Network Detection of Interactive SSH Impostors Using Deep Learning

Julien Piet, *UC Berkeley and Corelight;*
Aashish Sharma, *Lawrence Berkeley National Laboratory;*
Vern Paxson, *Corelight and UC Berkeley;* David Wagner, *UC Berkeley*

https://www.usenix.org/conference/usenixsecurity23/presentation/piet

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Network Detection of Interactive SSH Impostors Using Deep Learning

Julien Piet
UC Berkeley and Corelight

Aashish Sharma
Lawrence Berkeley National Laboratory

Vern Paxson
Corelight and UC Berkeley

David Wagner
UC Berkeley

Abstract

Impostors who have stolen a user's SSH login credentials can inflict significant harm to the systems to which the user has remote access. We consider the problem of identifying such imposters when they conduct interactive SSH logins by detecting discrepancies in the timing and sizes of the client-side data packets, which generally reflect the typing dynamics of the person sending keystrokes over the connection.

The problem of keystroke authentication using unknown freeform text has received limited-scale study to date. We develop a supervised approach based on using a transformer (a sequence model from the ML deep learning literature) and a custom "partition layer" that, once trained, takes as input the sequence of client packet timings and lengths, plus a purported user label, and outputs a decision regarding whether the sequence indeed corresponds to that user. We evaluate the model on 5 years of labeled SSH PCAPs (spanning 3,900 users) from a large research institute. While the performance specifics vary with training levels, we find that in all cases the model can catch over 95% of (injected) imposters within the first minutes of a connection, while incurring a manageable level of false positives per day.

1 Introduction

Attackers who compromise the SSH credentials of an enterprise user pose a serious threat due to their ready access to any server on which the user has an account. Such access will often include interactive sessions during which the attacker explores these servers to gauge their utility and assess any data they hold or additional access they can provide. Accordingly, there is a major benefit in being able to rapidly detect such sessions in order to contain and mitigate the damage.

While the strong confidentiality properties of SSH hide the interactive commands such imposters type from network monitoring, it is more difficult for imposters to alter the fine-grained dynamics of their individual keystrokes, which SSH generally sends in distinct packets. In this work we consider the problem of detecting imposters based on differences between their own keystroke dynamics and those of the legitimate users whose credentials they have stolen.

Considerable prior work has addressed keystroke-based authentication in constrained contexts: for users typing either fixed text (such as passwords), or freeform text for which the detector has available the specific characters typed (unlike for network monitoring of SSH) and in some cases for how long each key was pressed. To date, studies of the harder problem of network-based detection using only unknown freeform text have been of a proof-of-principle nature [15, 30], with very limited datasets (4–20 users), and lacking sufficient accuracy for operational viability.

To explore this problem in depth we leverage a singular dataset: 45 months of SSH PCAPs, labeled with usernames, recorded from real traffic at the border of a large research institute (12 TB, 4,000 users). Using a transformer—a sequence model from the ML deep learning literature—and a custom "partition layer", we train a neural network model that authenticates users with high accuracy, relying only on client packet timings and lengths. With ample training data (equivalent to an hour of typing for an average typist), the model can detect 99.2% of injected instances of imposters, while incurring for the site about 9 false positives per day. With much less training data (4 minutes of typing), the model still detects 94% of imposters, with false positives rising to about 30/day. Sites might plausibly deal with these false positives cheaply, either by requiring 2FA re-authentication, or contacting users out-of-band to confirm their activity.

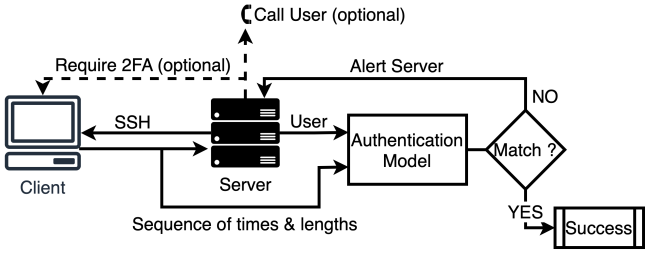

Figure 1: Operational architecture of our detector.

Detection is also quite scalable: the model can process more than 150 interactive sessions per second on hardware costing under $1,500. Figure 1 illustrates the operational use of our detector.

Our main contributions are:

- We can authenticate users in as little as 8 keystrokes, and continuously improve accuracy as the connection gets longer.
- We only require a modest amount of data per user. In our main evaluation, we exceed 96% authentication accuracy when using 4 minutes of typing over interactive SSH per user per week, and still obtain over 90% accuracy when training on as little as 40 seconds of typing over interactive SSH connections per user per week.
- We showcase our models on a 6-year dataset of real SSH traffic, totaling over 600,000 interactive sessions for thousands of different users.

We begin in § 2 with a review of related work. Then, we present our overall approach and its underlying rationale (§ 3), including feature extraction and embedding, and deep-learning architecture. We describe the datasets we employed and the associated processing in § 4. With that in place, § 5 presents our evaluation. We discuss a number of considerations in § 6 and then provide a brief summary (§ 7).

2 Related Work

Our task directly relates to the general field of Keystroke Authentication (KA). After sketching relevant prior work in KA, we also touch upon aspects of anomaly detection and deep learning salient for our effort, and then describe previous efforts at specifically leveraging SSH keystrokes.

2.1 Keystroke Authentication

KA is a subset of behavioral authentication, which instead of centering on concrete authenticators (passwords, 2FA token) assesses a user's activity—ideally forms of activity that the user exhibits unconsciously, and thus has difficulty changing. Such quasi-authenticators can be difficult for an attacker to forge as extracting them may require in-depth data about the user that the attacker wishes to impersonate. For keystroke dynamics, some work has used generative models to imitate typing patterns [20, 40], but the state of the art still comes up short in terms of perfectly fooling authentication systems.

KA also has the advantage of operating in a transparent and noninvasive fashion, potentially occurring in the background of a user performing some other task. KA in addition permits *continuous authentication* (though in this work we do not explore using our models in this fashion).

Because of these appealing features, keystroke authentication (KA) has been a rich area of research for the past two decades. We provide here a short summary of the main directions in this space, and refer the reader to comprehensive systematizations of knowledge for deeper discussion [5, 29, 49, 52].

The two main veins of KA research concern "fixed" [9, 12, 28, 42, 43, 48] and "free" [2, 22, 28, 34, 37] text-based authentication. In fixed-text authentication, users are asked to input a given string of text, whereas free-text authentication aims to identify users regardless of their inputs. Efforts in the literature generally rely on keystroke datasets containing the specific keys and the duration of each key press, as well as the inter-keystroke timing. These works employ main approaches:

Clustering: Early attempts at KA used clustering methods to identify users, recording per-user their typing statistics (key-press durations, and interarrival timings) per key sequence *n-graph*. By building vectors containing statistics about each possible sequence of n keys, when presented with a similar vector for a new input, the detector returns the user with the closest statistics. [9, 12, 22, 28, 28, 44]

Machine Learning: Follow-on work employs ML methods such as SVMs, decision trees, or Bayesian classification. As above, these generally employ per-character statistics [10, 18, 41, 42, 48]

Sequence models: More recent approaches draw upon ideas from language models to build better classifiers. Instead of looking at summary statistics per-character, these works use sequential feature vectors, and algorithms such as Hidden Markov Models, GRUs, LSTMs, or CNNs for classification. [2, 37, 43]

Unlike these prior works, we only have available inter-keystroke timings, not key-press durations nor clear-text characters. Thanks to the use of a transformer model, we overcome this limitation. Only the last approach (sequence models) can work in this setting, albeit this has not been tried in these prior works. We compare the efficacy of our approach to those in [2, 37] in § 5.

Previous work has established that keystroke dynamics leak user information, using keystrokes to identify users' genders [53], emotions [13], or even inferring possible content from inter-keystroke timing [3, 19, 51].

2.2 Anomaly Detection

Anomaly detection aims to identify events that deviate from known prior behavior, comparing notions of expected behavior to new activity (network traffic, in our context). The definition of "normal" comes from domain knowledge [23, 27] or data mining [16, 17, 45].

Anomaly detection has been used to detect imposters that use "pivoting" [7], where the attacker forwards commands

through an intermediate host. That work finds correlations between incoming and outgoing connections to a server to identify pivots. The method can be applied to SSH as well as other protocols, but it can only detect imposters that pivot, and mostly works in internal networks [26]. In contrast, our approach can detect any SSH imposter, including those connecting from external hosts and those that do not pivot.

Some prior work in this space draws upon sequence models to analyze packet payloads or sequences of packets [6, 25, 33, 39]. For SSH, anomaly-based solutions exist for detecting brute-forcing [25, 27] or finding intruders [16, 23]. These last two works have similar goals to ours. SSH-Cure [23] triggers off of patterns of scanning, brute-forcing and compromise in network traffic to identify intrusions. In contrast, we aim to find imposters even if none of these indications are present. The other work [16] uses statistics regarding packet lengths and times to cluster traffic and find anomalous activity, but its evaluation against the Lincoln Labs DARPA IDS dataset—known to be problematic [38]—appears to show limited power.

2.3 Network Classification via Deep Learning

Prior work has used deep learning to develop classifiers for analyzing network activity [1, 4, 24, 35, 50, 55]. These works often classify single network packets and/or focus on identifying application protocols, making them inapt for our detection problem. We note that similar to our method, some prior approaches leverage sequence models to classify network flows [10, 32]. FS-Net [32], in particular, achieves state-of-the-art web application classification results using a GRU architecture. Our goal is different from previous work in this space, but their ideas can be ported to our problem: we adapt and compare FS-Net to our model in § 5.

2.4 SSH Keystroke Analytics

Prior work has examined keystroke dynamics for SSH connections in several ways. The seminal work of Song et al. showed that SSH traffic leaks information about user passwords in its packet timing. [51]. We extend this by showing that packet timing leakage is reliable enough to authenticate users, with the help of deep learning. Guha et al. showed that one can reliably distinguish keystroke packets from other packets in SSH [21]. We employ a related technique to identify keystroke packets and identify interactive sessions based on SSH packet echos.

Finally, three prior efforts tackle tasks quite similar to ours. Koch and Rodosek used clustering to identify users based on their SSH traffic, but their evaluation was limited to 4 different users and showed limited accuracy [30]. Flucke used SSH inter-keystroke timings collected by employing a man-in-the-middle setup [15]; the evaluation is limited to a small, synthetic dataset, and also shows limited accuracy. Nielsen developed a KA approach for SSH [46], but it relies upon server-side decryption to enable standard keystroke-aware techniques, a major deployment hurdle.

In contrast, we achieve high accuracy in a real network with hundreds of active users without relying on traffic decryption.

3 Machine learning models

Our goal is to detect use of stolen user credentials (password or SSH private key) by imposters.

We consider two threat models: credential compromise (attacker learns user's SSH private key or passphrase), and device compromise (attacker gains control of user's device, including SSH private key or passphrase).

We formulate the basic problem as follows. Given an interactive SSH connection (*i.e.* a session in which the user interacts with the server via keystrokes) and the username of the user who purportedly authenticated over it to the server, we would like to know if the actual person using the connection is in fact who they claim to be.

To tackle this authentication task, we use deep learning sequence models to identify patterns in keystroke timings, with a Transformer encoder architecture [54] at the core of our model. Our model solely relies on two features: the sequences of packet lengths and inter-arrival times. We first introduce the intuition behind this feature choice and then describe some custom building blocks specifically designed for working with this sort of network data, enabling us to develop an effective feature embedding. We then describe the final architecture for how we use these building blocks for deep learning.

We implemented our models using the PyTorch library [47]. Our code is publically available here. [1]

3.1 Intuition

To detect imposters we rely on fundamental characteristics of the user—specifically, characteristic patterns in their typing. While SSH encrypts the individual keystrokes typed by a user, rendering them invisible to a network monitor, it still leaks timing information that we can employ to fingerprint specific users. In particular, to provide an interactive experience, SSH sends each keystroke as an individual packet, which the remote server in most circumstances echoes back (also in an individual packet). Thus, although encryption hides the content of each keystroke, we can still recover the inter-keystroke timings. In fact, the mean number of client packets per second in our data is close to the typing speed of an average human, as found in a large study [11].

Given a PCAP recording of an SSH connection, the first challenge is determining whether it includes keystrokes, and, if so, where. Users can employ SSH connections for interactive remote access, data transfers, tunneling of other protocols,

[1] https://github.com/wagner-group/ssh_keystroke_analytics

and/or running remote scripts. In addition, SSH enables multiplexing of multiple instances and types of activity at the same time. We discuss in § 4.3 different approaches we explored for identifying interactive connections; the most effective rests on looking for (1) the telltale "ping-pong" behavior of servers echoing back client keystrokes, and (2) packet rates that lie within bounds consistent with human typing.

For interactive connections, we then skip over the initial SSH handshake (readily identifiable by the sizes and sequencing of the data packets at the beginning of the connection). Subsequent data packets sent by the connection client should correspond to keystrokes.

Due to variances in network round-trip times (RTTs), the relative timing of these packets does not precisely match the original inter-keystroke timing. In addition, very large RTTs (100s of msecs—likely rare on modern networks) may alter how a user types as they switch to waiting to see keystroke echoes before proceeding.

Our detector must (and does) accommodate for (1) both of these potential sources of noise and (2) imperfect filtration of interactive connections.

The sizes of client packets also potentially contain information, such as larger-than-single-keystroke bursts due to escape sequences or the user cut-and-pasting text into their input. SSH somewhat masks the original size of client data, padding packets contents to multiples of 8, 16 or 32 bytes, depending on the chosen cipher. These padding parameters are given in clear text during session negotiation, thus we can provide the model a degraded length estimate to potentially learn from or leverage in classification.

Finally, we include in our threat model attackers who have taken control of a user's device. Accordingly, we exclude the client IP address (or any other device fingerprinting) from the data examined by our detector.

Given the above considerations, we thus reduce SSH connections—if deemed interactive—to a series of interarrival-time/approximate-length pairs reflecting client-to-server post-handshake activity, presumed to mainly correspond to keystrokes.

3.2 Feature embedding: the partition layer

For each SSH connection, we observe a sequence of packet interarrival times and approximate lengths. We need a way to convert this data into a form well-suited for use by an ML model, i.e., a feature embedding. We employ a custom feature embedding designed specifically for classification with network traces, which we call a "partition layer." The partition layer helps build robust, generalizable models by mapping continuous values into a finite number of bins.

Our partition layer discretizes each keystroke interarrival time and length into bins. Because these times and lengths have a highly non-uniform distribution, using them directly as input to the model does not perform well. Instead, we map them to a finite set of unequally sized bins; empirically we find that this improves the quality of the model.

Each bin corresponds to a range of values. Intuitively, this lets us capture notions such as "short keystroke interarrival time" or "large packet," which is useful for our problem: previous work has found that interarrival times fall into different modes that depend on the type of typist, the keyboard layout and the distance between the keys [51].

The key challenge is how to learn an appropriate range of values for each bin. It is not clear how to choose an appropriate set of bins manually, so instead we learn these ranges as we train the model.

With a naive mapping to ranges, this learning task is difficult. The process of discretization is discontinuous and non-differentiable. In particular, if we define a function f_i so that $f_i(x) = 1$ if x falls within the bounds of bin i and 0 otherwise, then the first derivative of f_i is 0 everywhere that it is defined. For standard learning methods based on gradient descent, this property of the derivative of f_i leaves them unable to update the bin bounds.

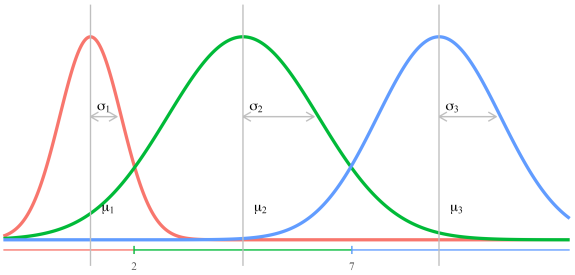

Figure 2: Gaussian functions corresponding to bins $(-\infty, 2]$, $[2, 7]$ and $[7, \infty)$. These functions have six learnable parameters.

We incorporate a custom mapping that lets us learn the optimal range for each bin. The main idea is to learn a "soft" mapping f_i, which is continuous and differentiable everywhere, and smoothly interpolates between 0 (outside the bin) and 1 (inside the bin). In particular, we use a mapping function of the form

$$f_i(x) = e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}},$$

based on a Gaussian probability distribution function, as illustrated in Figure 2. Intuitively, this mapping can be thought of as representing approximately the range $[\mu_i - \sigma_i, \mu_i + \sigma]$; values in that range are mapped to a number close to 1, and values far from that range are mapped to a number close to 0. This transformation "smooths" the problem, since each bin is no longer encoded as a one-hot vector, but as a smooth function of x. Also, this f_i is differentiable everywhere and has a smooth first derivative, which makes it apt for use with deep learning and optimization of each μ_i, σ_i with gradient descent.

In general, we use multiple bins. We map the timing/length value x to the feature vector $(f_0(x), \ldots, f_{c-1}(x))$, where each

f_i corresponds to a different bin.

We discovered empirically that one shortcoming of this approach is that the learning process might fail to learn a set of bins that covers the full range of values of x, and the bins may have gaps or excessive overlap. We solve this last problem by constraining the parameter μ_i for each bin as follows:

$$\mu_{i+1} = \mu_i + s \times \sigma_i, \text{ for } i = 0, 1, \ldots, c-1.$$

Here s is a learnable constant, and $\mu_0 = 0$. (Note that we still allow σ_i to vary.) Thus, as shown in Figure 3, the f_i are dependent, and the center of each bin is a certain number of standard deviations away from the previous one. This ensures that the bins cover a range of values $[0, \mu_c + \sigma_c]$ without gaps or overlaps, and thus they form a soft partition of $[0, \mu_c + \sigma_c]$.

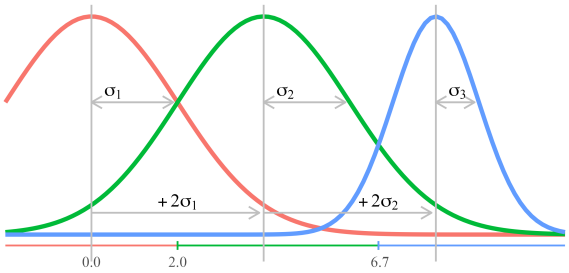

Figure 3: Partition of three dependent Gaussian functions, with gap parameter $s = 2$. The number of parameters reduces from 6 to 4.

We call *partition embedding* a set of dependent bins f_i. In practice, our partition layer uses multiple partition embeddings, and keeps a residual connection to the original input, which allows the model to leverage multiple sets of bins and the raw values. Denoting $\left(\left(f_i^k\right)_{i<c}\right)_{k<p}$ the set of p partitions, each with c bins, the encoding of a value x is the vector $(x, f_0^0(x), \ldots, f_{c-1}^0(x), f_0^1(x), \ldots, f_{c-1}^p(x))$.

Our definition of f_i is similar to Gaussian range encoding [31], but we add constraints to ensure that the bins form a partition. In our experiments, partitions are critical: they not only increase model accuracy by 2 to 3%, but they enable training to converge on difficult tasks with more users—without the partition constraints, training sometimes fails to converge.

3.3 Transformer model

We use a transformer encoder as the model. It accepts as input a fixed-length sequence of values (namely, packet interarrival times and lengths, encoded using the partition layer), and produces as output the prediction for the authentication task. The transformer architecture [54] has become a staple in the sequence modeling space, for its representation power, resilience to noise, and parallelizability. Our problem is a discriminative one, so we only needed to use an encoder and no decoder.

3.4 Model architecture

Figure 4 portrays the model architecture we use for authentication. The input to the model is a sequence of dimension $D_{in} \times S$ with the user encoded as a one-hot vector U of dimension N_U, where N_U is the number of users in the training set, and S is the length of the sequence, with D_{in} values per element of the sequence. In practice, the input is a sequence of packet lengths and interarrival times, so we use $D_{in} = 2$.

In total, the tunable parameters of this model are the number of partitions P, the number of bins per partition B, the length of the input sequence S, and the embedding dimension of the users D_U.

The output is a real value between 0 and 1, indicating the likeliness of the sample being from the alleged user. By default, if the output is above 0.5, we predict that the connection did originate from the purported user, otherwise we predict an imposter. We can adjust this value—coined the output threshold—globally, or even on a per-user basis, to tune the trade-off between false positives vs. false negatives. We illustrate the impact of setting the global threshold in § 5.1.

1. We first normalize the input sequence values to lie between 0 and 1. We divide each packet length by the maximal size of TCP packets in our traces (1,500 bytes). For interarrival times, there is no upper bound, thus we use an arc tangent $\text{Norm}(t) = \arctan ct$, where $c = 20$ to give small keystroke interarrival higher resolution. The output of this layer has the same dimension as the input, namely $D_{in} \times S$.

2. We then embed our inputs using the partition layer. This step takes as inputs the previously normalized values. We use P independent partitions with B bins per partition for packet lengths, and another P partitions of B bins for interarrival times. Thus, the input sequence $((\ell_1, t_1), \ldots, (\ell_S, t_S))$ is mapped to $(y_1, \ldots, y_S)$ where

   $$y_i = (\ell_i, t_i, F_1(\ell_i), \ldots, F_P(\ell_i), G_1(t_i), \ldots, G_P(t_i))$$

   and each F_j, G_j is a separate partition of B bins, i.e., $F_j(x) = (f_{j,1}(x), \ldots, f_{j,B}(x))$ with $f_{j,1}, \ldots, f_{j,B}$ a set of Gaussian mapping functions as described in § 3.2.

3. Next, we prepare the data for the transformer encoder. We first embed the user vector to a dimension D_U. We do this by one-hot encoding the users in a vector, and using a fully connected network to embed it to a lower dimension D_U. The specific dimension does not matter much on the classifier performance. In practice, we use $D_U = 32$. We then concatenate every element in the sequence with this value. The point of doing so instead of simply embedding the user in the first token of the sequence is to have the user be a parameter in every self-attention computation, not just in interactions between the first token and others. We then add padding to each

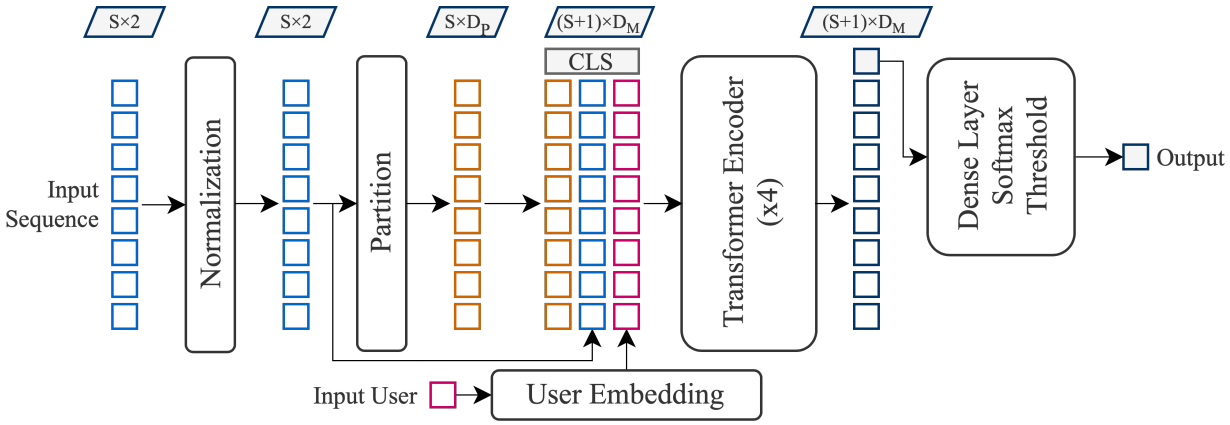

Figure 4: The model architecture. Layer dimensions are in parallelograms, rectangles are layers, and CLS is the class token.

element of the sequence to reach a dimension of $D_M \times S$, where D_M is the input dimension of the encoder.

After padding the vector, we prepend a learnable class token to the sequence, before inputting it to the Transformer encoder layers. In our experiments, we use four layers using 4 separate heads, with a model $D_M = 256$.

4. Finally, we apply the transformer encoder, then pass the output of the class token through a fully connected layer followed by a *sigmoid* to convert to a likelihood.

4 Dataset

Our data contains 632,000 interactive SSH connections, labeled by user, spanning over 6 years of network traffic at the Lawrence Berkeley National Laboratory (LBNL). We extracted these interactive connections from the much larger set of all SSH traffic at the site. We here describe this dataset: its nature (§ 4.1), how we derived labels (§ 4.2), how we filtered all SSH connections to only keep inactive traffic (§ 4.3), and how we converted each interactive session to features (§ 4.4). We then characterize the filtered connections (§ 4.5) and describe the training process (§ 4.6). Appendix A presents a more in-depth characterization of our dataset, including statistics about clients, servers, and users.

4.1 Data collection

LBNL is a large research institute with several thousand users and tens of thousands of systems on their network. The site's security team captures extensive traces and logs to support incident response, forensics, and threat hunting, and has done so for many years.

Our work was approved by LBNL's IT policy team. Since the 1990s, users at LBNL have been required to consent to network monitoring, which the site has used for both security monitoring and a number of research studies. Our primary analysis was all performed on fully anonymized datasets, with payloads stripped and usernames and IP addresses consistently remapped to generic tokens (e.g., "U1" for the first user appearing in a trace). The only exceptions concerned error/accuracy analysis, which when requiring access to the original (non-exported, site-resident) data was conducted under strict confidentiality restrictions by LBNL staff and by affiliates contractually bound by confidentiality requirements.

For this study we drew upon PCAPs of SSH traffic recorded at the site's Internet border, spanning from Aug 2016 through Dec 2022. The traces are made using a system that truncates per-connection capture once the size of a connection exceeds a given cutoff, typically 250KB for our datasets. Due to the heavy-tailed nature of SSH connection sizes, this approach greatly reduces the total volume of recorded traffic, but still leaves us ample per-connection data to work from. In total, our raw data spans 77 months, for a volume of 21 TB.

The number of connections per month varies significantly. As best as we can tell, this is due to variations in the packet capture setup (the syslog data discussed next does not reflect the same degree of fluctuation). We have no reason to believe these changes introduce bias in terms of which users we're able to analyze, but the lulls were large enough that for many months we lacked sufficient training or testing data for all but a small number of users.

In addition to the PCAP traces, we had access to syslogs captured from most of the site's SSH servers. These records allow us to label each inbound SSH connection with the associated username, as well as in most cases a fingerprint of the user's public key. We however are not able to label outbound SSH connections, and do not consider them further in our work.

Finally, we also had access to a full-packet PCAP of 100 days of Telnet/Rlogin traffic captured at LBNL in 2001 from Sep 7 to Dec 17, long archived for potential research purposes, which (after the filtering we describe below) yielded 24,000 interactive connections for 1,300 users. Telnet and Rlogin were SSH predecessor protocols, likewise providing remote login, but without encryption. As such, this dataset

includes cleartext keystrokes as well as the contents of what the server sent in reply. Per § 5.7, we use this dataset to both confirm that our approach works effectively with interactive login protocols in addition to SSH, and to analyze the nature of some of the false positives that our detector generates.

4.2 Deriving labels

We parsed the syslog datasets and matched them via transport 4-tuple to the SSH connections in the network traces. These logs include the username on the server that the client logged into, and in many cases the hash of the SSH key used to authenticate. Thus we could extract usernames as labels, and look for different usernames that share the same SSH key, which are suggestive of aliases. We treat a single username as the same user even when seen across multiple servers, as in general (but not always, see below) the site keeps usernames distinct for different people.

The syslog data is incomplete: some records are simply missing, and some servers do not report them. The most significant of the latter concerns a sister institute that receives a high volume of outbound SSH traffic from the site. In total, about 40% of SSH connections have no corresponding log, mostly due to this latter site as well as some outbound connections. While these lacunae diminish the data we can analyze, we have not identified any likely bias that the absences might induce, and at the end of the day we are still able to label 110M SSH connections.

Using usernames to identify specific people is imperfect, and in principle could muddle our detection since our underlying premise is that keystroke dynamics are tied to people, not usernames. Multiple people can use the same account (e.g., a server's `root` account), the same username on two different servers can correspond to two different people (again, common for `root`), and in general people might use different usernames on different systems. We searched for instances of these situations using the hashed SSH keys and found that such variances are rare, but do happen occasionally. We discuss how their presence affects our results in § 5.1.

Finally, the site's 9 security experts are confident that there are no actual account compromises in the datasets. This is the result of 20 years of SSH security, during which they have implemented multifactor authentication and proactively detect weak passwords and new accounts by mining system logs and brute-forcing accounts.

Even if a handful of compromised accounts occurred in our training sets, causing a few labels to be incorrect, this should prove acceptable since transformer-based models generally work well even in the presence of noisy data.

4.3 Filtering down to interactive connections

Given our problem formulation, we want to focus on only interactive SSH connections, as only interactive sessions carry information about the user's typing patterns. However we lack labels regarding whether a particular SSH connection was used for interactive login or for other purposes (bulk transfer, scripted execution, tunneling). Accordingly, we employ several heuristics to remove likely-non-interactive connections. Table 1 lists these heuristics and their impact on connection volumes.

Filter	Remaining connections
Raw data	115,000,000
Ping-pong behavior	890,000
Sufficient duration	718,000
High rate	667,000
Low rate	632,000

Table 1: Effects of interactivity filters on data volume

Our first heuristic assesses plausible interactivity by looking for the "ping-pong" behavior that interactive sessions will manifest (to at least some degree) due to keystroke echoes, as described in § 3.1. In looking for this behavior, we first identify and remove the handshake part of the SSH connection, which we track by matching to the SSH state machine the sequence, sizes, and directionality of the initial data packets. We require at least 10 ping-pong exchanges after the handshake to consider a flow as potentially interactive.

The second heuristic is to remove connections lasting less than 5 seconds, which removes short-lived scripts and small downloads, but shouldn't remove a significant portion of interactive connections.

These two filters significantly reduce the volume of data. For the initial 110M connections, only 3% last more than 5 seconds, and (separately) fewer than 1% manifest the ping-pong behavior. Of these latter, only 20% last less than 5 seconds, which suggests that our filtering procedures agree on interactive SSH connection identification.

After applying these filters, we obtain 718K SSH connections. From our exploration of the first three months of data, we identified that some non-interactive SSH connections still remained. These arose because automated SSH scripts sometimes exhibit the ping-pong pattern. Removing these is important to ensure that detection only relies on a user's typing patterns (and thus has the greatest potential to generalize) and not on the scripting the user employs.

We developed a heuristic to remove automation by analyzing client-side packet rates. We base this approach on a recent study on 170,000 participants that found that typical users type at an average of 4.3 characters per second (cps) [11]. The study found that fast typists average 8.6 cps (with the very fastest attaining 16.7 cps), and the slowest average 1 cps.

Non-interactive SSH connections can send packets at much higher rates than very quick typists, so we remove connections that send data at a higher rate than is consistent with human typing. For each flow, we estimate cps values by computing the rate of client packets using an 8-character sliding window

around each keystroke. We deem a connection as automated if the first quartile of these values exceeds 15 cps, a value unattainable by most typists. We also remove connections for which the first quartile is under 0.25 cps, as even if interactive these do not contain meaningful inter-keystroke information. These two heuristics remove 51,000 and 35,000 connections, respectively.

In total, this heuristic remove 86,000 automated connections, leaving us with 632,000 labeled SSH connections. It turns out that removing these connections improves classifier accuracy only slightly (about 1%), so it is unlikely this heuristic is biasing our results. However, the more important goal in removing these connnections is to enhance detection robustness, by trying to ensure that the packet arrival times represent user keystroke timings and not other artifacts.

4.4 Features

We extract from the remaining connections the sequence of client packet lengths and interarrival times. As discussed in § 3.1, we refrain from also including metadata such as IP addresses, SSH configurations/versions, or time-of-day, as these features could lead to less robust models in the presence of attackers who subvert a user's own device.

Packet lengths provide a useful feature because they can help the model distinguish keystroke packets (usually small in size) from other packets, such as cut-and-paste of text. However, the lengths we observe in SSH network traffic are imprecise due to padding introduced by the encryption and integrity algorithms chosen for each connection. From study of the SSH RFC [56] and the OpenSSH source code, we identified the relationships between plaintext length vs. ciphertext lengths for the different SSH ciphers and MAC algorithms. Using these relationships we can readily determine, for any ciphertext length, the range of all possible plaintext lengths that could have generated it.

To reflect this uncertainty in our modeling, during feature extraction we randomly sample a number in that possible range of values before applying the feature encoding. Doing so allows our model to generalize across ciphers, so that if a user's SSH connections employ one cipher during training and another cipher at test time, the model should still work well. Adding this small measure of noise to packet lengths also gives the training somewhat more varying data to master, which improves the robustness of the resulting model.

In an ablation study we found that using only packet interarrival times and ignoring packet lengths decreases accuracy by about 5 percentage points. We were not able to train a classifier that relies solely on packet lengths. This shows that interarrival times are the most useful feature, and gives us increased confidence that the model mostly relies on keystroke dynamics, rather than other patterns, to identify users.

4.5 Dataset characterizations and implications

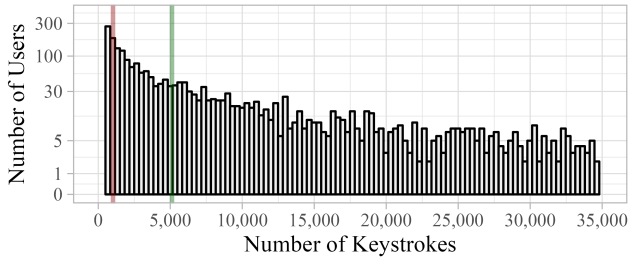

Figure 5: Histogram of keystrokes per user, truncated at 35K keystrokes, log-scaled Y-axis. The red line indicates the fewest keystrokes per user we have successfully trained a model with (1,024), the green line the number required for near-optimal results (5,120).

In total, our data includes 3,900 users. However, many of these are seldom seen. We need a minimum amount of training data to recognize a user, which will preclude developing detectors for rare users. Figure 5 presents a histogram of per-user keystroke count, aggregated over all of their connections. Even for quite limited training we require 1,024 keystrokes per user, but 37% of users do not have enough data over the span of the datasets to satisfy this requirement. For the three months (Aug–Oct 2016) we used to develop our approach, this requirement limited us to 50% of the users seen in this timespan.

While these discarded users might not be identifiable, we can still make use of them as *negative* examples when training detectors for users who do have enough data to provide positive examples. We discuss this further in the next section.

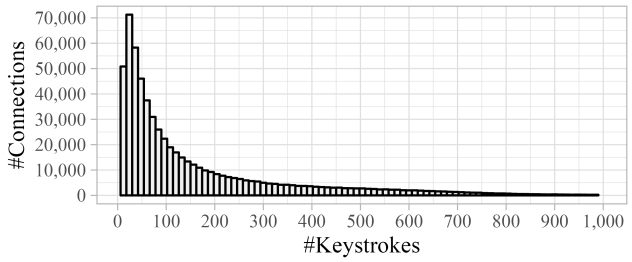

Figure 6: Histogram of keystrokes per SSH connection. The mode lies at 23 keystrokes. Note that this plot only incorporates interactive SSH connections that have passed our previous filters, which is why the histogram lacks very short connections.

Another important consideration for the success of our model is connection length. Very short connections lack sufficient information for accurate classification. Figure 6 shows a histogram of the number of client SSH packets (assumed

to be keystrokes, given our previous filtering) per connection. The plot shows that most connections are short-lived, with a mode of 23 keystrokes per connection, but also with a heavy tail: even though the median is 85 keystrokes per connection, the average is 170 keystrokes. We cut off the histogram at 1,000 keystrokes for clarity (only 0.6% of connections exceed this, with the longest being 2,200 keystrokes). Some connections were likely much larger, but with the remainder unavailable due to the truncation employed by the packet recording, as discussed in § 4.1.

4.6 Training process

In this section we describe how we train the detection model. First, we observe that we should avoid training on entire connections (those truncated at, e.g., 250KB by the packet capture mechanism), lest long connections dominate the dataset. With Figure 6 as guidance, we chose to truncate connections at 512 keystrokes (92% of connections are shorter than this threshold).

Short connections can also cause trouble, since they offer scant information. Our exploration of the first three months revealed that the detector can produce useful results for as little as 8 keystrokes, but not less, so we discarded shorter connections (in practice, a negligible amount of data).

As mentioned previously, we employ users with too little traffic as negative examples, and do not otherwise attempt to model them for detection. We set this threshold based on the user's total number of keystrokes, rather than the number of connections, since connections have variable lengths.

We start by training the model on full-length connections, then fine-tune by subsampling shorter contiguous sequences within each connection, to help the model classify shorter or partial connections. In deployment, to authenticate a user we run the model on the first 512 keystrokes, or on the full connection if shorter. Thus, long connections will only be labeled once they reach 512 keystrokes, which at the average user typing rate will take about two minutes. For prompter decisions, we could stop earlier, at the cost of a drop in performance. We investigate the accuracy of the model for varying input lengths in § 5.

Our dataset exhibits considerable class imbalance: some users have much more data than others. We found that unless we take steps to counteract this, the highest-volume users dominated and the model did not learn the behavior of low-volume users. To address this concern, we weight each class to restore balance. We do so by sampling an equal number of keystroke sequences per user, which, depending on the training keystroke threshold, often results in low-volume users having repeated samples in their training set.

Training requires negative samples (where the actual person typing doesn't match the authenticated username) as well as positive samples (where it does). We generate negative samples by randomly sampling connections from other users (including low-volume users we excluded due to having insufficient data). We sample as many negative examples as the user has positive ones to maintain a balanced training set.

This procedure ensures the model is able to distinguish traffic for a given user from traffic from *any* other user. We verified this by training the model from our first evaluation on 90% of users. Then, we tested this altered model on two test sets: the original testing set, and a testing set only containing the removed users. This second evaluation only has a drop in accuracy of 0.5%: our model can identify imposters even if they were not present in the training set.

For most evaluations we use 3 months of SSH data for training, and evaluate on the following month. The number of users varies, depending on how many users have enough keystrokes in the training data. For training periods where few users appear, real deployments should extend the training period in order to capture enough traffic. (This can arise, for example, for users who usually work from their office at the site—not seen by the border monitoring—but now and then work remotely, or log in during travel, providing occasional visibility.)

5 Evaluation

For our evaluation, we proceed as follows. First, we train a model on 3 months of data, and analyze authentication performance in the following month. We use this setting to evaluate our model's ability to authenticate short connections (§ 5.2), to cope with limited data (§ 5.3), to deal with congestion (§ 5.5), and to remain useful over time (§ 5.4).

Second, we evaluate the real-world usability of our approach by continuously training models during a full year of traffic in § 5.6.

We finish with three auxiliary evaluations. First, we apply the same methods to the Telnet/Rlogin dataset, to examine their applicability to other interactive login protocols, and to illuminate the failure modes of our approach. Second, we briefly discuss our experiences with a complementary classification model, which (once trained) takes as input unlabeled SSH connections and produces as output the name of the user who most likely produced the connection. Third, we compare the performance of our model with ones used in prior work for traffic classification and plaintext keystroke authentication.

5.1 SSH authentication results

For our first evaluation we used training data from Aug–Oct 2016, with the Nov 2016 data as a test set. We retained only users with at least 5,120 keystrokes per user. This corresponds to 10 connections with 512 keystrokes, equivalent to an hour of continuous typing for an average typist, or about 1 minute of typing per workday over the three months. This threshold provided us with a pool of 183 users. We used 12 partitions of 8 bins per dimension. We trained the model

over 30 epochs, with a learning rate of 5×10^{-6}, using the AdamW [36] optimizer and cross-entropy loss.

The model achieved an average per-user accuracy of 94.6%, which we computed by taking the average of user accuracies, to understand how well the model works for a typical user. The median user had an accuracy of 98.1%; 25% of users had an accuracy above 99.3%; and 25% of users were under 94.5%. The average accuracy is around this first quartile figure due to a few hard-to-classify users: the three worst users had accuracies under 66%.

Most errors stem from false positives, i.e., where a legitimate user is falsely accused by the model of being an imposter. The average per-user probability of failing to detect an adversary (false negative rate; FNR) is only 2.0%. While over 25% of users have a FNR under 1%, 8 out of the 183 users with a FNR above 5%, and the most problematic user had a 8.7% FNR. So at worst the detector would miss only 9 out of every 100 imposter interactive SSH connections for any of the 183 users, and on average only misses 2.

The false positive rates (FPR), however, are higher, with a FPR of 8.7% over all connections, though the median per-user FPR is 0.5%. A quarter of the users had no false positives in our evaluation, while the worst quarter experienced 8.5% FPR or higher. These figures suggest that the model can readily learn behaviors *not* expected for a given user, but can struggle at times to fully capture the complete range of actual behavior exhibited by a user.

We manually investigated errors for some of the users with high error rates and found potential explanations for these error rates:

- Some users exhibited large decreases in traffic volume at test time. Although they had sufficient training data, their test set was quite small, and thus a single mistake in a small set of test examples has a big impact on the overall user performance. (This effect also explains why some users with low traffic have perfect accuracy.)

- Users can also exhibit behavioral changes, e.g., due to different keyboard layouts. We found that some problematic users had a disproportionate (compared to other users) number of connections coming from new IP subnets, with previously unseen SSH clients and ciphers. This suggests either the user changed devices, or that someone else is actually using the account.

  Note that our model can still work in the presence of new ciphers, clients, and IP addresses—many users have connections from new subnets with new parameters, and new SSH client implementations, yet are still correctly authenticated.

We explore other possible sources of error using the Telnet/Rlogin dataset; see § 5.7 below.

While we might consider removing troublesome users from the training set after seeing their poor performance on a held-out portion of data, we found that poor performance on held-out training data is not predictive of poor performance in the testing data. We discuss possible mitigations for FPs in § 6.

We also investigated the impact on accuracy of username-to-user mismatches (i.e., incorrect labels). One scenario occurs when multiple different people have the same username. This could be due to different people sharing the same account, or different people with the same username but on different servers (homographs). Using contextual information for each connection, mainly SSH versions, ciphers, clients, endpoint IPs, and public key hashes for non-interactive authentication, we identified 21 accounts potentially shared by multiple users, and one homograph, over the 3 months of training data. Only 9 of the 21 shared accounts had enough traffic to include in our training set, with `root` being one (and achieving a sub-par accuracy of 88%). Of the remaining eight, only two have low accuracy (89.0% and 93.5%). We conclude that shared accounts can have a negative impact on performance, which is best embodied by the `root` account having the highest false negative rate of any user. Fortunately, these seem rare. Homographs seem less of an issue—the one in our dataset obtains near-perfect results.

Another possible scenario arises when a single user has multiple usernames. Using the same contextual information, we found 32 usernames in our 3 month training data that shared an SSH key with at least one other username, suggesting they were used by the same actual person. We found two such pairs that had sufficient data to be included in the dataset. One pair had almost perfect accuracy, while the other consists of `root` and another user with 95.4% accuracy.

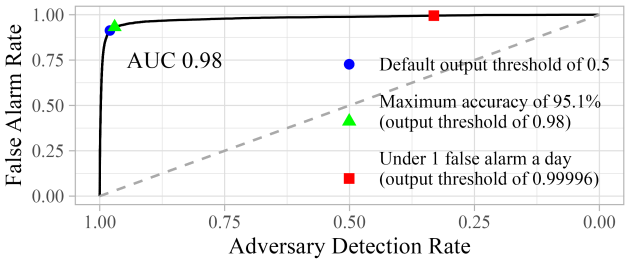

Figure 7: ROC curve for the § 5.1 model.

Impact of output threshold: Our model outputs a real value between 0 and 1. By default, we say a connection is correctly authenticated if this output is under 0.5 (§ 3.4). Increasing this output threshold will reduce the number of false alarms, at the cost of missing some adversaries. Figure 7 plots § 5.1's receiver operating characteristic curve on the testing data. Our default threshold of 0.5 achieves near optimal accuracy. Setting the threshold to 0.98 yields a maximal accuracy of 95.1%. We can tune our model to only get one false positive per day, at the cost of only catching one in three adversaries.

5.2 Time to detection

The discussion to this point has focused on detection for "max-sized" SSH connections, truncated at 512 keystrokes. We also examined the model's detection performance when less data is available at test time. For this evaluation, we limited our assessment to the first n client packets of each evaluated connection, using the same model as previously trained on 5,120 keystrokes per user.

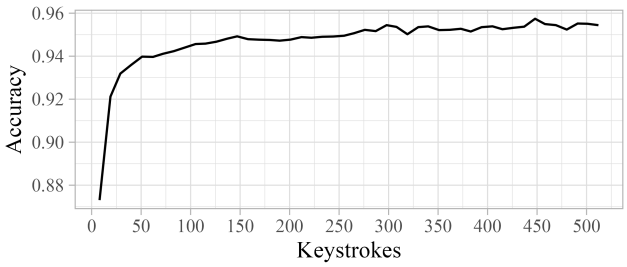

Figure 8: Authentication accuracy on Nov 2016 data, as a function of the number of client keystrokes analyzed.

Figure 8 plots the average per-user accuracy for varying values of n (ordered highest-to-lowest). As we would expect, performance declines for shorter connections, going from 95.5% for the maximal size to 87% for the minimal. However, the drop-off only becomes pronounced below about $n = 70$ keystrokes. This means that when using a model for detection, we can make decisions fairly quickly at the start of new connections, rather than having to wait for them to accrue hundreds of client packets: the median time until the 70^{th} client packet in our test set is 30 sec.

5.3 Training with limited data

Our evaluations above used a threshold of only training models for users with the equivalent of 10 maximal-sized connections, i.e., 5,120 keystrokes' worth of data. We now examine the model's performance when training with different data volumes.

Table 2 lists the results when varying the training threshold but otherwise using the same parameters as before. We find that generally, users with good accuracy for one model also did well in other models. The most data-hungry model, requiring thirty 512-keystroke connections, achieves a false negative rate of under 1% on average, but does not significantly outperform the model from the previous section. In fact, we found that performance of both models is similar for most users, however the worst users in the small model did significantly better than the worst users for the larger model. We see that even training with as few as five 512-keystroke connections, equivalent to 45 seconds of typing per week (for average users), we can get results within 6% of the most data-hungry model, while supporting more than 4 times as many users.

Keystroke Threshold	# Users	Accuracy	Max length accuracy	Min length accuracy	FPR	FNR
15,360	66	96.1%	96.1%	88.9%	7.1%	0.8%
10,240	103	94.9%	95.1%	87.9%	9.0%	1.3%
5,120	183	94.6%	95.5%	87.0%	8.7%	2.0%
2,560	275	90.9%	93.6%	80.7%	12.0%	2.8%
1,024	444	87.0%	92.1%	80.6%	11.5%	6.0%

Table 2: Authentication accuracy, false positive and false negative rates for different user thresholds

5.4 Performance over time

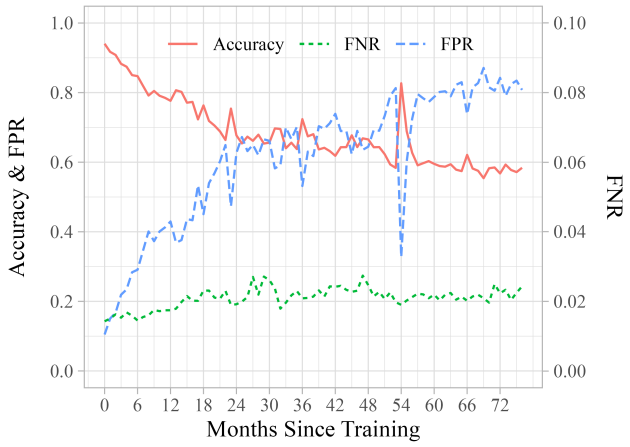

Figure 9: Decay of authentication accuracy over time

We based the above evaluations on training the model on three months of data and then evaluating on the month following the training data. We now examine for how long a trained model remains useful. Using the model trained on Aug–Oct 2016, with a threshold of 5,120 keystrokes per user, we evaluate it on the following 74 months of data. Figure 9 plots the average accuracy, FPR and FNR. While performance degrades considerably as time goes by, the model manages to achieve over 80% accuracy for the first 9 months, and the FNR stays negligible for the entire period.

This latter finding provides some confidence in the training method, as it appears that the model learns what makes each user unique, rather than learning what other traffic looks like. We find it plausible that users change their behaviors over time, so the FPR quickly grows, but the algorithm still achieves quasi one-sided error, even years after training. That said, given the low cost of training, in a real deployment we would still recommend retraining every month or so, to keep up with the evolution of user behavior, as well as to accommodate new users.

The spike in performance during the 54^{th} month is due to an increased volume of traffic from users in the training data.

5.5 Effect of network congestion

Our packet capture at LBNL is a multi-year trace of real network traffic—as such, many of the flows in our dataset exhibit congestion. In this section, we show that our main model from § 5.1 is resilient to network delays caused by traffic congestion.

Congestion is a complex phenomenon. Adding congestion to an existing packet capture has been recognized as a hard problem for years [14], for which we do not know of a solution with a valid theoretical basis. Therefore, we perform natural experiments, leveraging existing congestion already organically present in our dataset and analyzing its impact on detection accuracy, instead of simulating additional congestion, which would not have been well-founded.

We partitioned our data into three categories: category (1) connections have no congestion, category (2) have low congestion ($<$ 10 retransmitted packets), and category (3) have high congestion ($\geq$ 10 retransmitted packets). We find that each category has enough data to support realistic assessments in natural experiments. In particular, over our full dataset, we have:

Category (1): 290K connections, 40 client implementations, 33K client IPs, 2.5K server IPs, 3.6K users.

Category (2): 270K connections, 30 client implementations, 37K client IPs, 2.3K server IPs, 3.3K users.

Category (3): 80K connections, 25 client implementations, 16K client IPs, 1.5K server IPs, 2.2K users.

For the training set on which we developed most of our models (spanning 3 months in 2016), incorporating 183 users with over 5120 keystrokes, we have:

Category (1): 11K connections, 14 client implementations, 1.6K client IPs, 250 server IPs, 183 users.

Category (2): 9.5K connections, 13 client implementations, 1.8K client IPs, 230 server IPs, 181 users.

Category (3): 2.7K connections, 10 client implementations, 700 client IPs, 150 server IPs, 165 users.

Most users and client implementations are present in all three congestion scenarios, so in general the only varying factor between each is the level of congestion. Given that even for the 3-month subset we have well over 150 users in each category, we believe this approach is robust against any small-number effects, and provides insight into how performance varies with network congestion.

Test flows without congestion have average performance: they obtain 95.0% accuracy, with 8.0% false positives and 1.9% false negatives. The model detects flows with little congestion with slightly higher accuracy, classifying these flows correctly in 95.5% of cases, with a false positive rate of 6.7%, and a false negative rate of 2.1%. This finding illustrates the benefits of employing organic data: network artifacts present in traffic add noise to the training set, making the model more robust to future traffic delays.

Finally, the model's performance on high congestion flows is worse than for the previous two categories. It obtains 93.9% accuracy, with 9.5% false positives, and 2.5% false negatives. Although these numbers are close to the original figures, it is clear that heavy congestion makes the problem somewhat more difficult.

5.6 Continuous evaluation

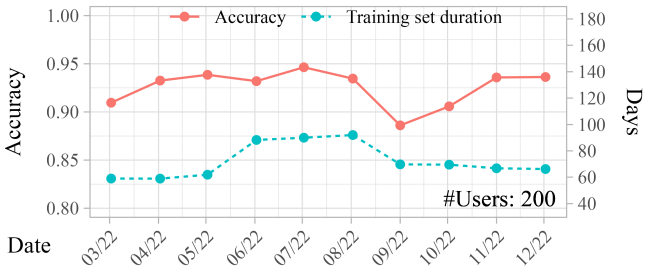

Figure 10: Authentication accuracy per month, when retraining monthly for 200 users, 2022. The right axis gives the number of days in the training set.

We now look at the detection model in an extended setting for all of 2022. We build a new classifier for use in each new month using data from previous months. To deal with the diminished data in some of the PCAPs (per § 4.1), instead of training on a fixed period of time to evaluate on the next month, we fix a target number of users (200), keep the original keystroke per-user minimum (5,120), and determine the start of the training data window with respect to these requirements. Doing so allows us to compare the performance of the model over time in equal standings.

We trained a separate model for each evaluation month, plotting the resulting accuracies and training data duration in Figure 10. The accuracy of the resulting classifiers varies from 88.6% to 94.6%. Interestingly, the worst two months are September and October, which are trained with summer data, during which fewer users are active. The model detects 97% of adversaries, and achieves an overall aggregate FPR of 11.9%. This represents a total of 7,000 false positives over the course of 10 months, or an average of 23 false positives per day.

We can reduce this number of false positives by increasing the output threshold. By using an output threshold of 0.98 (optimal value from § 5.1), the false positive rate goes down to 8.7%, while the model still detects 95% of adversaries. In this setting, the number of false positives per day falls to 17.

We discuss ways to deal with false positives in § 6.

5.7 Telnet/Rlogin results

We also evaluated the same authentication model on the Telnet/Rlogin dataset. We used the first two months of data for training, and the last month for testing, for which we obtained 63 users using a keystroke threshold of 15,320. The model obtained 95.6% accuracy averaged over all users (slightly higher, 97.6%, when only using 512-length samples), demonstrating that the technique works for other interactive protocols.

In addition, because Telnet and Rlogin provide cleartext data, we can directly inspect the model's errors to gain greater insight into their nature. As for SSH, the model has near one-sided error, so we focus on the false positives, which we found arose for several reasons:

- Some users had quite similar sessions (mostly running the Pine email reader), but with client IP addresses from totally different subnets, suggesting differing network dynamics might have altered the user's typing style. For example, the RTTs for one particular problematic user averaged 150 ms in the training set, 130 ms in the true positive test samples, but fell to 20 ms for false positives—highly suggestive of the impact of network dynamics on the user's typing style.
- Some false positives arose from connections that contained virtually no regular keystrokes, instead heavy use of escape sequences (such as up/down arrows) when reading email.
- Switches to different basic types of activity can induce errors. For example, one user went from reading email in earlier sessions to coding. One's keystroke patterns might plausibly change depending on the task at hand.

It seems reasonable that these sorts of changes could also occur in SSH connections, possibly explaining some of the errors we were not able to track down.

Finally, we note that our technique's success for Telnet and Rlogin suggests we may be able to extend the approach to work with other interactive protocols, such as RDP.

5.8 SSH classification

While our main focus has been on the authentication problem given its practical importance, we also extended our approach to develop *classification* models that infer *which* user is associated with an unlabeled SSH. Classification is a harder problem than authentication—and less useful for security purposes, since we usually care about identifying intruders, not classifying unlabeled SSH connections—but we can use classification to shed light on some of the behavior of the authentication models.

In the interest of brevity, we omit the details of the deep learning architecture used for classification as it is quite similar to what we use for authentication (actually slightly simpler).

We use the same setup as for the first authentication evaluation (§ 5.1). Overall, the classifier achieves an accuracy of 78%, which goes up to 86% when only using 512-length samples, and down to 50% for connections of 8 keystrokes. Although these accuracies seem inferior to those achieved for authentication, one should keep in mind that a random classifier in this setting would only achieve an accuracy of $\frac{1}{183} \approx 0.5\%$. With only 8 keystrokes, we can build a classifier 100 times more accurate than random, and if we use as few as 40 keystroke, the accuracy exceeds 75%.

Of more direct interest, we can employ the classification models to analyze more subtle errors exhibited by the authentication models. In particular, we can look more deeply at the effects of users with multiple usernames. The user with `root` access that we discussed earlier in § 5.1 gets classified as `root` in 2.5% of samples—small, but showing some sense of confusion. We also looked at mistakes made by the model when classifying users with little traffic, labeled as "other". We found that two of these low-volume users were in fact systematically mistaken for the same high-volume user, and upon further investigation, discovered that these accounts all reflected the same person, showing how keystroke dynamics are tied to individuals, not specific usernames.

5.9 Comparison to other ML architectures

Prior work has not appeared for our particular problem—keystroke authentication without contents, at scale—but has for network traffic classification and plaintext keystroke authentication, so we undertook of a comparison against these. Using PyTorch [47], we implemented FS-Net [32], which achieves state-of-the-art results for web traffic classification and was designed for sequential data, the CNN+GRU architecture from [37], and the more recent TypeNet architecture [2].

To give the models equal functionality to ours, we added a user input to all three. We also modified FS-Net to support continuous values such as times rather than categorical data, and extended the input vectors used by CNN+GRU and TypeNet to include packet lengths, to give the models equal standing to ours.

Unfortunately we were unable to successfully train the CNN+GRU model. Despite trying different rates and model parameters, we did not find a configuration for it that performed better than random choice. We did however successfully train the other two models.

Table 3 presents the accuracy of TypeNet, FS-Net and our model. The first two rows show authentication accuracy on 66 users, the last two on 183 users, in the same setting as in § 5.3. On the easier task involving 66 unique users, both other models produce good results, though a bit lower than our model. TypeNet performs well on the harder task with 183 users, but struggles to keep up with our results for 512-length samples. FS-Net's accuracy is 6% lower than ours when evaluating on all samples, and 5% lower when restricting to 512-keystroke

# Users	Keystroke Threshold	Evaluation Sample Lengths	TypeNet [2]	FSNet [32]	Our model
66	15,360	All	93.4%	94.9%	96.1%
66	15,360	512	92.7%	95.8%	96.1%
183	5,120	All	92.1%	88.8%	94.6%
183	5,120	512	92.0%	90.6%	95.5%

Table 3: Accuracy of other models on two different authentication tasks

connections. We attribute our better results to the partition layer, which more aptly captures keystroke behavior.

6 Discussion and Limitations

Limitation	Comments
Operational impact of FPs	See § 6.1
Data availability	See § 6.2
User coverage	See § 6.2
Evasion	See § 6.3
Shared usernames	See § 5.1
User behavior change	Model might not capture large changes in user behavior due to device changes or congestion.
Non-explainable results	Deep learning produces results that are hard to interpret.
No real intrusions	Our data does not contain real SSH intrusions.
Data imperfections	Our data lacks ground truth to distinguish interactive use from other uses.

Table 4: Summary of limitations.

Our work raises several considerations and possibilities for follow-on explorations, which we briefly sketch here. We also frame our work's limitations.

6.1 Real world deployment and base-rate fallacy

Practitioners are rightfully wary of seemingly low false positive rates for detectors [8]. In the context of our work, an overwhelming majority of interactive SSH traffic will be legitimate, so even with low false positive rates, our model will generate more false positives than true positives. For example, in our continuous evaluation from § 5.6 the FPR of 8.7% in the second example translates over time to 5,200 false positives. Thus, for the 200 users with the most training data available, analysts would have to cope with an average of 17 false positives per day. When using the standard output threshold, that number rises to 23 false positives per day.

The first way to limit false positives is to increase the output threshold, as discussed in § 5.1. However, this decreases the likelihood of detecting an adversary. Instead, we propose other ways of dealing with false positives.

One approach would be for the site to require the user to reauthenticate using a second factor, terminating the connection if not attended to promptly. A related approach would be to contact the user out-of-band, for example via a cell phone voice conversation; this does not integrate as readily with existing security mechanisms but provides robustness against attackers powerful enough to manipulate the second factor. Both approaches are shown in the operational diagram of our detector (Figure 1).

Alternatively, the detector could only report events to security analysts once a day per user. For the continuous evaluation from § 5.6, this lowers the 17 FPs/day to 11; a few users are responsible for most of the false positives, because of changes in their behavior. More broadly, we might consider setting a threshold on the number of confirmed false positives per user, ignoring subsequent alerts for that user until the next model update. Doing so is somewhat risky, because a powerful attacker who subverts the user's system for a sufficiently long period of time could purposefully add noise to the user's legitimate SSH connections in order to exceed the threshold, and then attack unnoticed. (This scenario however requires the attacker to subvert the user's system's kernel.) In particular, this mitigation would work well against attackers with stolen credentials but who have not compromised one of the user's client devices.

A variation of this idea would be to use an online machine learning model, instead of a fixed one, so it can be updated in the presence of false positives. This, however, could open the door to poisoning attacks.

6.2 Limited data

The authentication model needs to know which user logged in when it sees the flow, which requires it to have access to the server's system logs. This requirement suggests a possible avenue for future work: can unsupervised frameworks detect imposters without system logs for ground truth? Generative deep learning models could learn keystroke dynamics from an unlabeled set of interactive SSH traffic, without needing negative examples. These models would not be able to tell two users from the same company apart, but could differentiate between employees and outsiders.

Another issue concerns low-volume and new users. We note that sites might in practice have considerably more data available per user than we had in our datasets, since our data only included traffic that crossed the Internet border. Internal traffic (local system to local system) could occur in high volume, providing the necessary data much more quickly. In

addition, sites could consider requiring low-volume users to perform an extra multifactor authentication step until they have provided enough data for training.

6.3 Evasion

Attackers who are aware our detector is in use can take steps to try to evade it. If they have control over a user's device for sufficiently long they could record the user's typing patterns and train a generative model to fool our classifier, as developed in [20, 40]. We note however that today these state-of-the-art techniques still do not perform well enough to be worrisome—they only double the likelihood of avoiding detection.

A simpler approach would be for the attacker to modify their SSH client to send packets with timings that match a previously seen user connection, precisely duplicating its keystroke timings (or adding some jitter to complicate identification of the mimicry). This might not always exactly work, given varying packet sizes, and the need to often wait for server responses. A possible countermeasure for this evasion might be to include server packet timings and lengths in our features in addition to the client packets.

Finally, an attacker with full control over the user's device could wait for the user to log into a desired target system and type enough so that the connection is considered legitimate, then hijack the connection to inject the attacker's desired commands. One could counter this technique by using continuous authentication, rather than only authenticating the beginning of login sessions. We leave this for future work.

6.4 Limitations

In addition to the limitations previously discussed (impact of false positives, data availability, partial user coverage, attacker evasion, shared usernames), our approach comes with drawbacks as summarized in Table 4.

User behavior change. Our dataset includes users with multiple types of devices and connections with various amounts of congestion. The model accommodates these variations, but in general the model might not work well for extreme cases such as very high congestion, or users switching from desktop computers to smartphones. The model only learns behaviors present in the training set.

No real intrusions. Our dataset does not contain real intrusions. Despite this, we are confident that our models identify the unique patterns in each user's behavior, as shown by our evaluations, and the user-removal experiment § 4.6.

Data imperfections. We use data from a physical site to demonstrate the applicability of our solution, enduring the imperfections that inevitably accompany large real-world data at scale. Because of the complexity of the site's network, we are likely missing some connections and some system logs, meaning we cannot label all inbound SSH connections. We do not have robust ground truth to distinguish interactive SSH sessions from other forms of SSH, so we have to resort to heuristics. While we have confidence in our data filtering process, we cannot be certain we have fully removed all artifacts, and some non-interactive connections may remain.

7 Summary

We have developed a detection model that leverages keystroke dynamics (client packet sizes and timing) for interactive SSH sessions, with an aim of confirming whether the person generating the keystrokes corresponds to the reported username in the login. Our model uses a deep-learning transformer together with a novel "partition layer." The model is trained on labeled instances from past logins by the given user.

Our evaluation draws upon 6 years of labeled SSH connections from a large research institute. We examine how much training data is needed, finding that the model can achieve high performance when trained on modest levels of data (equivalent to a few minutes of typing by an average user), and remains effective at identifying imposters over several years. However, false positives rise over time due to shifting user behaviors. Re-training the model monthly reduces the number of false positives to about 20 per day (for authenticating the 200 most active users), or about 10 per day if we only report the first alert per user each day. Sites can likely handle these alerts by requiring re-authentication using a second factor, or contacting the detected users out-of-band to confirm their access.

Acknowledgements

We thank LBNL's network security team, in particular Jay Krous for faciliating access to the data, Michael Simitasin for extensive feedback on the paper, and James Welcher for setting up the infrastructure and logs. We are grateful to Corelight's Labs team for their feedback on our work. We are also grateful to our anonymous shepherd and reviewers for their insightful comments. This work was supported by generous gifts from C3 AI, Open Philanthropy and Google.

References

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile Encrypted Traffic Classification using Deep Learning. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE.

[2] Alejandro Acien et al. TypeNet: Deep learning keystroke biometrics. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2021.

[3] Ebenezer Akinyemi Ajayi et al. Keystrokes Timing Analysis and Timing Attacks System on Secure Shell: Instance Based Learning (IBL) Model Approach Revisited. *Advances in Multidisciplinary Research Journal*, 2016.

[4] Isra Al-Turaiki and Najwa Altwaijry. A convolutional neural network for improved anomaly-based network intrusion detection. *Big Data*, 2021.

[5] Md Liakat Ali, John V Monaco, Charles C Tappert, and Meikang Qiu. Keystroke biometric systems for user authentication. *Journal of Signal Processing Systems*, 2017.

[6] Sara A Althubiti, Eric Marcell Jones, and Kaushik Roy. LSTM for anomaly-based network intrusion detection. In *2018 28th International telecommunication networks and applications conference (ITNAC)*.

[7] Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing*, 2020.

[8] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.

[9] S. Bleha, C. Slivinsky, and B. Hussien. Computer-access security systems using keystroke dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990.

[10] Jiahao Cao et al. Fingerprinting SDN applications via encrypted control traffic. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*.

[11] Vivek Dhakal et al. Observations on typing from 136 million keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018.

[12] Salima Douhou and Jan R Magnus. The reliability of user authentication through keystroke dynamics. *Statistica Neerlandica*, 2009.

[13] Clayton Epp, Michael Lippold, and Regan L Mandryk. Identifying emotional states using keystroke dynamics. In *Proceedings of the sigchi conference on human factors in computing systems*, 2011.

[14] Sally Floyd and Vern Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 2001.

[15] Thomas J Flucke. Identification of Users via SSH Timing Attack. Master's thesis, Calpoly, 2020.

[16] Vahid Aghaei Foroushani, Fazlollah Adibnia, and Elham Hojati. Intrusion detection in encrypted accesses with SSH protocol to network public servers. In *2008 International Conference on Computer and Communication Engineering*.

[17] Sahil Garg and Shalini Batra. Fuzzified cuckoo based clustering technique for network anomaly detection. *Computers & Electrical Engineering*, 2018.

[18] Romain Giot, Mohamad El-Abed, Baptiste Hemery, and Christophe Rosenberger. Unconstrained keystroke dynamics authentication with shared secret. *Computers & security*, 2011.

[19] Nahuel González, Enrique P Calot, Jorge S Ierache, and Waldo Hasperué. The Reverse Problem of Keystroke Dynamics: Guessing Typed Text with Keystroke Timings Only. In *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*.

[20] Nahuel González et al. Towards liveness detection in keystroke dynamics: Revealing synthetic forgeries. *Systems and Soft Computing*, 2022.

[21] Saptarshi Guha et al. A streaming statistical algorithm for detection of SSH keystroke packets in TCP connections. Technical report, Purdue Univ Lafayette, 2011.

[22] Daniele Gunetti and Claudia Picardi. Keystroke analysis of free text. *ACM Transactions on Information and System Security (TISSEC)*, 2005.

[23] Rick Hofstede, Luuk Hendriks, Anna Sperotto, and Aiko Pras. SSH compromise detection using NetFlow/IPFIX. *ACM SIGCOMM Computer Communication Review*, 2014.

[24] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. nPrint: A Standard Data Representation for Network Traffic Analysis. *arXiv preprint arXiv:2008.02695*, 2020.

[25] Md Delwar Hossain, Hideya Ochiai, Fall Doudou, and Youki Kadobayashi. SSH and FTP brute-force Attacks Detection in Computer Networks: LSTM and Machine Learning Approaches. In *2020 5th International Conference on Computer and Communication Systems (ICCCS)*.

[26] Martin Husák, Giovanni Apruzzese, Shanchieh Jay Yang, and Gordon Werner. Towards an efficient detection of pivoting activity. In *2021 IFIP/IEEE International Symposium on Integrated Network Management*.

[27] Mobin Javed and Vern Paxson. Detecting stealthy, distributed SSH brute-forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*.

[28] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 1990.

[29] Kevin S Killourhy and Roy A Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*.

[30] Robert Koch and Gabi Dreo Rodosek. User identification in encrypted network communications. In *2010 International Conference on Network and Service Management*.

[31] Bing Li et al. Two-Stream Convolution Augmented Transformer for Human Activity Recognition. *AAAI Conference on Artificial Intelligence*, 2021.

[32] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*.

[33] Jiaxin Liu et al. Deep anomaly detection in packet payload. *Neurocomputing*, 2022.

[34] Ximing Liu, Yingjiu Li, and Robert H Deng. Typingproof: Usable, secure and low-cost two-factor authentication based on keystroke timings. In *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.

[35] Manuel Lopez-Martin et al. Network Traffic Classifier with Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*, 5, 2017.

[36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[37] Xiaofeng Lu et al. Continuous authentication by free-text keystroke based on CNN and RNN. *Computers & Security*, 2020.

[38] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[39] Ali H Mirza and Selin Cosan. Computer network intrusion detection using sequential LSTM neural networks autoencoders. In *2018 26th signal processing and communications applications conference (SIU)*.

[40] John V Monaco, Md Liakat Ali, and Charles C Tappert. Spoofing key-press latencies with a generative keystroke dynamics model. In *2015 IEEE 7th international conference on biometrics theory, applications and systems (BTAS)*.

[41] John V Monaco, Ned Bakelman, Sung-Hyuk Cha, and Charles C Tappert. Recent advances in the development of a long-text-input keystroke biometric authentication system for arbitrary text input. In *2013 European Intelligence and Security Informatics Conference*.

[42] John V. Monaco et al. One-handed Keystroke Biometric Identification Competition. In *2015 International Conference on Biometrics (ICB)*.

[43] John V. Monaco and Charles C. Tappert. The Partially Observable Hidden Markov Model and its Application to Keystroke Dynamics, 2016.

[44] Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 2000.

[45] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, 2007.

[46] Bjørn Ivar Nielsen. Continuous Authentication on an SSH Connection. Master's thesis, NTNU, 2022.

[47] Adam Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. 2019.

[48] N. Pavaday and K.M.S. Soyjaudah. Enhancing performance of Bayes classifier for the hardened password mechanism. In *AFRICON 2007*.

[49] Paulo Henrique Pisani and Ana Carolina Lorena. A systematic review on keystroke dynamics. *Journal of the Brazilian Computer Society*, 2013.

[50] Shahbaz Rezaei and Xin Liu. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Communications Magazine*, 57, 2019.

[51] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *10th USENIX Security Symposium*, 2001.

[52] Pin Shen Teh, Andrew Beng Jin Teoh, and Shigang Yue. A survey of keystroke dynamics biometrics. *The Scientific World Journal*, 2013.

[53] Ioannis Tsimperidis, Avi Arampatzis, and Alexandros Karakos. Keystroke dynamics features for gender recognition. *Digital Investigation*, 2018.

[54] Ashish Vaswani et al. Attention is all you need. *Advances in neural information processing systems*, 2017.

[55] Wei Wang et al. Malware Traffic Classification using Convolutional Neural Network for Representation Learning. In *2017 International Conference on Information Networking*. IEEE.

[56] Tatu Ylonen and Chris Lonvick. The secure shell (SSH) protocol architecture. Technical report, 2006.

A Dataset description

We evaluated our model on extensive operational data collected at the Lawrence Berkeley National Laboratory's border, a large research institute with several thousand users and tens of thousands of systems on their network. Using real data ensures our experiments reflect practical considerations. Our evaluations show that our model achieves high accuracy, despite underlying network delays and variations in user behavior. We now describe the site's interactive SSH traffic, after the filters from Section 4.3, to convey how it is representative of real networks.

Lawrence Berkeley National Laboratory's network comprises thousands of users. As of early 2023, the site sees an average of 18,000 unique devices per day, for an average daily volume of 20 TB crossing the network border.

We observe a total of 632,000 interactive connections, ranging from August 2016 to December 2022. These sessions span 3,900 users, going to 2,750 unique server IPs, and coming from 55,000 unique client IPs. Our dataset includes 39 different SSH client implementations, with a total of 800 client versions. 12% of connections are from internal subnets, located at the site's premises, while the rest comes from external networks.

Network round trip time (RTT) variances add noise to packet timings, and large RTTs can influence keystroke dynamics, as users might wait for packet echoes before typing new characters. We extracted our data from a real network, and therefore it contains a gamut of network delays. The average round trip time is 30ms, with an interquartile range of 23ms. We observe a tail of large RTTs: half the interactive SSH connections have a RTT under 13ms, but the 5% largest are above 150ms, with a maximal observed RTT of almost 4 seconds.

Our models are resilient to RTT variations. We did not find bad performance to correlate to large variations in the training set. However, for our Telnet analysis we found that large differences in RTT between training samples and testing samples could degrade performance.

Another observable sign of network delays is congestion, visible through packet retransmissions. 45% of connections do not have any retransmissions, while another 42% have a small amount of congestion (between 1 and 10 retransmissions), and 13% of connections experience heavy congestion, with over 10 retransmitted packets. We show in § 5.5 that connections with mild congestion did not impact model performance. Connections with high congestion had a minor impact, decreasing model accuracy by just 2%.

We windowed interarrival times to values attainable by a human typist, between 1 millisecond and 15 seconds, to characterize user typing speeds in our dataset. The average "character-per-second" count is 4.5, close to the observation in [11]. This reinforces our belief that packet inter-arrival times are a good proxy for inter-keystroke timings.

We now describe different entities in our dataset in depth, to emphasize the diversity of our data and the applicability of our method.

Users On average, each user has interactive SSH connections with 3 different servers. 50% of users only interact with 2 servers, while the top 1% connect to over 19 servers, the maximum being account "root", which has been seen interacting with 352 distinct IPs.

The median user is seen from 5 different client IPs, but only 1 client SSH implementation. This most likely means the median user only connects from a single device, but the IP changes due to DHCP.

25% of users have at least two different SSH client implementations, and 1% of them have 5 or more. Many of these implementations are OS-specific, which seems to indicate a significant proportion of users use multiple devices to connect to SSH servers.

Servers Each server receives SSH connections from on average 5 different users with interactive connections. 50% of them are used by a single interactive user. The top 5% see at least 10 users, the maximum being 837 users for a single server. These high-volume servers are used as login platforms, before accessing other resources.

Clients 90% of client IPs are associated with a single user, most likely being DHCP addresses from large pools such as those maintained by ISPs. A small fraction of IPs are recycled by many users, which can be explained by NATs. 95% of IPs only have a single client SSH implementation, which seems to indicate most devices have a single SSH implementation.

Agents The most common SSH software are OpenSSH, used in 90% of interactive connections, and PuTTY, used in 4.5% of connections. Specifically, 18% of interactive connections use the Ubuntu implementation of OpenSSH, 4% use Debian, and 1.5% use a Windows implementation of OpenSSH. In total, 91% of users have a connection that uses some version of OpenSSH.

7.5% of all interactive connections use some Windows implementation (PuTTY, OpenSSH for Windows, or other software), and 22% of users have at least one flow from a Windows machine. In fact, at least 10% of users use both Windows and Unix implementations of SSH.

Smartphone connections (using the JuiceSSH client) only make up 0.03% of all traffic, and are used by less than 1% of all users.