D2U: Data Driven User Emulation for the Enhancement of Cyber Testing, Training, and Data Set Generation

Anonymous Author(s)

ABSTRACT

Whether testing intrusion detection systems, conducting training exercises, or creating data sets to be used by the broader cybersecurity community, realistic user behavior is a critical component of a cyber range. Existing methods either rely on network level data or replay recorded user actions to approximate real users in a network. Our work is the first to produce generative models trained on actual user data (sequences of application usage) collected on endpoints. Once trained to the user's behavioral data, these models can generate novel sequences of actions from the same distribution as the training data. These sequences of actions are then fed to our custom software via configuration files, which replicate those behaviors on end devices. Notably, our models are platform agnostic and could generate behavior data for any emulation software package. In this paper we present our model generation process, software architecture, and an initial evaluation of the fidelity of our models. Our software is currently deployed in a cyber range to help evaluate the efficacy of defensive cyber technologies. We suggest additional ways that the cyber community as a whole can benefit from more realistic user behavior emulation. The data used to train our model, as well as sample configuration files produced by the model, are available at [redacted].

KEYWORDS

data sets, experimental infrastructure, user emulation, data driven

1 INTRODUCTION

It is difficult to generate benign data sets for training cyber tools and realistic background traffic to use in cyber exercises or tool testing [1, 17], yet both are critical to the cybersecurity community. In this paper we present D2U, a novel user behavior emulation technology. Unlike current user emulators [5, 16, 20], D2U does more than replay previously-observed behavior, or call heuristics for stringing together patterns of actions. Instead, D2U provides the ability to generate unlimited, novel sequences of realistic behavior and orchestrate a plethora of virtual machines to enact these behaviors, thereby emulating a network of users. This ability to generate realistic, new behaviors sets D2U apart from existing emulation technologies.

In a one-time training phase D2U collects fine-grained data on real computer users and creates probabilistic generative models of each users' unique patterns of behaviors. These models enable the production of new sequences of behaviors that appear like real user data. D2U includes a suite of metrics allowing visual quality analysis to measure the realism of the generated behaviors.

Next, user behavior, as generated by our models, is fed into custom software that is able to actuate those behaviors on real or virtual devices—a "digital twin" of the user on that device. By placing these twins on many devices on the network, D2U provides solutions to a variety of problems, including: (1) generating realistic host and network data for testing cyber tools (e.g., needed to evaluate emerging cybersecurity technologies, such as User Behavior Analytics (UEBA), Anomaly Detection (AD), and Intrusion Detection System (IDS) technologies), (2) creating cyber deception technologies (e.g., camouflaging real users/traffic with realistic emulated versions), (3) creating training data for machine-learning-based IT and cyber tools, (4) enhancing the realism of cyber exercises (e.g., red-team events for testing and practicing network defense), and (5) generating novel datasets to support both research and industry. D2U is currently running in a cyber testbed with over 300 active emulated users at [redacted] to support large-scale experiments of defensive cyber technologies.

2 MODELING APP USAGE SEQUENCES

To create a data-driven model of a user's behavior, data recording the user's activity is collected from the user's host for an ample period of time (many days to weeks). The data is used to train a generative model, that is, an algorithm that, once trained, can be used to produce novel sequences of activities that mimic the input data's qualities. This section details the data collection and preprocessing, then introduces the generative models tested alongside sequential qualities of the data from previous research that drive evaluation and model selection.

2.1 Data Collection & Preprocessing

For D2U a collection script was deployed to a user's host, which records a timestamped observation each 0.5s of the user's "active application", i.e., the frontmost application in the operating system's (OS's) user interface (UI). Our collection script, written in Python, leverages the AppKit[1] package for Mac OS data collection, which is used in this paper, although we created analogous collection scripts for Linux and Windows OSes from a variety of Python modules. The program was designed to collect data continuously for eight active hours (not including sleep periods where logging was automatically paused). Notably, the program recorded `loginwindow` as the app name before the computer would sleep/pause logging, which allows our model to generate realistic pauses in the user's day. For data collection, users were instructed to use their computers as normal, but to not turn off their machines at the end of a work day. When a

CSET'21, August 09, 2021, Virtual
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]https://developer.apple.com/documentation/appkit

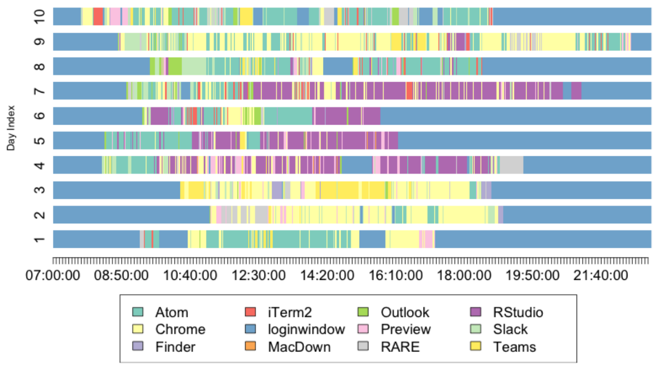


Figure 1: Ground Truth Data Example: Ten days (2 work weeks) of data (after preprocessing) for a user.

collection period was complete, they were instructed to restart the data collection program.

Each run of the collection program provides a time series of active app observations: $(s'(t_0), \ldots, s'(t_L))$, where $t_{k+1} = t_k + 0.5$s for most k, and L = 8hrs $\times 60^2$s/hr $\times 2$ = 57.6K observations. Note, we use s' to denote the raw sequence reserving the simple notation s for it's analogous sequence of apps after preprocessing. Once we have obtained a sufficient number of observation sequences from a user, we normalize our data to have the following properties:

- A single sequence per day, filtering out days in which there was insufficient active data (e.g. user was logged out). When denoting the sequence for a particular day is needed, we let s^i denote the sequence for the i-th day.
- A uniform start and end time (this depends on the user), which we define as simply the minimum start time ($\min_{s'} t_0$) and maximum end time ($\max_{s'} t_L$) observed.
- Uniform and uniformly spaced timestamps in each day's sequence (i.e., all s are sequences with the same time stamps), coarsened to five second intervals. To do this we simply set $s(t)$ to be the closest previous observation—set $s(t) := s'(t_k)$ where $t \in (t_k, t_{k+1})$—provided we have observations s' near time t. In the alternative case, where there is a large gap in a day's data collection (e.g., from the collection script ending but not being restarted quickly) we copy subsequences of s' occurring before and after the gap to fill in the unknown portion, then define $s(t)$ as just previously mentioned.
- We replace seldomly used apps—defined as apps that occur in only a single collection period or that make up less than 1% of the data—with a distinguished symbol, RARE, effectively binning all infrequent observations.

Now armed with uniform sequences of user behavior for many days, we consider the sequential properties and models that may faithfully reproduce those properties.

Before proceeding we note that Figures 1, 2, 3, and in Table 3 leverage R package of Gabadinho et al. [6].

2.2 Sequential Concepts and Modeling

We seek models that are "high-fidelity", where we define the fidelity of a user model in terms of the similarity of the activity sequences the model generates to that of the true user sequences; hence, gauging how "good" a model is depends solely on defining similarity of two sequences. Sequential data appears in a wide variety of domains, hence, a wide variety of similarity measures for sequences exist; e.g., string metrics such as Hamming for binary data [10], edit distances (e.g., Levenshtein) often for text applications [9], or Kendall-Tao distance often for ranking comparisons [11], to name a few. Studer & Ritschard [19] provide a thorough survey.

Considering Figure 1, which visualizes ten days (sequences) of a user's application usage, we observe wide variance in behaviors that characterizes real user data: different app distributions appear at different times, long spells of usage of a particular app are common, but frequently with many short interruptions into multiple other apps; and there is a general time window of work hours in the day. Our goal is to create/discover a model that, when trained on data as in Figure 1, will produce data that has similar qualities. While the general workflow for creating such a model is to identify metrics that measure the desired qualities, and then leverage these metrics to design and evaluate models, it is unknown what combination of sequential measures capture the qualities of real user's behavior—and this is a highly non-trivial problem.

Although working in social sciences, Studer & Ritschard [19] encountered this very problem, and have laid a foundation of sequential concepts that frames our approach. We consider similarity of sequential activity data with regard to four dimensions or sequence characteristics as itemized by Studer & Ritschard:

- *Sequencing* - the ordering of distinct applications (i.e., order in which a user jumps from one application to another);
- *Duration* - the lengths of consecutive observations of the same application (i.e., uninterrupted time spent in each application);
- *Timing* - the time(s) of day at which each application is used;
- *Distribution* - the total time spent in each distinct application.

Continuing, we introduce necessary terminology and notation, following previous works ([6, 19]) but adapted for our needs.

- The terms *state* and *app* and *symbol* are used interchangeably to refer to the user's active application.
- A *Symbol Time Sequence (STS)* is a sequence $s(t)$ where each element of the sequence $s(t_i)$ denotes the symbol at time t_i.
- A *spell* is a consecutive subsequence of the same symbol, or equivalently, an uninterrupted period spent in the same application; it follows that spell length, the number of consecutive same symbols, is *duration* as defined above.
- The *Distinct Successive States (DSS)* is the sequence of distinct symbols observed, where all spells are treated as length 1. We denote a DSS using $u = (u(1), \ldots, u(m))$.
- A *Distinct Successive State Duration Sequence (DSSDS)* is a sequence of (symbol, spell duration) tuples for each spell in the DSS.

Example: For the toy STS, $s = (a, b, b, b, a, c, c)$ there are three states or symbols, namely, a, b and c; symbol a has two spells of length 1, b has a spell of length 3, and c has a spell of length 2; the DSS is $u = (a, b, a, c)$; the corresponding DSSDS is $((a, 1), (b, 3), (a, 1), (c, 2))$.

Knowing the DSSDS is a redundant but alternative specification of the original STS. As we shall see, choosing whether to consider a user activity sequences as an STS or a DSSDS has impact on model choice and accuracy that we explore via initial results.

2.3 Modeling

We develop a flexible framework for modeling user activity sequences where a model can be specified by three modeling decisions:

(1) *Sequence Representation*: STS or DSSDS (Sec. 2.3.1);
(2) *Temporal Structure*: Flat or Hierarchical (Sec. 2.3.2).
(3) *Model Type & Hyperparameters*: Markov Chain (MC), Hidden Markov Model (HMM), or Random Surfer (RS); hyperparameters vary per model type (Sec. 2.3.3);

The four possible graphical model frameworks are depicted in Table 1 and correspond to each of the two choices for the two structural modeling decisions ((1) and (2)). Model types (3) are shown in Table 2 with the model-specific hyperparameters tested. Each of the four structures (choices for (1) and (2)) are used with one of the three model types, yielding a total of $4 \times 3 = 12$ model combinations ($4 \times 6 = 24$ when considering hyperparameters we tested).

2.3.1 Sequence Representation: STS vs. DSSDS. Refer to Section 2.2 for definitions and examples of STS, DSSDS, and related concepts/terminology. STS models regard and generate the original, full, STS sequence, which includes constant subsequences if an app is used for consecutive time intervals. For models with Markov assumptions, spell lengths that deviate from the mean can be problematic. On the contrary, the DSSDS modeling choice indicates that the model will be trained on and generate a DSS (Distinct State Sequence, containing no repeated symbols) combined with the number of intervals to remain in each symbol. More specifically, a symbol is drawn from the symbol model; then, the duration for the chosen symbol is sampled from the given symbols' duration probability distribution. The spell duration distributions are learned for each symbol from all training sequences.

2.3.2 Temporal Structure: Flat vs. Hierarchical. The Flat structure simply regards data throughout the day identically and builds a user model (based on other structural choices (1), and modeling choices (3)), while the Hierarchical structure incorporates a latent variable c that regards the time of day. For the Hierarchical structure, the input data sequences are split into time window subsequences (1 hour in our case), vectorized, and clustered via $K-$means clustering. We learn K from each user via the elbow method. See Figure 2.

Let $v_1, \ldots, v_K$ denote the cluster centers, which are simply the expected percent each app appeared in that cluster's constituent data. Since each time window, w, in the training data now dons a cluster assignment, $c_w \in \{1, \ldots, K\}$, a Markov distribution of order 2 is learned on the sequence of clusters observed. (This is used to generate a cluster for each hour of the day). Finally, a user model is trained from the data in each cluster, resulting in models $M_1, \ldots, M_K$ (based on the other structural and modeling choices (1),

Table 1: Graphical Models: Four general model frameworks where the Hierarchical models contain Flat models as submodels (shown in rectangles). s_t denotes th app at timepoint t; u_i denotes the $i-$th successive app in the DSS; d_i denotes the duration of app u_i; c_w is the temporal cluster for window w. Dotted lines dependence, as dictated by the specific model types (see Table 2); whereas solid lines indicate dependence that is computed simply as $P(X_i|Pa_{X_i}) = \frac{\#(X_i)}{\#(Pa_{X_i})}$ where Pa_{X_i} denotes the parents of node X_i, and $\#(\cdot)$ denotes the empirical frequency of the node(s)' value(s). Note that dependencies shown may not be complete/accurate for all the model types; e.g., for a Markov chain of order 2 (a Flat STS model), s_t is dependent on s_{t-2} as well as s_{t-1}, and for a Hidden Markov Model (a Flat STS model), the hidden states, H_{t-1}, H_t are not depicted.

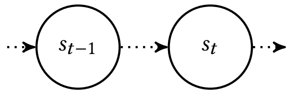
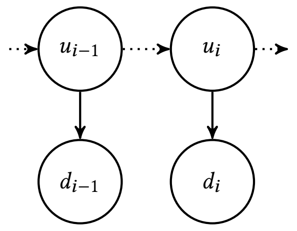
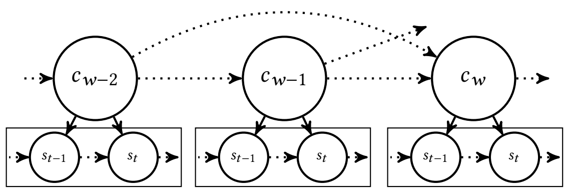
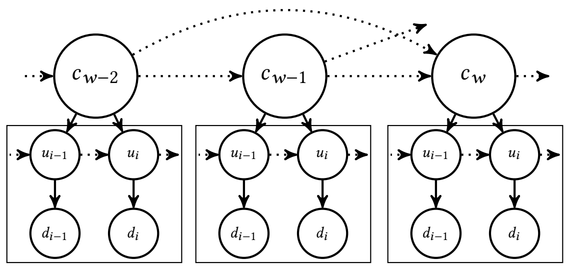
	STS	**DSSDS**
Flat		
Hierarchical		

Table 2: Specific Model Types with Tested Hyperparameters

Type	Hyperparameters
Markov Chain (MC)	$m = 1, 2$
Hidden Markov Model (HMM)	$n_h = 3, 5, 7$
Random Surfer (RS)	$\alpha = [20, 20], \beta_{i,j} = \delta_{i,j} = 1.1$

(3), (4)). These K user models attempt to capture that users' behavior as observed in the data that forms that particular cluster. The generative model process is then as follows: at each time window (w) a cluster (c_w) is chosen based on the current time window, and the previous two chosen time window clusters; a subsequence for that time window (that hour of the day) is generated from the corresponding cluster's user model (M_{c_w}). A priori, the benefit of this approach is that natural differences in behavior throughout the day are taken into account. As our results show, this hierarchical model also generates higher variance sets of sequences, which tends to produce more realistic sequence sets.

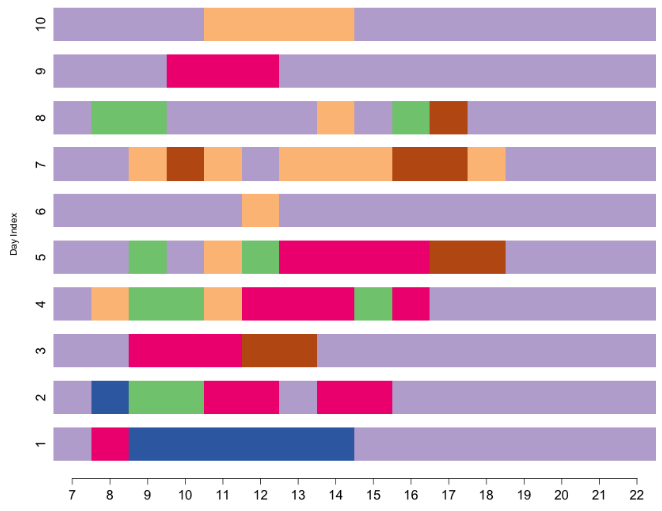


Figure 2: Clusters for each 1-hour time window (w) found for user data from Figure 1) with $K = 7$ (for Hierarchical temporal structure).

2.3.3 Specific Model Types. Markov Chain (MC): This model uses an MC model of order m, the lone hyperparameter; i.e., the probability of a symbol depends only on the previous m symbols. This model is learned from the input data sequences and is implemented using Pomegranate Python package.[2] For details on theory see, e.g., [7]. As a MC of order $m = 1$ is perhaps the simplest model that respects sequential data, we consider it the benchmark.

Hidden Markov Model (HMM): HMM models are comprised of n_h hidden or latent states (n_h is a hyperparameter), an MC model for transitioning among the hidden states (of order 1 in our case, so a transition matrix), and each hidden state is furnished with an emission probability, which is simply a distribution over the observed symbols; for details on HMM theory, see, e.g., [15]. The generative process uses the transition matrix to sample the latent state, then a symbol is sampled from emission probability for that state. This model is trained from the input data sequences using the Baum-Welch algorithm and implemented using Pomegranate Python package.[2]

Random Surfer: The Random Surfer model is the underlying model in the PageRank algorithm [3]. The "surfer" (user) moves between applications (symbols) as a mixture of a symbol transition matrix T (row-stochastic matrix giving an MC model of order 1) and a "teleportation" distribution p (multinomial distribution) on the n symbols. Specifically, at each step, the user will, with probability π, choose the next symbol based on T and the current symbol, or, with probability $1 - \pi$, sample the next symbol from the multinomial distribution of symbols, p, which is independent of the current symbol. This model was developed to mimic the behavior of a surfer browsing through web pages, choosing at each step to either follow a link on the current page, or jump to a entirely unrelated page. The model is parameterized by π, p, and T, which are learned from data by optimizing the posterior distribution (Maximum a Posteriori estimate), using a gradient ascent algorithm. See A for more details. The hyperparameters of our implementation define the prior distributions on each parameter: mixing parameter $\pi \sim \text{Beta}(\alpha)$, transition matrix rows (multinomials) $T(i, \cdot) \sim \text{Dirichlet}(\beta_i)$ and multinomial $p \sim \text{Dirichlet}(\delta)$. As shown in Table 2, we use 20 for both α components, strongly encouraging equal use of T and p, and use 1.1 for all Dirichlet values.

2.4 Qualitative Modeling Results

We present results for a particular user for which we have 42 days of sequential data after the preprocessing step, collected across ~ 2 months. For the hierarchical models, $K = 7$ clusters were found for this user by the elbow method. A sample of the clusters is depicted in Figure 2. We note that K varies per user in empirical experiments, so we suggest learning this parameter per user.

Regardless of the model used, our results found trends based on each of the structural decisions (choices (1) and (2)). Regard Figure 3, which shows representative samples from the benchmark model, (MC, $m = 1$) for each of the four structural decisions. In short, regardless of model choice, STS models have spell lengths that are all very short—i.e., apps are changed too often and too sporadically. To explain this, consider sampling the same apps for an extended period of time. Although longer spells are not infrequent, the probability of staying in the same app, say $P(a|a)$, is small since our data has so many app changes. Thus, the probability of a spell of length k is equal (for MC with $m = 1$ or otherwise close to) $P(a|a)^k$, which tends to 0 quickly as k grows.

Our results also confirm that Flat models provide unrealistic data for the time of day (unsurprisingly as they do not regard the time of day), while the Hierarchical models do capture time-of-day trends. Overall, regardless of the model choice, DSSDS Hierarchical models are best, as they have much more realistic spell lengths (app use duration) and regard the time of day.

[2]https://github.com/jmschrei/pomegranate

Table 3: Structural Choices Pros & Cons: Two days (7AM - midnight) of data sampled from the trained Markov chain of order 1 (benchmark model) for each structural choice. Compare with ground truth data and regard color key in Figure 1. For both Flat and Hierarchical structure, STS models have unrealistically short spell length; that is, they jump between apps too often. DSSDS models (which sample a necessarily different symbol and the duration to remain in that symbol) capture much more realistic spells. For both STS and DSSDS models, the Flat structure disregards time of day—notice early and late app usage along with long periods logged out during the middle of the day, neither of which occurred in ground truth data. As designed, the Hierarchical models capture time-of-day trends in the data. Finally, we note that while this data is only two days from the MC order 1 model, the advantages and pitfalls of these structural choices are representative for all models.

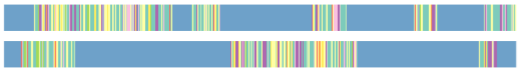
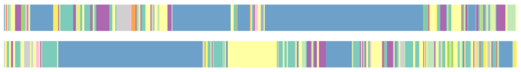
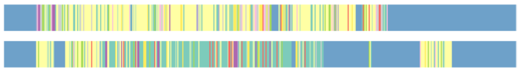
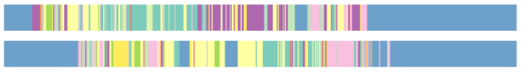
	STS	DSSDS
Flat		
Hierarchical		

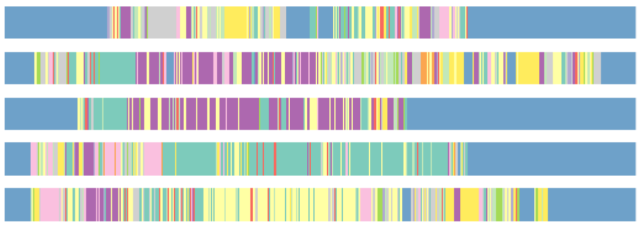
Figure 3 depicts the best performing models under the DSSDS Hierarchical structure. The MC $m = 2$ model and the HMM $n_h = 5, 7$ (7 not shown) provide very realistic generated app sequences. The Random Surfer model is a bit more inclined to change apps rapidly; in particular, the large number of logged-out periods midday seem unrealistic. These preliminary results conclude that the DSSDS Hierarchical MC and HMM models are the best choices.

We note that the Hierarchical modeling framework will allow different model types per cluster (e.g., M_1 is MC, while M_2 is Random Surfer), although we did not test this. In light of these results, for clusters with high entropy and overall short spells, we suspect the Random Surfer model to excel. Admittedly, further testing on much more data from many different users is needed to draw more convincing conclusions. As such a study will require a recruitment and data collection from many participants, it is out of scope for this preliminary work.

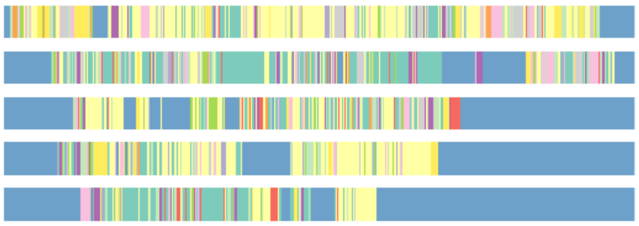
3 D2U DEPLOYMENT

To deploy D2U inside a cyber range, we developed a custom software architecture. This architecture, shown in Figure 4, is scalable, enabling hundreds or thousands of emulators to run simultaneously, and extensible, allowing additional user actions to be added and expanded with minimal effort using python. A front-end graphical user interface enables analysts to view the status of and send commands to all active user emulators.

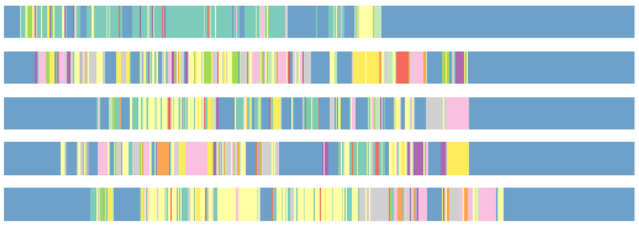
When using this architecture for data generation or testing, it is important to ensure that the emulated user and the management server are communicating out of band from the traffic generated by the emulated user node. Generated traffic should be from the emulated user's actions and not from contact with the management server. In cases where only a limited number of emulators are required or a more lightweight solution is desired, D2U can be run headless without connecting to either the frontend or the management server. When running headless, less verbose logging information is stored locally.



(a) Markov Chain Model ($m = 2$)



(b) Hidden Markov Model ($n_h = 5$)



(c) Random Surfer Model

Figure 3: Five days sampled from each of the best DSSDS Hierarchical models, (K = 7 clusters, 1 hour window length) trained on sequences from user in Figure 1.

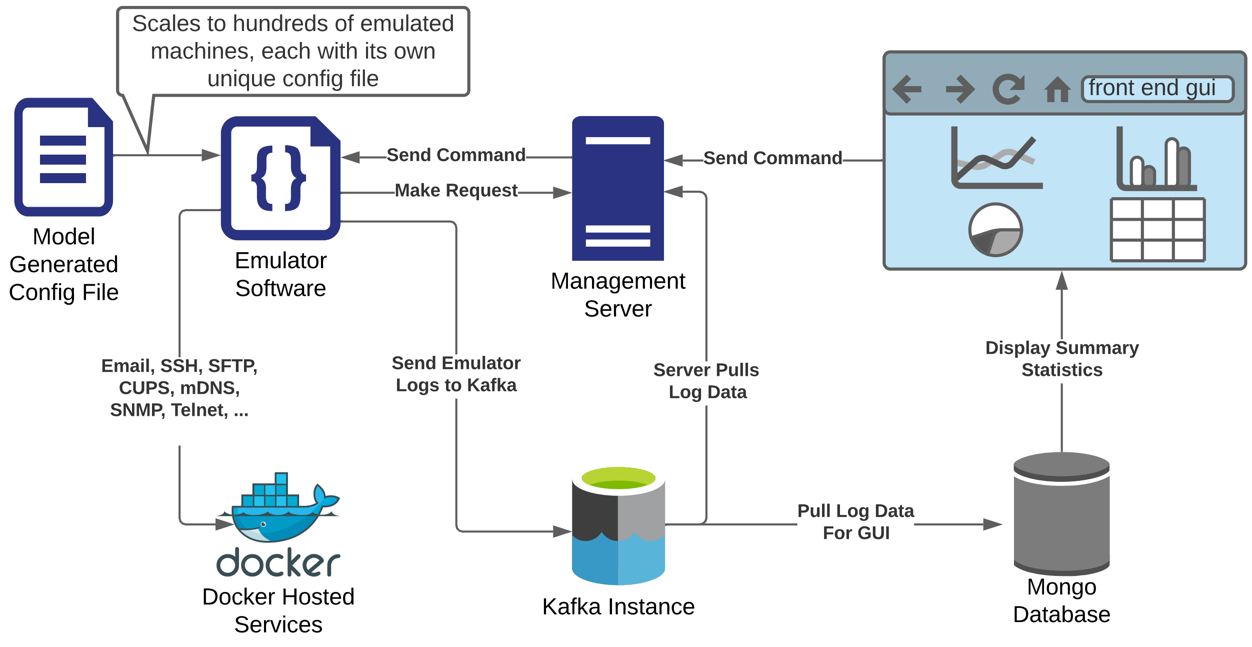

Scales to hundreds of emulated machines, each with its own unique config file
Model Generated Config File
Emulator Software
Send Command
Make Request
Management Server
Send Command
front end gui
Display Summary Statistics
Email, SSH, SFTP, CUPS, mDNS, SNMP, Telnet, ...
Send Emulator Logs to Kafka
Server Pulls Log Data
docker
Docker Hosted Services
Kafka Instance
Pull Log Data For GUI
Mongo Database

Figure 4: D2U Deployment Architecture

4 RELATED WORK

The problem of reproducing realistic data for use in cyber testing and training is well established in the literature [1, 17]. Many existing datasets cannot be properly validated and are outdated [1, 14, 21]. A critical step towards more realistic cyber data for training and testing is high fidelity user emulation [5]. This paper describes D2U, a generative model based solution to the problem of emulating realistic user behavior on end devices. In the remainder of this section, we highlight how D2U differs from existing user emulation technologies.

4.1 Existing User Emulation Technologies

Existing user emulation technologies take one of the following approaches: (1) modeling and generating network traffic directly [12, 13, 16], (2) replaying recorded user behaviors [5], and (3) using human generated configuration files (these may be approximated from real data) [20]. Solutions that fall into category (1) are fundamentally different from D2U because they replay or generate network traffic, whereas D2U emulates user behavior on end devices. In addition to the fact that it is more realistic to generate network traffic by emulating behavior on end devices, simply generating network traffic is also limited in its ability to test tools focused on anomalous user behavior.

Dutta et al. [5] developed user bots to address this shortcoming in traffic generators while testing insider threat detection systems, and was the major work we identified in category (2). These bots can be run in an enterprise system to test live deployments or in a cyber range. To enhance the realism of these bots, they replay user behavior data recorded during a study of West Point Cadets.

Finally, there are numerous technologies that fall into category (3), so for the sake of space we include notable representative samples. GHOSTS [20] was developed by Carnegie Mellon in order to build accurate, autonomous non-player characters (NPCs) for cyber warfare exercises. It is written in C# and supports web browsing, terminal commands, email, and editing office documents. It also allows the injection of commands during the exercise. Configuration of users is accomplished using JSON files that specify what actions a user will perform and provide sample text for emails and documents.

Some privately owned companies, such as Skaion [3] and SimSpace [4], also provide user behavior emulation capabilities. However, the code is proprietary and therefore cannot be analyzed or extended by the broader community. To our knowledge, the models used in these emulators are not based on measurements of individual user behaviors.

4.2 Relation to D2U

In contrast to the three approaches used by existing technologies, D2U provides the ability to generate unlimited, novel sequences of realistic behavior by building models from collected user data. Not only does D2U produce high fidelity user behaviors, it also provides an endless supply of such behaviors. Given the current lack of quality data for cyber training and testing, this capability represents an important step forward for the cyber community.

[3] http://www.skaion.com/
[4] https://simspace.com/

5 CONCLUSION

D2U provides unlimited, novel sequences of user behavior using generative models based on actual user data for use in testing and training. In this paper we described our processes for data collection and model generation, as well as qualitatively demonstrating the performance of our model and describing the software architecture used in deployment. Next steps include the following:

(1) Collect additional forms of user data to use as input to our models. In addition to application sequence data, file access logs, CPU usage data, browser logs, and screenshot image analysis would offer additional insight into user behaviors. It would also be beneficial to explore the pros and cons of different approaches to model generation with larger sets of user data.
(2) Pioneer the next generation emulated users: While our approach is an adequate solution for the problem we encountered (cyber range environment), a logical next step conceptually is adding a responsive and "smart" AI component to these emulated users; e.g., embedding NLP components to "read" and realistically respond to chat messages, emails, etc. In short, broadening the task from faithfully generating sequences of actions to building emulated users that seek to pass the Turing test. An example step in this direction would be to use an extra layer in the model where a user is "holding in mind" multiple applications per task and switching between them.

ARTIFACT AVAILABILITY

The application sequence data used to train our model, as well as sample configuration files produced by the model, are available at [redacted]. This technology is currently under provisional patent [redacted].

ACKNOWLEDGMENT

The research is based upon work supported by the Department of Defense (DOD), Naval Information Warfare Systems Command (NAVWAR), via the Department of Energy (DOE) under contract DE-AC05-00OR22725. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the DOD, NAVWAR, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

REFERENCES

[1] William H Allen. Mixing wheat with the chaff: Creating useful test data for ids evaluation. *IEEE Security & Privacy*, 5(4):65–67, 2007.
[2] Aragats Amirkhanyan, Andrey Sapegin, Marian Gawron, Feng Cheng, and Christoph Meinel. Simulation user behavior on a security testbed using user behavior states graph. In *Proceedings of the 8th International Conference on Security of Information and Networks*, pages 217–223, 2015.
[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.
[4] Ramkumar Chinchani, Aarthie Muthukrishnan, Madhusudhanan Chandrasekaran, and Shambhu Upadhyaya. Racoon: rapidly generating user command data for anomaly detection from customizable template. In *20th Annual Computer Security Applications Conference*, pages 189–202. IEEE, 2004.
[5] Preetam Dutta, Gabriel Ryan, Aleksander Zieba, and Salvatore Stolfo. Simulated user bots: Real time testing of insider threat detection systems. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 228–236. IEEE, 2018.
[6] Alexis Gabadinho, Gilbert Ritschard, Nicolas S. Müller, and Matthias Studer. Analyzing and visualizing state sequences in r with traminer. *Journal of Statistical Software*, 40(4), 2011.
[7] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation.* John Wiley & Sons, 2017.
[8] Ashish Garg, S Vidyaraman, S Upadhyaya, and K Kwiat. Usim: a user behavior simulation framework for training and testing idses in gui based systems. In *39th Annual Simulation Symposium (ANSS'06)*, pages 8–pp. IEEE, 2006.
[9] Wael H Gomaa, Aly A Fahmy, et al. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.
[10] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.
[11] William R Knight. A computer method for calculating kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.
[12] Samir Mammadov, Dhanish Mehta, Evan Stoner, and Marco M Carvalho. High fidelity adaptive cyber emulation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.
[13] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand, and Mathieu Couture. Automatic evaluation of intrusion detection systems. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 361–370. IEEE, 2006.
[14] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.
[15] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
[16] Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. Lariat: Lincoln adaptable real-time information assurance testbed. In *Proceedings, IEEE Aerospace Conference*, volume 6, pages 6–6. IEEE, 2002.
[17] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.
[18] Alex Shye, Benjamin Scholbrock, Gokhan Memik, and Peter A Dinda. Characterizing and Modeling User Activity on Smartphones. Technical report, 2010.
[19] Matthias Studer and Gilbert Ritschard. A comparative review of sequence dissimilarity measures. 2014.
[20] Dustin D Updyke, Geoffrey B Dobson, Thomas G Podnar, Luke J Osterritter, Benjamin L Earl, and Adam D Cerini. Ghosts in the machine: A framework for cyber-warfare exercise npc simulation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States, 2018.
[21] Stefano Zanero. Flaws and frauds in the evaluation of ids/ips technologies. In *Proc. of FIRST*. Citeseer, 2007.

A RANDOM SURFER (PAGERANK) MODEL IMPLEMENTATION

Let $s = [s_0, \ldots, s_m]$, with $s_t \in \{1, \ldots, n\}$, indicating the application (of n applications) observed at each time interval for $m + 1$ consecutive time intervals. From each users we will have ideally multiple (l) observation sequences, and use superscript k, s_t^k to denote the k^{th} sequence of observations from that user.

We follow the PageRank [3] diffusion process to model the user's trace of applications with a mixture model, although our end goal is different—we are not interested in using the stationary vector to rank nodes (nodes represent applications in our case), but instead leverage the mixture model between a markov transition matrix and non-markov "starting" distribution for modeling the application sequence per user. Notationally, let

- $\pi \in [0, 1]$ be a mixing parameter or dampening factor, simply a binomial probability for the mixture model (coin flip) between p and T;
- $p : \{1, \ldots, n\} \to [0, 1]^n$, be the starting distribution ($\sum p(j) = 1$), a multinomial over all n applications representing the likelihood of starting a new task with that application;

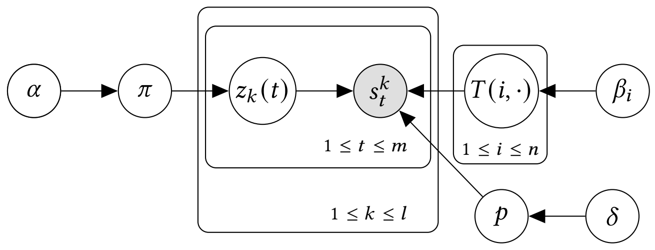


Figure 5: Plate diagram for mixture model. Each s^k is a sequence of observed state changes (applications used) and are assumed independent. We have priors as follows: mixing parameter $\pi \sim$ Beta(α), transition matrix rows (multinomials) $T(i,\cdot) \sim$ Dirichlet(β_i) and multinomial $p \sim$ Dirichlet(δ). While initial state $s_0^k \sim p$, subsequent states are sampled with a mixture from $T(s_{t-1}^k, \cdot)$ with probability π (case $z = 1$) or sampled from p with probability $1 - \pi$ (case $z = 0$).

- $T \in [0,1]^{n\times n}$ be a row-stochastic transition matrix, $\forall i \sum_j T(i,j) = 1$, so that $T(i,j) = P[s_t = j | s_{t-1} = i]$, i.e., the probability the user transitions from application i to j.

We interpret T as giving the likelihood of application transitions from the natural workflow, as opposed to starting a new task, as modeled by p.

As depicted in the plate diagram in Figure 5, our generative model uses p as the starting distribution (for sampling s_0), then for each subsequent t, a binomial with mixing parameter π is used to decide between sampling from $T(\cdot, s_t)$ or independently drawing s_t from p. We use the "starting distribution", p, both for the initial application choice (sampling s_0) and as the second distribution in the mixture model, as users will choose an applications sometimes as the start of a new task, while other times based on an inclination from their current application (e.g., clicking a link in an email) modeled by transition matrix T. Finally, multiple sequence observations $\{s_t^k\}_k$ will be considered i.i.d. samples from this generative model. Formally,

$$\begin{aligned} P[\{s^k\}_k | \pi, T, p] &= \prod_k P[s_t^k | \pi, T, p] \\ &= \prod_k P(s_0^k | p) \prod_t P(z_t | \pi) P(s_t^k | s_{t-1}^k, z, T, p) \\ &= \prod_k p(s_0^k) \prod_t [\pi \times T(s_{t-1}^k, s_t^k) + (1-\pi) \times p(s_t^k)]. \end{aligned} \tag{1}$$

We seek the parameters optimizing the posterior distribution (Maximum A Posteriori or MAP estimate) using conjugate priors as follows: mixing parameter $\pi \sim$ Beta(α), transition matrix rows (multinomials) $T(i,\cdot) \sim$ Dirichlet(β) and multinomial $p \sim$Dirichlet(δ), i.e.,

$$\hat{\pi}, \hat{T}, \hat{p} := \underset{\pi, T, p}{\arg\max}\, P(\pi, T, p | \{s^k\}, \alpha, \beta, \delta) \ . \tag{2}$$

subject to constraints

$$\begin{aligned} \sum_j T(i,j) = 1, &\quad 0 \le T(i,j) \le 1 \\ \sum_j p(j) = 1, &\quad 0 \le p(j) \le 1. \end{aligned} \tag{3}$$

Hence this is an optimization over $1 + n \times (n-1) + (n-1) = (n+1)(n-1)+1$ parameters: π, $\{T(i,j) : i = 1,\dots,n, j = 1,\dots,n-1\}$, and $\{p(j) : j = 1,\dots,n-1\}$, since $T(i,n) = 1 - \sum_{j<n} T(i,j)$, and $p(n) = 1 - \sum_{j<n} p(j)$. We use gradient ascent to optimize this. We note that this is not a sufficient method in general for finding the global maximum—our function to be maximized is not in general concave (or more intuitively, concaved down). Yet, we plot the function for our data to find it at least appears concave in our case. Informed by this analysis, we simply walk up hill using gradient ascent until convergence, and confirm convergence at the visualized maximum.