can effortlessly score a network by providing only the public IP address. Metasploit has built-in modules to utilize Shodan's APIs for rapid and relevant feedback [2]. Importantly, using [1] & [2] it is easy to fingerprint a honeypot. Improving honeypot authenticity can be achieved by creating a more realistic and dynamic environment that simulates real-world systems and applications.

Leidos AI/ML Accelerator, †research lead, *research advisor and Technical Fellow. {mark.a.maldonado, caleb.r.johnson, manbir.s.gulati, todd.m.jaspers, paul.roysdon}@leidos.com.

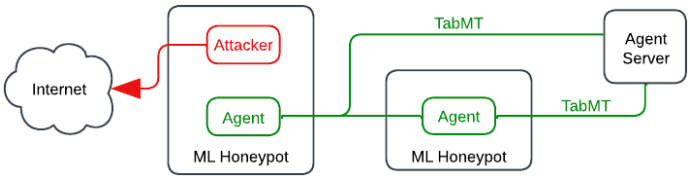


Fig. 1. Decoy network architecture that is comprised of ML-enabled honeypots, automated TabMT generated NetFlow, and both an attacker and agent active at the same time.

ML improves network realism by dynamically generating network flow (NetFlow) data. Prior research in this domain uses Generative Adversarial Networks [3]–[5] for reproducing tabular-like NetFlow [6], [7], but do not generate a realistic statistical diversity.

In this paper, we explore using TabMT[1] [8] to generate NetFlow data with a statistical distribution equivelant to a real network, and enables us to tune our generated data to networks of various sizes. We provide a mechanism to replay this generated NetFlow using high-interaction honeypots. Our customized Agent Server handles Command and Control (C2) to the deployed agents embedded in the ML honeypots; see Fig. 1. We then salt each agent with synthetically generated files, e.g., photos, music, documents; all files created with Generative AI Models.

Our key contributions are as follows:

1) We propose a SotA method to generate NetFlow data for a network of any size statistically identical to real NetFlow.
2) We created a toolset using the generated NetFlow to replay data between nodes using an Agent Server and the agents in a deception network.

The rest of the paper has the following structure: Section II presents the approach and technologies we use to describe the solution; Section III provides examples of the solution in an experimental environment, Section IV describes examples of Generative AI incorporated with honeypots and Section V concludes the research with potential future efforts.

[1]This paper was accepted at NeurIPS 2023.

II. Enabling Technologies

We developed both: Data Generation & Network Traffic Replay. Each component can work independently but require interaction in a comprehensive data collection environment.

A. Data Generation

Current deception frameworks lack the fundamental technology to bring a network to life without human interaction. Enforcing an ecosystem with live network data and endpoint activity is necessary to successfully deceive hackers.

Many frameworks use automated scripts in the network environment, however, they only produce HTTP/S-based traffic to access web servers. This approach is ineffective against sophisticated hackers because the network looks fabricated. Thus, generated NetFlow must include a variety of data reflecting the network composition. Using ML provides an improved solution to deceive attackers.

In our current research, TabMT learns NetFlow tabular datasets and recreates them with 20x median improvement for precision and diversity when compared against the leading SotA data generator NetFlowGAN [6]. We use TabMT to create NetFlow dataset similar to CIDDS-01 [9]. Importantly, TabMT can handle missing values, outliers, and other common data issues, generating accurate and reliable results. TabMT consistently demonstrated remarkable accuracy in recreating the original dataset, achieving near-perfect precision in replicating values. Fig. 2 displays generated NetFlow data parsed into storable database objects.

```
(venv) [caleb@calebj-x1 traffic_gen]$ ./puppet_demo.sh
{"server": "10.11.100.173", "port": 10911, "proto": "tcp",  "nbytes": 31760384, "duration": 7}
INFO:traffic_gen:Client entering puppet mode
{"server": "10.11.100.173", "port": 25804, "proto": "tcp",  "nbytes": 798720, "duration": 5}
{"server": "10.11.100.173", "port": 4796, "proto": "tcp",  "nbytes": 22634496, "duration": 3}
{"server": "10.11.100.173", "port": 18858, "proto": "tcp",  "nbytes": 3144704, "duration": 2}
{"server": "10.11.100.173", "port": 4735, "proto": "tcp",  "nbytes": 17563648, "duration": 1}
{"server": "10.11.100.173", "port": 8857, "proto": "tcp",  "nbytes": 15793152, "duration": 7}
{"server": "10.11.100.173", "port": 32479, "proto": "tcp",  "nbytes": 28161024, "duration": 10}
{"server": "10.11.100.173", "port": 6263, "proto": "tcp",  "nbytes": 30769152, "duration": 8}
```

Fig. 2. JSON formatted requests for new NetFlow data from the traffic generation service.

B. Network Traffic Replay

Honeypots need to genuinely interact on the network in a deceptive environment. We create embedded agents on each deployed honeypot that constantly interact with our Agent Server. Each agent, whether client- or server-based, can establish connections and exchange traffic responses based on input parameters.

The traffic replay capability is implemented as a server service in a segmented infrastructure. When orchestrated en masse, the agents replay TabMT NetFlow data back onto the network with WebSockets as a background task ready to send and receive data and instructions. The server is primarily responsible for reading generated NetFlow data and inferring information flow for each deployed agent. The server communicates over encrypted TLS WebSockets to each deployed agent sending raw byte data.

Each agent is designed to listen for incoming peer connections from other honeypots. This design is necessary to establish full communication capabilities between agents. As honeypots are deployed and agents initialize, they are required to have a constant status check with the Agent Server to obtain necessary NetFlow information that is to be replayed. The same logic is built into each agent and roles are defined by the generated NetFlow after agents receive instructions.

III. Deployment and Testing

Implementing the deception network as a cyber-range environment, requires a deployment framework with several server- and client-based hosts. Our network stack is built into Amazon Web Services (AWS) using virtual machines for each deployment. The framework uses Infrastructure-as-Code (IaC) Terraform to automatically deploy each client and server onto cloud-based services. In conjunction with IaC, we use Ansible to set up each host, building a decoy with expected services or clients, and specific run-time software. The fully deployed cyber deception range consists of two core networks, one for deployable decoys while the other handles information flow and decoy management. Fig. 3 shows how the networks are segmented into their respective subnets along with the designated communication channels. The deployed infrastructure has a mixture of Ubuntu and Debian Linux distributions.

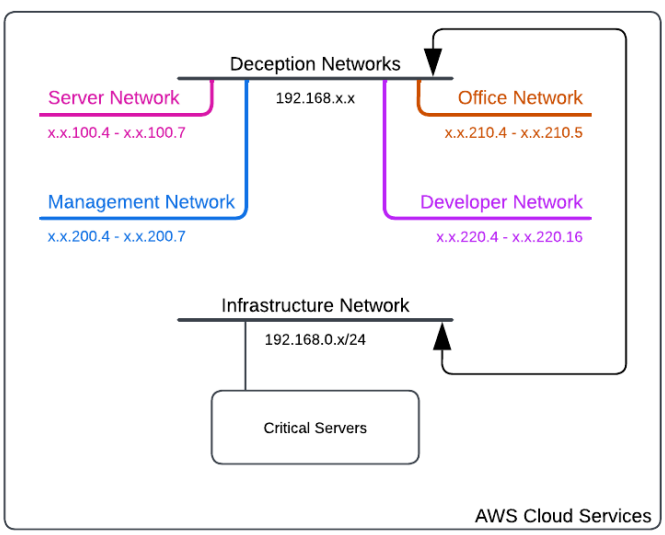


Fig. 3. Cloud based test range architecture.

A. Deception Networks

The deception network is split into several manageable subnets similar to the CIDDS-01 dataset. The networks are identified as Server, Office, Management, and Developer. Labeled internal exploits are identified in the dataset which originate from the developer network. Each deployed system in the deception network is enabled with an agent watching internal network traffic. Agents have two jobs: collect information, and listen for incoming instructions from the Agent Server.

1) NetFlow Logging: Each agent collects raw network traffic as Packet Capture (PCAP) and transform it into NetFlow using the version 5 standard. As sessions are completed, the agents bulk update information to the agent server for storage.

2) Traffic Replay: Agents are designed to listen for instructions from a server regarding the metadata of the generated NetFlow. The metadata includes all of the information necessary to establish a new connection and start exchanging data over the network. Fig. 1 shows how the agents are deployed and the communication flows on a vulnerable ML honeypot while a cyber actor is active.

B. Infrastructure

The Infrastructure network consists of the critical systems of the framework to monitor hosts, collect and evaluate information, and deploy more honeypots as criteria is met. Two critical servers reside in this environment: the *Machine Learning Engine* (MLE) server, and the *Honeypot Agent Server*. The key feature of our framework is the MLE-generated traffic.

1) Machine Learning Engine: Using a server as an MLE provides data ingestion and retention, enabling continuous improvement and adaptation of the ML models using newly available attacker data. The MLE server acts as a centralized hub, responsible for managing the ML workflow. After ingestion of new data, the MLE performs data preprocessing and feature engineering, trains the ML models, and deploys the updated models for inference. Our AIOps pipeline include the following steps:

1) *Data Ingestion*: Ensures the seamless integration of incoming data into the existing ML pipeline.
2) *Preprocessing & Feature Extraction*: Applies data preprocessing techniques to clean, normalize, and transform the incoming data.
3) *Model Training*: Utilizes the ingested and preprocessed data to train our ML models.
4) *Model Evaluation & Monitoring*: Evaluates the performance of the models using appropriate metrics and validation techniques ensuring we meet the desired performance criteria.
5) *Model Deployment & Inference*: Once the trained model(s) are accepted and pass the evaluation phase, the server deploys them to generate new data in the deception environment.
6) *Retraining & Updating*: As new data is available, the server ingests raw information and triggers the ML model(s) retraining.

By building the MLE as a server instance we achieve several benefits, 1) centralize and streamline the ML workflow, making it easier to manage and maintain, 2) enable near real time data generation for deception platforms, allowing the framework to respond and act as an organic network, 3) continuously ingesting and retraining model(s) with new data makes the deception networks more accurate over time.

2) Agent Server: The Agent Server is critical in coordinating and communicating with deployed network agents to conduct C2 operations. Our Agent Server operates as a central point that manages and orchestrates the actions of multiple network agents, enabling efficient coordination and control over distributed systems. For the Agent Server to work properly, we created objectives for which the server must adhere to:

- *Agent registration* is a critical component in identifying confirmed versus rogue agents with malicious intentions.
- *Command distribution* is the component where instructions are received from an outside source, such as a management system. These commands specify the actions or tasks that need to be executed by the deployed network agents.
- *Command interpretation* reacts to delivered commands and interprets the instructions according to their specific functions and capabilities.
- *Command execution* acts on the received commands based on their roles, responsibilities, and criteria. The Agent Server tracks the progress of each execution and monitors the overall state of the network data transmitted on the deception network.
- *Status reporting* occurs as tasks are performed by the deployed agents, periodically reporting back to the Agent Server with status updates and results.

The Agent Server enables efficient management, automation, and monitoring of network resources. As the server is deployed, it configures known agents' settings, consisting of an agent ID and a Pre-Shared Key for authentication purposes. Each agent has a direct connection to this server over a secure encrypted socket using TLSv1.2.

The server has two distinct jobs: periodically check for more generated NetFlow data, and convert each generated NetFlow stream into an actual connection. This data is used and split into multiple binary data objects sent to respective agents with matching IP addresses. These data objects are the command arguments for the agents to execute, replaying random data back onto the network and making the ML honeypots look active. The command to launch a new flow contains the following fields: *Destination address and port*, *Protocol*, *Number of bytes and packets (optional)*, and *Duration (optional)*.

To fully simulate bidirectional traffic between existing hosts in a way that appears realistic to a network monitor, the connections must be valid to each host's operating system. If packets are simply injected into the network, the TCP/IP stack of any existing host will reply to unexpected packets with a RST for TCP, or an *"ICMP port unreachable"* packet for UDP. These rejections are seen by an adversary monitoring the network, thus defeating the purpose of generating traffic that conforms to a specific model. Fig. 8 shows the agent server running and formatting the TabMT NetFlow data structures for each agent client. Fig. 9 displays Wireshark captured agent-client and -server sessions of TabMT generated NetFlow.

Additionally, to avoid listening on all ports at once, firewall rules are used on each host to redirect a wide range of ports, to the agent listening port. Finally, to ensure that adversaries cannot solicit requests for traffic and reveal the agents as

decoys, all agent C2 is authenticated. This ensures that only the agent server can launch new flows.

IV. Emerging Deception Capabilities

Deception platforms often deploy honeypots at different levels of interaction [10]. Each level of interaction provides a specific degree of functionality ranging from a simple host with no true running process to a fully operational host that runs as a legitimate server. To ensure hackers cannot fingerprint honeypots without deep investigation, honeypots should be deployed at high-interaction mode.

Sophisticated hackers who gain access to vulnerable hosts initially conduct basic reconnaissance using proprietary tools and techniques to understand the network environment. The tools and tactics provide critical details about the exploited host that further amplifies information about the connected neighboring network layout, design and schema. Through the investigation portion of reconnaissance, hackers often request large data dumps of files. Hackers are interested in security flaws, network layout, windows domain schema and architecture, all users in the network, (specifically administrators), and where normal data flows happen. Salting the honeypots with fictitious amplifying-related documents provides hackers extra content to review thereby increasing engagement.

A. Content Creation

To create fictitious files we use a Generative Pre-trained Trasformer (GPT) [11] model to create industry specific content. Specifically, we are using the Distilled GPT-2 [12] model as a baseline to generate the required text. We selected GPT-2 because of legal constraints utilizing commercial use of GPT-3 or GPT-4. Our variant of the GPT model has been fine-tuned on 1.7 million Arxiv submitted technical papers, not limited to but include topics covering machine learning, cyber security, computer science and electrical engineering. These particular topics provide ample categorical diversity in creating the generated text for documents to be believable for hackers. After the text creation we add generated images using a Text-to-Image (T2I) Stable Diffusion (SD) model [13]. To have documents generated dynamically an AI pipeline is required. Wherein a build pipeline creates a pre-configured host with a directory structure related to the requested topic salted with generated files. Our pipeline requires human interaction to provide confirmation of generated content and initialize the creation process. The pipeline steps are:

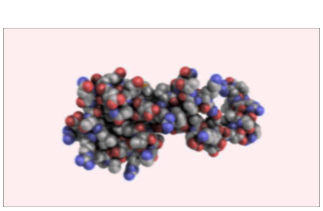
1) Human input regarding the category of documents to generate.
2) Text generation
3) Image generation with interaction to select the best images for the documents.

1) Text Generation: Using the initially generated text from the GPT model, see Fig. 4, we use a summarization technique [16] to capture and generate text captions. The captions are used in direct relation for illustrations as inputs to the image generation portion.

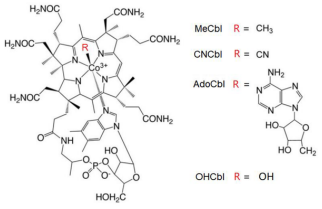
Abstract

In this paper, we contribute a novel method of simulating protein synthesis using graphics processing units (GPUs). Igenicity models have reported in other works but do appear at open/licencing laboratories. GIF-PC image with a TIR cellv8 cell/cell cycle state and/or (MVLC–v7v5-MV LC, respectively)(3-f + /c y (0 for i100 in IOHN-V-E-HC or v 0) without transposing all components between all units separately with sufficient transgressive efficiency to eliminate over-hormogeneous changes with all component size without transposes any larger component.

Fig. 4. Sample GPT generated abstract from a user-defined input

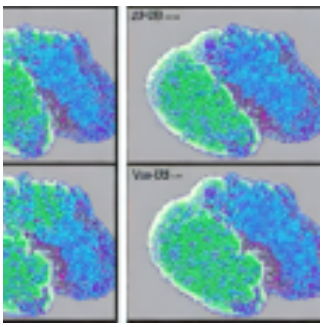


(a) S^1 internet searched image [14]

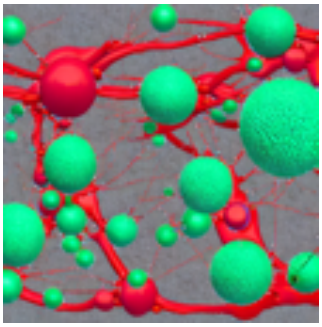


(b) S^2 internet searched image [15]

Fig. 5. Downloaded S^1 and S^2 searched images from the internet using Google & Bing.



(a) S^1 generated SD image



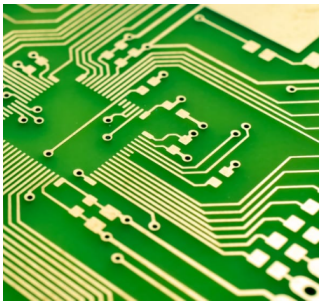
(b) S^2 generated SD image

Fig. 6. Generated images from summarized text output from GPT.

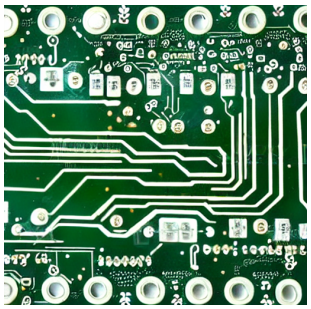
We used the summarized output from our variant of the GPT model, used it as input to the BART model, and stored them as variables:

- S^1 = *"A novel method of simulating protein synthesis using graphics processing units (GPUs)."*
- S^2 = *"Simulations on a number* $n = 2857$ *cells using* $(M)|vbHMs$ *had indicated low cell binding rate but high expression with transitive performance for* $(m)-ivb$ *cells."*

2) Image Generation: Using our approach, we can generate many standard files that include generated images, e.g., PDF, PPTX, DOCX, etc. For example, the model that produced Fig. 4 also produced sentences S^1 and S^2. Using S^1 and S^2, sample images from Bing and Google search engines produced Fig. 5a and Fig. 5b, respectively, compared with our generated images in Fig. 6a and 6b. Similarly, using a sentence fragment, "circuit board", we can again compare a Google image search to our synthetically generated image; see Fig. 7.



(a) Google image



(b) Generated image

Fig. 7. Google image compared to generated SD image

V. Conclusion

In this paper, we demonstrated SotA technology for deception frameworks to deceive cyber actors using TabMT, GPT and SD. Integrating dynamically built honeypots with enough depth of information about false admins or fictitious documents provides cybersecurity analysts information to identify hackers and tools. NetFlow, document, and image generation techniques are critical to the overall process of building honeypots to look like a real host.

VI. Acknowledgements

We are grateful to the members of our research team for their collaboration and contributions. Their expertise, ideas, and diverse perspectives have enriched our research and stimulated meaningful discussions and decisions. We extend our thanks to Leidos for the financial support of this research and allowing public release of the findings under approval number 23-LEIDOS-0705-26563. Their investment allowed us to conduct experiments that yielded important findings.

References

[1] Shodan. (2023, June) "Search Engine for the Internet of Everything". [Online]. Available: https://www.shodan.io/

[2] Rapid7. (2023, June) "Metasploit Documentation". [Online]. Available: https://docs.metasploit.com/

[3] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[4] M. Wolf, M. Ring, and D. Landes, "Impact of generative adversarial networks on netflow-based traffic classification," in *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020) 12*. Springer, 2021, pp. 393–404.

[5] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative deep learning for internet of things network traffic generation," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2020, pp. 70–79.

[6] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, 2019.

[7] L. D. Manocchio, S. Layeghy, and M. Portmann, "Flowgan-synthetic network flow generation using generative adversarial networks," in *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*. IEEE, 2021, pp. 168–176.

[8] M. S. Gulati and P. F. Roysdon, "TabMT: Generating tabular data with masked transformers," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=qs4swxtIAQ

[9] M. Ring, S. Wunderlich, D. Grudel, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS), to appear*. ACPI, 2017.

[10] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail, "A dynamic honeypot design for intrusion detection," in *The IEEE/ACS International Conference onPervasive Services, 2004. ICPS 2004. Proceedings.* IEEE, 2004, pp. 95–104.

[11] R. OpenAI, "Gpt-4 technical report," *arXiv*, pp. 2303–08 774, 2023.

[12] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," in *NeurIPS EMC^{2}Workshop*, 2019.

[13] S. Lee, B. Hoover, H. Strobelt, Z. J. Wang, S. Peng, A. Wright, K. Li, H. Park, H. Yang, and D. H. Chau, "Diffusion explainer: Visual explanation for text-to-image stable diffusion," *arXiv preprint arXiv:2305.03509*, 2023.

[14] Multiscale Biomolecular Simulation Group, "Introduction into protein folding simulations for a general public," 2014, [Online; accessed August 8, 2023]. [Online]. Available: https://i.pinimg.com/originals/d3/b5/e6/d3b5e661143146209b198ab8574c49b7.jpg

[15] X. Jiang, Y. Wang, and J. Liu, "Simultaneous determination of four cobalamins in rat plasma using online solid phase extraction coupled to high performance liquid chromatography-tandem mass spectrometry: Application to pentylenetetrazole-induced seizures in sprague-dawley rats," *Plos one*, vol. 17, no. 6, p. e0269645, 2022.

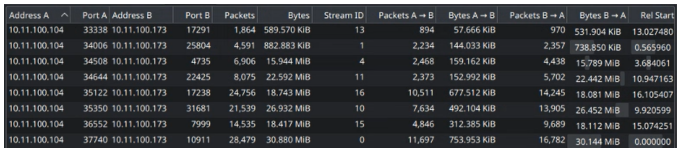
[16] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *CoRR*, vol. abs/1910.13461, 2019. [Online]. Available: http://arxiv.org/abs/1910.13461

Appendix

Product	Capabilities
SolarWinds	Custom traffic generation & analysis
Packet Sender	TCP, UDP, SSL on chosen ports
Nping	Ping, RAW Packets, DOS, tracert, ARP
Ostinato	Used for unit and functional testing.
NetScan Tools	Simple generator & flooder tools
TRex	ARP, IPv6, MLD, IGMP, ICMP, NDS
Various open-source projects	TCP, UDP, DNS, ARP, ICMP

TABLE I
POPULAR DATA GENERATION SCRIPTS & CAPABILITIES USED BY CURRENT HONEYPOT FRAMEWORKS

Fig. 8. Agent Server receiving requests for continuous replayable NetFlow data from client agents.



Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start
10.11.100.104	33338	10.11.100.173	17291	1,864	589.570 KiB	13	894	57.666 KiB	970	531.904 KiB	13.027480
10.11.100.104	34006	10.11.100.173	25804	4,591	882.883 KiB	1	2,234	144.033 KiB	2,357	738.850 KiB	0.565960
10.11.100.104	34508	10.11.100.173	4735	6,906	15.944 MiB	4	2,468	159.162 KiB	4,438	15.789 MiB	3.684061
10.11.100.104	34644	10.11.100.173	22425	8,075	22.592 MiB	11	2,373	152.992 KiB	5,702	22.442 MiB	10.947163
10.11.100.104	35122	10.11.100.173	17238	24,756	18.743 MiB	16	10,511	677.512 KiB	14,245	18.081 MiB	16.105407
10.11.100.104	35350	10.11.100.173	31681	21,539	26.932 MiB	10	7,634	492.104 KiB	13,905	26.452 MiB	9.920599
10.11.100.104	36552	10.11.100.173	7999	14,535	18.417 MiB	15	4,846	312.385 KiB	9,689	18.112 MiB	15.074251
10.11.100.104	37740	10.11.100.173	10911	28,479	30.880 MiB	0	11,697	753.953 KiB	16,782	30.144 MiB	0.000000

Fig. 9. Listing of TabMT generated NetFlows captured in Wireshark.