ICS 2011
12th INFORMS Computing Society Conference
Monterey, California, January 9–11, 2011
pp. 000–000

 | ISBN 000-00000-00000
DOI 10.1287/ics.2011.XXXX

A Streaming Statistical Algorithm for Detection of SSH Keystroke Packets in TCP Connections

Saptarshi Guha
Purdue University, West Lafayette IN 47904, USA, sguha@purdue.edu

Paul Kidwell
Lawrence Livermore Laboratories, Livermore CA 94551, USA, kidwellpaul@gmail.com

Ashrith Barthur, William S. Cleveland
Purdue University, West Lafayette IN 47904, USA, {abarthur@purdue.edu, wsc@purdue.edu}

John Gerth
Stanford University, Palo Alto CA 94305, USA, gerth@graphics.stanford.edu

Carter Bullard
QoSient LLC, carter@qosient.com

Abstract A streaming statistical algorithm detects SSH client keystroke packets in a TCP connection on any port. Input data are timestamps and TCP-IP header fields of packets in both directions, measured at a monitor on the path between the hosts. No packet content is included. The algorithm uses the packet dynamics just preceding and following a client packet with data to classify the packet as a keystroke or non-keystroke. The dynamics are described by classification variables derived from the arrival timestamps and the packet data sizes, sequence numbers, acknowledgement numbers, and flags. The algorithm succeeds because a keystroke creates an identifiable dynamical pattern. Final testing of the algorithm is based on analysis of about 1 million connections covering all common network protocols. Data visualization and the statistical design of experiments play a critical role in the analysis. It is common to treat the choice of tuning parameters of a statistical or machine learning algorithm as an optimization that finds one set of parameter values. Instead, we run a designed experiment that treats the tuning parameters as statistical tuning factors, which yields valuable information about algorithm performance. One application of the algorithm is identification of any TCP connection as an SSH interactive session, allowing detection of backdoor SSH servers. More generally, the algorithm demonstrates the potential for the use of detailed packet dynamics to classify connections, important for network security. The algorithm is has been prototyped in the widely-used Argus traffic audit software system.

Keywords Network security, statistics, interactive SSH session, Argus network traffic auditor, backdoor login, design of experiments, data visualization

1. Introduction

The Secure Shell (SSH) protocol[15] is a widely-used, encrypted, method for Internet communication. Initially developed to replace the unencrypted Telnet protocol, SSH provides not only interactive command-line access, but also file transfer via SCP and SFTP and a general tunneling mechanism which can be used to forward arbitrary application traffic such as X11. Because of its popularity, SSH servers are subject to intense attacks in the hopes of acquiring login credentials. Intruders may also use SSH protocol via non-standard service

ports in order to instantiate backdoors on compromised machines. Although any intrusion is worrisome, compromises where a human being logs in and begins an interactive exploration of the compromised machine is particularly dangerous. Such an intrusion would indicate a specific interest in the victim machine or its normal users, and would warrant an urgent investigation.

Our streaming statistical algorithm detects the SSH client keystroke packets in a TCP connection on any port. This has a number of potential applications, including detecting intrusions. The classification variables used in the algorithm are the following:

- the arrival order at the monitor, a positive integer, of each packet
- whether a packet is from the client or server
- whether there are more than 22 packets in the connection
- whether there are client packets after packet 22 or not
- data size for client packets
- data size for server packets
- the number of packets between a client packet with data and the next acknowledging server packet with data
- interarrival time of two successive client packets with data
- interarrival time of two successive echoing server packets
- the ratio of a client interarrival time and the interarrival time of the corresponding echoing server packets
- the number of packets between two successive client packets with data.

These variables constitute a description of the detailed *packet dynamics* of a TCP connection. There exists a large body of literature devoted to the study of packet dynamics in relation to TCP performance and control e.g., [11, 12]. Our variables have some overlap with those of the TCP studies, but are not the same since our goal is network security.

Other work in network security has employed variables that also overlap with ours to detect interactive TCP logins. This past work, discussed in Section 7, uses aggregate statistics across a connection based on certain variables, and not the detailed packet dynamics of our algorithm. Our work uses the detailed dynamics because the goal of packet-level keystroke detection is more fundamental; it is the first algorithm to classify individual packets as keystrokes or not, and detecting interactive TCP logins is just one of many potential applications.

Our algorithm succeeds because a keystroke creates an identifiable dynamical pattern. Such success has already been hypothesized by Donoho et al. [5]:

> There are also other sources of information that we haven't discussed, the key one being the two-way nature of interactive sessions. There is far more information than just the keystrokes on the forward path through the stepping stones, there are also the echoes and command output on the reverse path, and it should be possible to use information about these to substantially improve detection.

This hypothesis is not recent. Progress has been slow on the idea because study of packet dynamics for network security requires detailed, comprehensive statistical analysis of large, complex packet-level databases; just passing over the data and creating statistical summaries does not shed sufficient light on the dynamics. Recently, though there have been major advances in computational environments for analysis of large, complex datasets, and our work has made use of them [7].

It might be thought that aggregate statistics would scale much more readily because the packet dynamics require more detail, at least formally. In fact, working at the packet level allowed us to develop an algorithm that is streaming and easy to implement in a traffic monitor, allowing it to scale to high traffic rates. The algorithm is has been prototyped in

the widely-used Argus traffic audit software system, and the expectation is that it will be very efficient at high traffic rates.

The following is a guide to the article. Section 2 gives a detailed description of the algorithm. Section 3 describes the two-phase process of developing the algorithm from the TCP packet traces collected. The algorithm has 8 tuning factors whose values have a large impact on the performance of the algorithm; the factors are thresholds for variables used in the algorithm for keystroke detection. Section 4 describes the formal test regimens and data. Section 5 describes a designed experiment that varies the tuning factors to determine their impact on false positives and false negatives in classifying a TCP connection as interactive or non-interactive. Section 6 investigates the accuracy of classifying a client packet as a keystroke or not, using 36 scripted SSH connections. Section 7 presents past work in connection classification and the relationship to the work here. Section 8 discusses justification, results, limitations, and future work.

2. Details of the Algorithm

The algorithm consists of a sequence of 10 rules, applied in the order described below. The first two are SSH-Protocol Rules, Rules I and II, and the next 8 are Packet-Dynamics Rules, Rules 1 to 8.

A keystroke is defined as a key depression that is part of typing to create text, including the `Enter/Return` that ends the line. The definition of typing excludes key depressions that result in navigation through a document being viewed. Key depressions that are part of the SSH handshake are not considered as keystrokes to be detected by algorithm for reasons given below.

2.1. SSH-Protocol Rules

These rules are based on the specifications of the SSH protocol and not the statistical behavior of the classification variables.

Rule I. SSH Handshake Startup Deletion. Absence of Client Packets with Data. *Action: Packets with arrival order 1 to 22 are deleted and the remainder of the algorithm begins with the packet whose arrival order is 23. If there are more then 22 packets, but none are client packets with data, then the algorithm returns this status.*

The SSH handshake has keystrokes, for example, typing a password. However the resulting packet dynamics are very different from those of keystrokes once the SSH session is established. Furthermore, the handshake occurs for both interactive and non-interactive SSH connections, which have no keystrokes after the password. So the keystrokes of the handshake are not a target of the algorithm, that is, that are not keystroke true positives. It is best then, to minimize false positives, to remove as much of the handshake as possible. In doing so, we do not want to remove packets not a part of the handshake. The SSH protocol suggests that the minimum number of packets that can occur in the handshake is 22. So the first 22 packets are not considered by the algorithm.

Rule II. SSH Packet Size. *Action: Packets with arrival orders 23 and above that have data, must have all data sizes equal to $4k$ where k is a positive integer. If any packet is not one of these sizes, then all client data packets are classified as non-keystrokes.*

The SSH protocol specifies that the data in packets have the above sizes [16].

2.2. Packet-Dynamics Rules

These rules center on the classifying of the client packets with data in order of their arrival, each starting out as a keystroke candidate. If a candidate does not pass a rule, it is classified as a non-keystroke and its candidacy ends. In other words, to be classified as a keystroke,

the packet must pass Rules 1 to 8. There are 8 statistical tuning factors that provide classification limits for classification variables. The conceptual framework is that keystroke packets have high probability to lie within the limits, and the non-keystroke packets have a high probability to lie outside. The values of the limits are chosen based on the statistical properties of the classification variables and the statistical performance of the algorithm.

Rule 1. Minimum Client Data Size. *Classification variable d_c = size of data in client packet. Statistical tuning factor $d_c^{(m)}$. Action: Candidate passes if $d_c \geq d_c^{(m)}$.*

Even though the current candidate data size d_c conforms to the SSH sizes of Rule II, there is a smaller range of likely values of d_c. $d_c^{(m)}$ is a statistical tuning factor that specifies the minimum value of the likely range for d_c.

Rule 2. Maximum Client Data Size. *Statistical tuning factor $d_c^{(M)}$. Action: Candidate passes if $d_c \leq d_c^{(M)}$.*

$d_c^{(M)}$ is a statistical tuning factor that specifies the maximum value of the likely range for d_c.

Rule 3. Maximum Server Echo Gap Size. *Classification variable g_s = server echo gap count. Statistical tuning factor $g_s^{(M)}$. Action: candidate passes if $g_s \leq g_s^{(M)}$.*

The server echo packet of the current candidate packet, which is determined by matching the sequence number of the latter with the acknowledgement number of the former, plays a part in the rules. The number of packets between the current candidate packet and its server echo is the echo gap size, g_s. For a keystroke, the gap is not large. The maximum, $g_s^{(M)}$, is a statistical tuning factor for the maximum of the likely range.

Rule 4. Minimum Server Echo Data Size. *Classification variable d_s = size of data in server echo packet. Statistical tuning factor $d_s^{(m)}$. Action: candidate passes if $d_s > d_s^{(m)}$.*

As with d_c, the server echo data size d_s has a likely range of observed values. $d_s^{(m)}$ is a statistical tuning factor that specifies the minimum value of the likely range for d_s.

Rule 5. Maximum Server Echo Data Size. *Statistical tuning factor $d_s^{(M)}$ Action: candidate passes if $d_s \leq d_s^{(M)}$.*

$d_s^{(M)}$ is a statistical tuning factor that specifies the maximum value of the likely range for d_s.

Rule 6. Minimum Client Candidate Interarrival Time. *Classification variable i_s = interarrival time between the candidate packet and the last packet before the candidate that passed Rule 5. Statistical tuning factor $i_c^{(m)}$. Action: Candidate passes if $i_c \geq i_c^{(m)}$. If the current candidate packet is the first of the connection with $d_c > 0$, then it automatically passes.*

Client keystroke packets created by typing in an interactive SSH login have cadences that speed up and slow down. There are typing bursts, short cognitive pauses, and long cognitive pauses. Thus, i_c can vary substantially for true keystrokes. However, it has a lower bound determined by the limit of human typing speed. The statistical tuning factor $i_c^{(m)}$ is a lower bound of the likely interarrival time for human typing.

Rule 7. Maximum Absolute Log Interarrival Ratio. *Variable i_s = the interarrival time between the two server echos for the two client candidates in Rule 6. Classification variable $\ell_{cs} = |\log_{10}(i_c/i_s)|$. Statistical tuning factor $\ell_{cs}^{(M)}$. Action: Candidate passes if $\ell_{cs} \leq \ell_{cs}^{(M)}$. If the current candidate packet is the first of the connection with $d_c > 0$, then it automatically passes.*

Under the assumption that jitter observed at the monitor is negligible, the ratio of i_c and i_s would be close to 1 whatever the absolute value of i_c. The statistical tuning factor $|\log_{10}(i_c/i_s)|$ places a limit on the deviation from 1.

Rule 8. Maximum Previous-Current Gap. *Classification variable g_{pc} = the previous-current gap, the count of packets between the previous and current candidates. Statistical tuning factor $g_{pc}^{(M)}$. The current candidate can pass this rule from the initial evaluation, making it a keystroke positive, or it can be deferred, making it a tentative candidate until the next candidate packet has its initial evaluation. In either case the current candidate will become the previous candidate when a new current candidate is encountered. The following actions are based on the state of the previous candidate and value of g_{pc}:*

(1) State: $g_{pc} > g_{pc}^{(M)}$, previous = tentative.
Action: previous fails; current becomes tentative.
(2) State: $g_{pc} > g_{pc}^{(M)}$, previous = passed.
Action: current becomes tentative.
(3) State: $g_{pc} \leq g_{pc}^{(M)}$, previous = tentative.
Action: previous passes; current passes.
(4) State: $g_{pc} \leq g_{pc}^{(M)}$, previous = passed.
Action: current passes.

The rule is based on behaviors during an interactive session. The typical pattern is an alternation between typing of text at the client and the sending of command output by the server. During a typing episode, a common transmission pattern seen by the monitor is a keystroke from the client, an echo from the server, an ACK from the client, and then the next keystroke. If this happens for the previous and current packets, $g_{pc} = 2$. $g_{pc}^{(M)}$ is a statistical tuning factor that is the maximum of the likely range of values of gaps between keystrokes during a typing episode. During a server output episode, the pattern tends to be packets from the server to the client carrying the output, and ACKs from the client. This tends to create a gap between the two keystrokes occurring before that is larger than $g_{pc}^{(M)}$. The principle of the rule is the following: if the two gaps before and after a candidate packet are larger than $g_{pc}^{(M)}$, then it unlikely that the candidate is a keystroke because it would imply output from typing one letter, whereas a normal line requires at least one character plus `Enter/Return`.

3. Packet Trace Collection and Algorithm Development

3.1. Packet Trace Collection

Our research required packet traces — arrival timestamps and TCP/IP headers — on a gateway link carrying both directions of traffic between an inside network and the outside. The requirement corresponds to the targeted initial application of our keystroke algorithm, which is protection of an inside network by a monitor on the gateway link.

Furthermore, the research required trace collections for two types of connections: *commodity* and *scripted*. The former are the everyday traffic on a link carrying out the many applications running on the inside network. The latter are interactive connections initiated specifically for development and testing of the algorithm; they consist of prescribed commands and prescribed text input to programs that create or display documents. The commodity connections enable testing the classification of connections as interactive or non-interactive. The scripted connections enable testing of both the classification of connections, and the classification of client packets with data as keystroke or non-keystroke.

Traces of commodity connections were first obtained from a previous collection of 4 days 18 hours from the University of Leipzig Internet access link [13]. The inside network in this case is the University of Leipzig campus and certain off-campus subnets.

The research also needed trace collection for an inside network for which we could access log files for SSH to allow us to have highly accurate determinations of whether a port 22 connection was interactive or non-interactive.

We set up trace collection on a subnet of the Purdue Statistics Department. A monitor collected traces with `tcpdump` running on a server connected to the span port of a switch that sees all traffic in and out of the subnet and between two virtual LANS making up the subnet. Both commodity and scripted connections were collected. For connections with inside host port 22 we were able to access log files. But logging for the OpenSSH 5.3 server `sshd` was inadequate for verifying the correct classification, so simple modifications were made to the source code to emit additional messages for each connection. These messages captured the details of authentication and session characteristics, such as whether it was a login, single command, or subsystem request; had an allocated pseudo-tty; had requested port forwarding or X-window tunneling. Additional log messages recorded the intra-session forwarding and tunneling activity and, at session close, summary statistics on data transfer. This additional logging permits independent determination of SSH session attributes. In particular, it enabled highly accurate determinations of whether a connection with inside host port 22 was interactive or non-interactive. For scripted connections at inside host port 22, the `ssh` client program was modified to emit a tracer UDP packet every time it sent a keystroke packet. These tracer packets were collected by the subnet monitor along with packets from the SSH session, yielding a stream where the location of keystroke packets for scripted connections was precisely known.

3.2. Algorithm Development

Our development of the keystroke classification algorithm was done together with the development of its use in an application: classification of a connection as interactive or not.

The first phase of the work was an exploratory, unstructured study of traffic using both visualization methods and numeric methods of statistical analysis to gain a basic understanding of connection packet dynamics: arrival times, packet sizes, flags, sequence numbers, and acknowledgement numbers, as well as secondary variables derived from these primary variables. There was a comparison of the dynamics when keystrokes occurred and when they did not using the commodity traces from Leipzig, and commodity and scripted connections from Purdue. Conclusions from this empirical study, guided by descriptions of the SSH Architecture[15] and of Transport[16] Protocols, led to a succession of versions ending in the the streaming rules-based statistical algorithm described in Section 2. Details of this first phase work are not conveyed here.

The second phase of work was formal testing to verify previous conclusions, and to select certain tuning factors of the algorithm. This used commodity and scripted connections from Purdue, and is described in Sections 4-6.

4. Formal Test Regimens

The first data source for the formal tests consisted of 12 precisely defined interactive scripts, each resulting in a connection. They ranged from the simple command `ls` to more complex activities using the `emacs` editor or utilizing sequences of commands. Each script was executed 3 separate times yielding 36 connections. These scripted connections enabled the determination of false positives and false negatives for classification of client packets with data into keystroke and non-keystroke. In addition, the 36 connections enabled, for the application of of classification of connections into interactive and non-interactive, determination of the number of false negatives, `fn.22script`, among the 36. This is connection classification Test Regimen 1.

The second data source was five days of traffic collection on the Purdue subnet monitor, which resulted in 1,021,336 commodity connections.

Of the 7964 commodity connections which had one host using port 22, Rule I eliminated the 6672 with fewer than 23 packets. Of those, 207 connections with no client data packets were dropped. Rule II, which checks conformance of packet with SSH protocol eliminated

another 38. Of these we were able to resolve all but one as actual packet errors leading to session termination. We were unable to deploy the revised `sshd` on all inside hosts and certainly not on any external servers, which reduced the 1047 commodity port 22 connections to 369 verifiable SSH sessions; 195 interactive logins and 174 non-interactive sessions (123 single commands and 51 sftp transfers). Performance of the classification of the 195 interactive connections as interactive or non-interactive is measured by the number of false negatives, `fn.22`. This is connection classification Test Regimen 2. Performance of the classification of the 174 non-interactive connections as interactive or non-interactive is measured by the number of false positives, `fp.22`.This is connection classification Test Regimen 3.

Of the 1,013,372 commodity connections which did not involve port 22, Rule I eliminated 703,353 with fewer than 23 packets leaving 310,019 connections which were all assumed to be non-interactive for purposes of the test. Of those, 150,228 connections with no client data packets after packet 22 were dropped. Application of Rule II dropped another 158,516 leaving 1275 candidate connections. Performance of the classification of these connections as interactive or non-interactive is measured by the number of false positives, `fp.not22`. This is connection classification Test Regimen 4.

5. A Multi-Factor Designed Experiment

The Packet-Dynamics Rules have 8 statistical tuning factors, as described in Section 2, that are thresholds for the variables used in the rules. The factors, their mathematical notation, and their units of measurement are shown in the first 3 columns of Table 1. Performance of the algorithm is measured by the 4 responses — `fp.not22`, `fp.22`, `fn.22`, and `fn.22script` — described in Section 4. We ran a multi-response designed experiment to determine the dependence of the responses on the levels of the statistical tuning factors.

In the course of our pilot studies of performance we developed insight about ranges of the statistical factors for which the performance is quite good. Based on this we designed and ran a multi-factor fractional factorial designed experiment that varied all 8 factors in a systematic way in a region deemed to have reasonable performance, and studied how the above 4 responses changed with the values of the factors.

The experiment used the 1680 connections remaining after application of the SSH-Protocol Rules to the total 1,021,336 connections. For Test Regimens 1 to 4, remaining are 36, 195, 174, and 1275 connections, respectively. While only 0.16% of the connections remain, an absolute number of 1680 would be far too many for security analysts to review, making the Packet-Dynamics Rules critical. The Packet-Dynamics Rules were applied to the first 1500 packets of each of the 1680 connections.

The choice of tuning parameters for statistical models and algorithms is often approached in the statistics and machine learning literature as just an optimization: the best choice of the values of the tuning parameters for the responses. Running a designed experiment goes well beyond this by providing valuable knowledge about the impact of the factors on the responses, the relationship of the responses as the factors change, the sensitivity of the algorithm to the changes in the values of the factors chosen for the experiment, and the trade-off between false positives and false negatives.

5.1. Experimental Design

The first step in the design was to select 3 values of each statistical tuning factor, which are shown in the last 3 columns of Table 1 so that candidate packet pass-through values increase from left to right. The factors that are minima, shown with superscript m, allow more candidate packets to pass through as their values decrease; their values are ordered largest to smallest. The factors that are maxima, shown with superscript M, allow more candidate packets to pass through as their values increase; their values are ordered smallest to largest.

TABLE 1. Statistical tuning factors and their values in the designed experiment.

Tuning Factor	Notation	Units	1	2	3
Minimum Client Data Size	$d_c^{(m)}$	bytes	48	32	24
Maximum Client Data Size	$d_c^{(M)}$	bytes	64	128	256
Server Echo Gap Size	$g_s^{(M)}$	number	2	3	4
Minimum Server Echo Data Size	$d_s^{(m)}$	bytes	44	32	24
Maximum Server Echo Data Size	$d_s^{(M)}$	bytes	64	128	256
Minimum Client Candidate Interarrival	$i_c^{(m)}$	$\log_{10}$ sec	-0.7	-1	-1.3
Maximum Absolute Log Interarrival Ratio	$\ell_{cs}^{(M)}$	$\lvert\log_{10}$ ratio$\rvert$	0.05	0.075	0.1
Maximum Previous-Current Gap	$g_{pc}^{(M)}$	number	2	4	7

Note. Increasing candidate packet pass-through: $\longrightarrow$

Given 8 statistical tuning factors, there are $3^8 = 6561$ possible combinations requiring one experimental run. A run means applying the algorithm to each of the 1680 connections. We chose a fractional factorial design of with 243 runs from ([14]). It is a minimum-aberration resolution V design that spreads the points across the 8 dimensional space of the statistical tuning factors to enable good characterization of the effects of the factors on the responses. The design has a certain balance of the values of the factors. Each of the 3 values of a factor occurs in 81 runs ($81\times3 = 243$). Each of the 9 combinations of values of 2 factors occur 27 times ($27\times9 = 243$).

5.2. Experimental Results: Dependencies Among the Responses

Each run of the experiment produces a 4-tuple of values of the four response variables. So the 243 runs produce 243 points in a 4-dimensional space. Figure 1 is a scatterplot matrix: all pairwise scatterplots of the four variables. The response names appear in a diagonal of the matrix. The vertical scales of the 4 scatterplots in each row of the matrix are the variable whose name appears in the row. The horizontal scales of the 4 scatterplots in a column of the matrix are the variable whose name appears in the column. Points have been jittered, a small amount of noise added, because a number of plotting locations have multiple points. Given a lower-left origin, the notation for the scatterplot in column i and row j is scatterplot ij. So the lower left scatterplot is 11. Note that scatterplot ij has the same variables as scatterplot ji, but with the scales reversed.

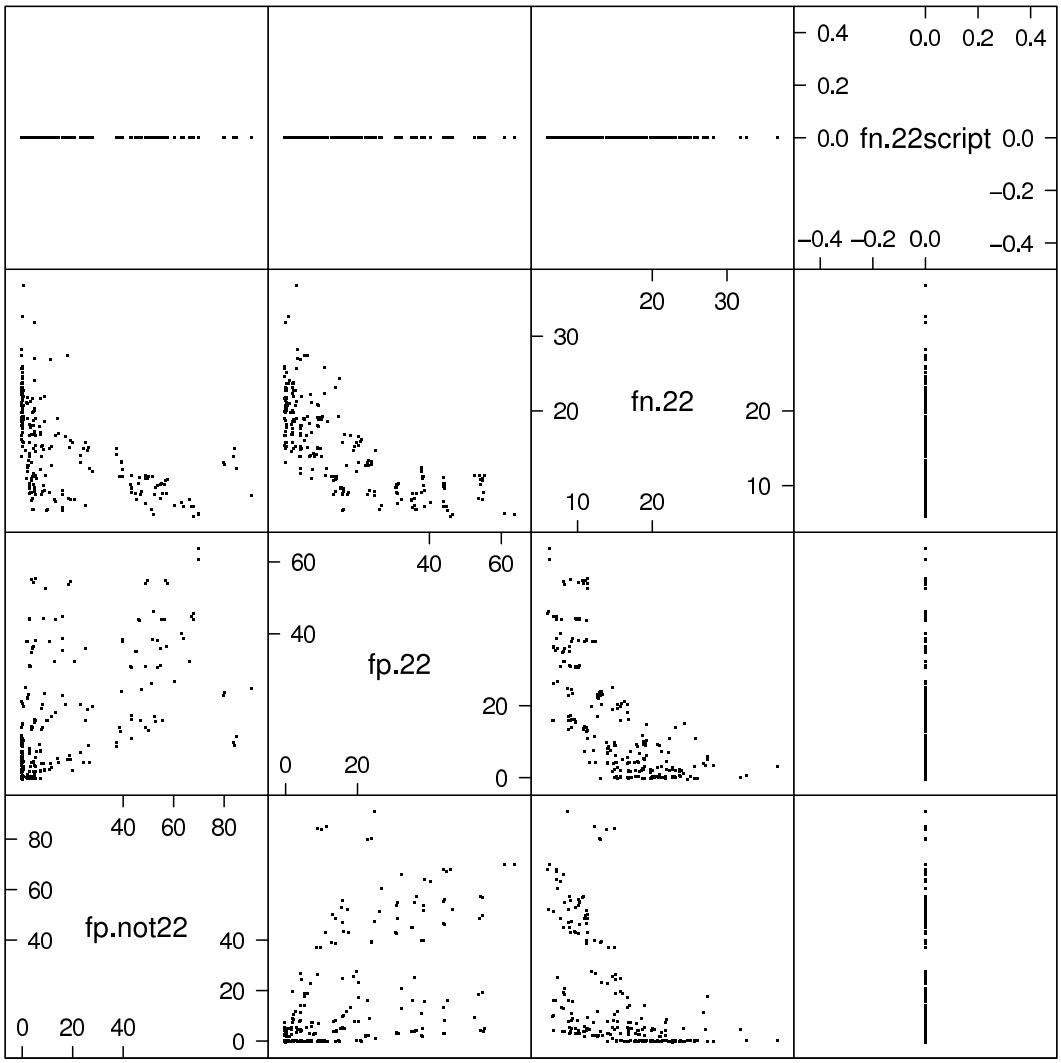
Figure 1 reveals important relationships among the responses. Scatterplot 32 shows a substantial trade-off between `fn.22` and `fp.22`, a strong negative dependence. The same negative dependence occurs for the three other pairs of one false positive and one false negative in scatterplot 31. Scatterplot 21 shows `fp.22` and `fp.not22` have a positive dependence, although there are there are a number of quite low values of `fp.not22` for large values of `fp.22`. This likely occurs because non-interactive connections at port 22, since they are SSH, can behave differently from those at other ports.

What is surprising is the lack of a relationship between `fn.22` and `fn.22script`. This might be occurring because the scripted typing, which requires having to follow prescribed commands under test conditions, is likely somewhat different from that for the every-day commodity connections. However, as we will see, there is a region of the statistical tuning factors for which both these responses are small, which suggests a robustness in the algorithm to different typing conditions.

5.3. Pass-Through Analysis of False-Positives and False-Negatives

The trade-off between false positives and false negatives occurs through different levels of candidate packet pass-through. As discussed earlier, for each rule, as we go left to right in Table 1 through the levels of the statistical tuning factor, more candidate packets pass the

FIGURE 1. Scatterplot matrix of the 4 responses — `fp.22`, `fn.22`, `fp.not22`, and `fn.22script` — for the 243 runs of the fractional factorial designed experiment.

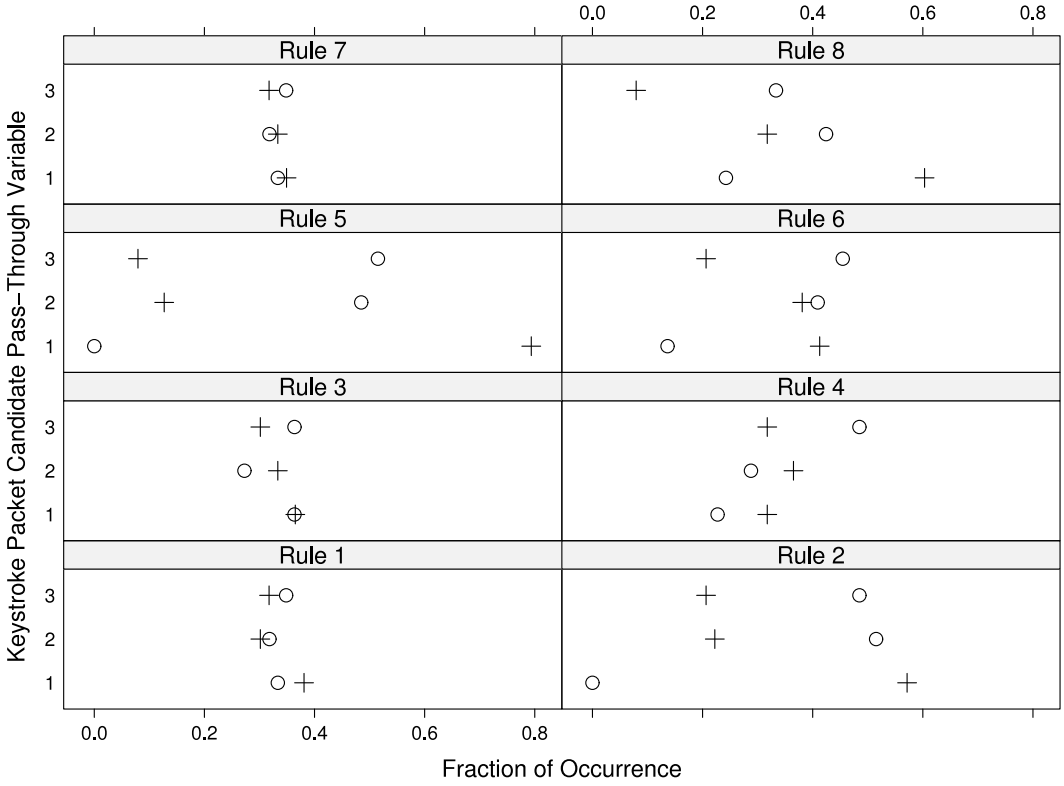


rule. This, in principle, increases the number of false positives and decreases the number of false negatives. We can use this notion to determine the rules whose changing levels are most responsible for the trade-off.

We define a pass-through level variable for the values of the statistical tuning factors: level 1 = the lowest level (column 4 in Table 1), level 2 = the middle level (column 5), and level 3 = the highest level (column 6). Next we select two sets of runs, the good-negative set, for which $\texttt{fn.22} \leq 10$, and the good-positive set, for which $\texttt{fp.22} \leq 2$. The cut-off values were chosen to achieve two goals. The first, which we can see achieved in Figure 1, is that different runs are in the two sets. The second is that the two sets have about the same number of runs; there are 66 in the good-negative and 63 in the good positive. For each rule, the fractions of occurrence of of 1, 2, and 3 are compared for the good negative and the good positive. The fractions are displayed in Figure 2.

Figure 2 shows little to negligible difference between the good-negative and good-positive sets for Rules 1, 3, 4, and 7; these rules play, overall, little role in the trade-off between false positives and false negatives. The other rules do substantially more. The fractions for

FIGURE 2. For each rule, the fraction of occurrences of 3 levels of the pass-through variable for a set of runs with low `fn.22` (∘) and another set of runs with low `fp.22` (+).



the good-negative set tend to be higher for levels 2 and 3, and lower for level 1, than the good-positive set, creating more pass-through. The most dramatic differences are in Rules 2 and 5; pass-through level 1 is never taken by the good negative runs.

5.4. Tuning Parameter Values for Small Values of the Responses

Trade-off of false positives and false negatives is informative because tolerances for each are not necessarily the same for different inside networks. For the Purdue subnet, we seek false negatives as low as possible; a missed intrusion can have major consequences. False positives can be more readily tolerated provided they do not become more than a minor burden for security analysts. With this in mind, we select runs with a partial compromise that gives somewhat higher priority to control of false negatives: `fn.22` ≤ 10, `fp.not22` ≤ 10, `fp.22` $<= 20$, and `fn.22script` $= 0$. This was achieved by 8 runs. Figure 3 displays the fraction of occurrences of the pass-through variable in the same manner as Figure 2. Comparison of the two figures shows that for the compromise runs in Figure 3, there is a greater balancing of pass-through than in Figure 2. One run that is a very reasonable choice from the 8 runs is shown in Table 2 which has 2 values of the pass-through variable at level 2, and 2 each at levels 1 and 3, exactly balancing the pass-through.

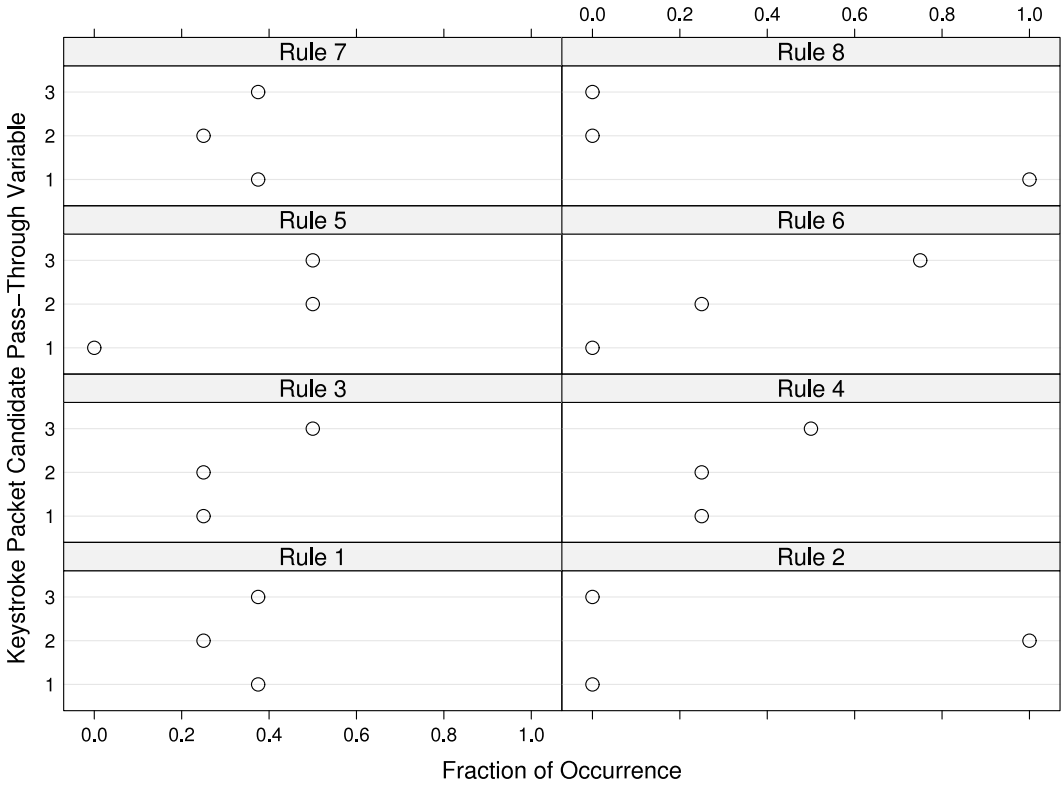
5.5. Rule Impact

The analysis of the 4 responses has given insight about the statistical tuning factors for Rules 1 to 8, showing how much they change the responses over the ranges chosen in the experiment, but this does not speak directly to the impact that each rule has on the classification We studied the impact by analyzing the numbers of dropped packets by the rules.

TABLE 2. Values of pass-through variables to minimize false positives and false negatives, but with a preference for greater control of false negatives.

Rule	1	2	3	4	5	6	7	8
Tuning Factor	$d_c^{(m)}$	$d_c^{(M)}$	$g_s^{(M)}$	$d_s^{(m)}$	$d_s^{(M)}$	$i_c^{(m)}$	$\ell_{cs}^{(M)}$	$g_{pc}^{(M)}$
Pass-Through Variable	1	2	2	3	3	3	1	1

FIGURE 3. For each rule, the fraction of occurrences of 3 levels of the pass-through variable for runs that are a partial compromise of false positives and false negatives, but with somewhat higher priority to control of false negatives.



For each of Rules 1 to 8, the impact metric for a rule based on one connection is the count of client candidate packets dropped by the rule. If after application of a rule, for example Rule 4, no client packets with data remain in the connection, then the values of the metric for succeeding rules, Rules 5 to 8 in our example, are 0. Below we will study the impact metric for a collection of m connections; in this case each rule has m values of the impact metric, and we will study the 8 distributions of metric values, each with m values, to determine the impact. Here, we describe results of the analysis of the impact metric for the connection collections of Test Regimen 3 and 4.

The value of m for Regimen 4 is 1275 (the number of connections). Thus for each of Rules 1 to 8, there are 1275 by 8 counts of packets dropped, one count per connection. If the rule did not drop packets for a connection then the count for the rule is 0. For Rules 1 to 8, the number of connections with nonzero packets dropped are the following:

$$(1)\,182,\ (2)\,1135,\ (3)\,65,\ (4)\,0,\ (5)\,51,\ (6)\,44,\ (7)\,21,\ (8)\,99.$$

Let the above number for Rule k be v_k. The panel for Rule k of Figure 4 graphs the v_k nonzero numbers for each the rule. Let n_i be the i-th largest value of these numbers. Then

$\log_2(n_i)$, where $\log_2$ is log base 2, is plotted against i/v_k. So a fraction i/v_i of the values on the vertical scale are less than or equal to $\log_2(n_i)$.

The values of v_k and Figure 4 show that Rule 2 has the largest impact by far. It dropped packets in 1135 of the 1275 connections, Rule 1 has the next largest impact, and then Rules 8 and 5. Rule 4 appears to have the least impact, with none of the connections affected.

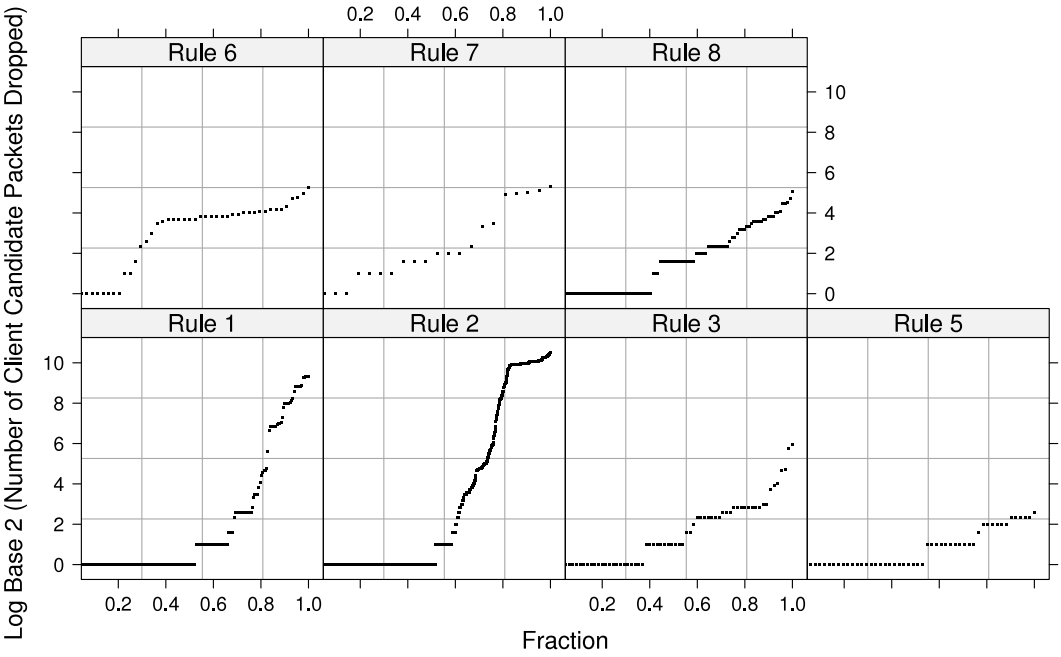
The value of m for Regimen 3 is 174. For Rules 1 to 8, the number of connections with nonzero packets dropped are the following:

$$(1)\,150,\ (2)\,173,\ (3)\,54,\ (4)\,0,\ (5)\,73,\ (6)\,168,\ (7)\,18,\ (8)\,174.$$

Figure 5 displays the nonzero counts of packets dropped using the same display method as Figure 4. In the case, the impact of Rules 1, 2, 6, and 8 are quite substantial. Rule 4 remains as the one no impact at all.

A rule with no impact, such as Rule 7, might be a candidate for removal, but more investigation is needed. It is possible that our design has not covered the best region of the factor space. Increasing the value of the cut off might result in different results. This topic is discussed further in the future work discussion of Section 8.

FIGURE 4. Each panel plots the log base two of the nonzero counts of client keystrokes dropped by one rule for the 1275 connections from Test Regimen 4.

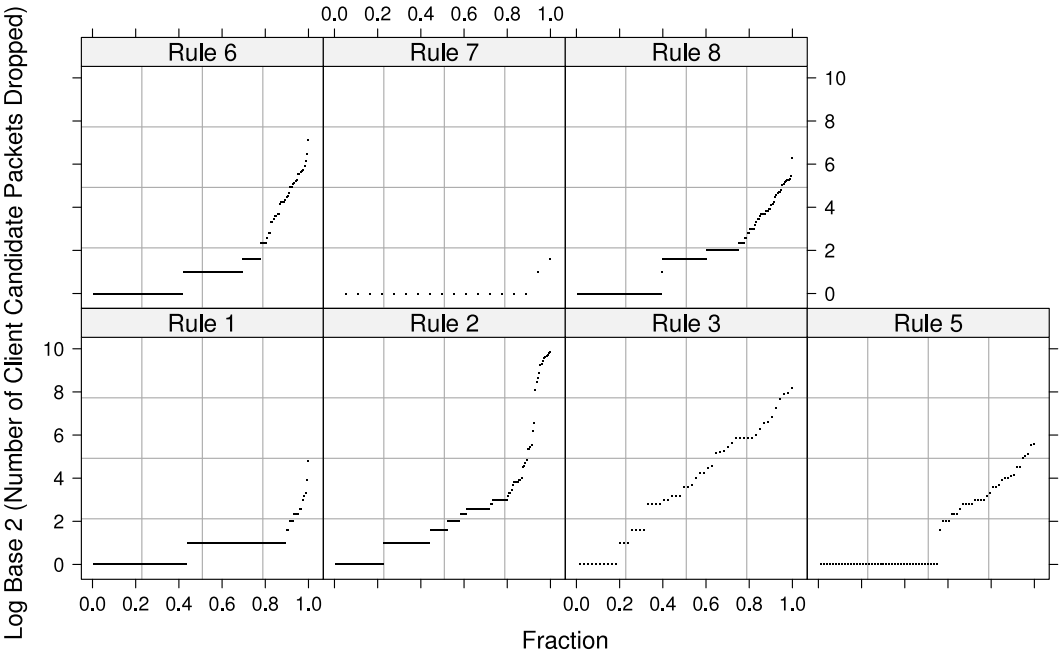


6. Testing the Keystroke Packet Classification

The methodology for Test Regimen 1 allows us to test the accuracy of the classification of client packets with data as keystrokes or non-keystrokes. The SSH connections of the regimen consist of 12 sessions of scripted activities. Each session represented a simple interactive task.

All the activities were performed by one individual from a client outside the Purdue subnet to a server within it. Each session was repeated 3 times for a total of 36 connections. For each connection, the SSH packets and UDP key-press tracer packets were collected. (The

FIGURE 5. Each panel plots the log base 2 of the nonzero counts of client keystrokes dropped by one rule for 176 connections from Test Regimen 3.



tracer packets are used as labels for the client keystroke packets). The algorithm was then applied to the SSH packets using statistical tuning factor values from Table 2.

Client packets containing data are candidates to be keystrokes. In the sessions, candidates are either (1) keystrokes, or (2) document navigation packets. Classification errors in category (1) are false negatives with false positives as category (2).

Each of Sessions 1-3 consisted of a single command resulting in a small number of packets. For example, one session is a command inquiring about the system version information:

```
uname -a <return>
```

Table 3 shows results for Sessions 1-3. There were no false positives or false negatives.

TABLE 3. Test Regimen: Results for Sessions 1 to 3

Session	Keystrokes	False Neg.	False Pos.	Client Data Pkts.	Client Pkts.	Server Data Pkts.	Server Pkts.
	3	0	0	9	25	15	17
1	3	0	0	9	26	16	18
	3	0	0	9	26	16	18
	13	0	0	19	46	25	26
2	13	0	0	19	44	25	26
	13	0	0	19	44	23	24
	9	0	0	15	38	22	24
3	9	0	0	15	37	21	23
	9	0	0	15	37	21	23

Each of Sessions 4-8 consisted of multiple commands over several lines. Compared with Sessions 1-3, there were more keystrokes and sessions were longer. One session was a set of application development build commands.

```
cd tmp <return>
cd git-1.6.3.1 <return>
./configure <return>
```

Table 4 shows results for Sessions 4-8. There were no false positives or false negatives.

TABLE 4. Test Regimen: Results for Sessions 4 to 8

Session	Keystrokes	False Neg.	False Pos.	Client Data Pkts.	Client Pkts.	Server Data Pkts.	Server Pkts.
	32	0	0	38	84	45	47
4	32	0	0	38	85	46	48
	32	0	0	38	84	45	47
	53	0	0	59	262	202	204
5	53	0	0	59	262	202	204
	53	0	0	59	264	204	206
	31	0	0	37	452	414	416
6	31	0	0	37	517	479	481
	31	0	0	37	518	480	482
	203	0	0	209	462	252	254
7	203	0	0	209	460	250	252
	203	0	0	209	462	252	254
	173	0	0	180	1771	1591	1594
8	173	0	0	180	1788	1608	1610
	173	0	0	180	1791	1611	1613

Sessions 9-12 were applications which themselves required users to type text execution, e.g. the `vi` editor. Editors also have key presses that are not keystrokes by our definition, but rather act as navigation or editing aids. For example, pressing the `j` key in `vi` command mode moves the cursor down one line. Table 4 shows results for Sessions 9-12. The causes of false negatives and positives for the session were the same for the 3 connections.

The 3 false negatives in Session 9 (creating a document using the `emacs` editor), came from a text creation character that also cleared the screen. Here, the data size of the server echo for this was larger than the maximum for Rule 5. The remaining two, the keystroke sequence (`[ctrl-x] [ctrl-s]`) which is a command to save the file, caused zero byte server acknowledgments that were discarded by Rule 2.

The false negative in Session 10, was caused by typing `q` to quit a `man` page browser. This keystroke triggered more than 5 packets to be sent between the client and server. This results in a value of g_{pc} larger than the Rule 8 value of $g_{pc}^{(M)} = 2$. In addition, the previous key press was also for navigation and resulted again $g_{pc} > g_{pc}^{(M)} = 2$. This led to Rule 8 rejecting the `q`.

Session 11 consisted of launching `vi` and transcribing a paragraph. The first time `i` was pressed, `vi` switched to text-entry mode, cleared the screen, and changed the status line. The server responded with 1448 bytes and the keystroke was rejected by Rule 5. The first keystroke of the text-mode typing entry was a false negative because its first server acknowledgment contained no data which caused rejection by Rule 4.

In each connection of Session 12, browsing `man` pages, there were 6 false negatives due to the typing of `q` as described above for Session 10. One of the activities in this session was to search for a word in a `man` page. The search was initiated by pressing `/`, typing the search word and pressing `Enter`. The `man` program then found and highlighted the searched word. The packet corresponding to the `Enter` was rejected by Rule 5 because the server response was 1448 bytes. The 2 false positives were the two consecutive presses of the space bar to navigate to the following page.

TABLE 5. Test Regimen 1: Results for Sessions 9 to 12

Session	Keystrokes	False Neg.	False Pos.	Client Data Pkts.	Client Pkts.	Server Data Pkts.	Server Pkts.
	54	3	0	60	163	102	106
9	54	3	0	60	165	104	108
	54	3	0	60	164	103	107
	9	1	0	19	57	37	39
10	9	1	0	19	57	37	39
	9	1	0	19	57	37	39
	338	2	0	344	697	353	357
11	338	2	0	344	697	353	357
	338	2	0	344	696	352	356
	177	7	2	198	490	303	307
12	177	7	2	198	489	301	305
	177	7	2	198	495	306	310

7. Past Work

There is a large literature on classifying network traffic by attributes of its connections or packets rather than payload Examples of this research from Montigny-Leboeuf[3] , Dunigan[6], Hernandez [8], Karagiannis[10], Wright[2], Horton[9], and Alshammari[1] employ statistical and machine-learning methods to cluster and classify traffic by application protocol and across dimensions such as interactive vs. bulk transfer.

Our work is most similar to research in detecting backdoors and stepping stone traffic. Zhang and Paxson[18] label connections with a high proportion of small packets with interarrival times in the range of 10 ms to 2 sec as interactive. It also checks packet lengths for conformance to the SSH protocol: if 75% of the packets meet the specification, the connection is declared SSH. In contrast, our work performs differently by identifying individual keystroke packets.

Timing analysis along with packet size has been used in the detection of stepping stones by Zhang [19] and detection of interactive terminal sessions over stepping stones by Yung [17]. Ding[4] estimated the full RTT for long chains in a stepping stone attack by identifying the interval from the end of command output from the server to the next client input using TCP sequence and acknowledgement numbers. This is similar to our approach although our algorithm analyzes intervals in the reverse (client to server) direction.

Donoho[5] was able to demonstrate that even if traffic is jittered for purposes of evasion, there are theoretical limits on the ability of attackers to disguise traffic and showed a wavelet-based approach for multi-scale detection. The stepping stone detection Donoho was engaged in sought to correlate the inbound and outbound connections of an intermediate stepping stone machine. Our interest is in detecting the traffic at the ultimate target of the attack and being the endpoint of the chain, our algorithm must find patterns within an single connection.

8. Discussion

8.1. Results

The algorithm succeeds because the keystroke and echo packets create an identifiable dynamical pattern. The size and timing relationships of these packets is clearly different from those which are seen for machine-generated traffic.

The SSH protocol plays an important role in the algorithm by dramatically reducing the number of connections under consideration. Here, encryption is our friend. In the first instance, setting up the encrypted session requires almost two dozen packets. This eliminates the majority of connections from consideration. In the second instance, the sizes mandated by the smearing required for encryption create a stark length signature which drops half of the remaining connections. In what remains, only an handful of client packets will qualify as candidates.

And yet, because a human will eventually need to resolve connections classified as interactive, eliminating 99.84% of the traffic still leaves far too much. This is where the Packet-Dynamics Rules become crucial. The formal testing and the experiment to explore the effect of the tuning factors reveal how thresholds can be set to achieve different goals.

One might object that the combination of SSH protocol attributes with features unique to human typing is an isolated opportunity and that studying network traffic dynamics will prove a dead end. We argue that this will not be the case. Machine-to-machine communication appears to be full of patterns induced by applications, libraries, and other software layers providing a rich surface for detecting both expected an unexpected behavior.

8.2. Limitations

The Packet-Dynamics Rules focus on the timing characteristics between client keystroke and server echo and its performance could suffer in conditions of heavy or widely fluctuating load. All of our experience to date has been on networks where (a) there is no substantial jitter for the packets during an individual connection; (b) servers are not so burdened as to have to defer generating echoes; and (c) the monitor is close enough, in terms of network distance, that there are no congestion delays between it and the inside servers. We have tried informally to stress the algorithm by using stepping stones on remote networks, but formal testing is warranted.

The SSH protocol permits the multiplexing of data streams within a single active connection through a variety of tunneling mechanisms. This could result in keystroke traffic being shrouded by other data. In simple instances of X11 tunnels, the algorithm still makes correct decisions, but we did not test all of the myriad possible tunnel and port forwarding configurations.

Of course attackers would try to evade detection by the algorithm. In giving sample evasion scenarios, we divide the rules into roughly three groups: *SSH Protocol:* Rules I and II which limit connections to SSH traffic; *Size:* Rules 1,2,4,5 identify candidate keystrokes by packet size ranges; *Timing:* Rules 3,6,7,8 identify timing characteristics of interactive traffic.

In *Timing*, Rule 6, which depends on the cadences derived from the human factors of typing, can be directly attacked using an unmodified SSH client if the session is launched with the "-T" flag. This prevents the server from allocating a pseudo-tty and forces the client program to assemble keystrokes into entire lines of input before sending them to the server. However, this has significant negative consequences for the attacker because the session no longer appears interactive. Many commands useful for *ad hoc* exploration will simply not work or have their function severely reduced.

To evade *Size* constraints an attacker who is able to modify the SSH client code can convince the algorithm that the session is non-interactive by padding all keystroke packets

to have lengths larger than the maximum client packet size in Rule 2. Such padding would have to be carefully done so as to be discarded by the server unpacking code.

The *SSH Protocol*[16] Rule II requires that **every** keystroke packet have a length which is a multiple of 4. Attacking the algorithm here can be done in the case where the algorithm is running on a machine intermediate to the client and server. The attack involves convincing the monitor that Rule II has been violated even though the victim server believes it hasn't. This could be done if the monitor is several network hops from the victim.. A skilled attacker could inject a chaff packet whose length was not a multiple of 4 but which had a reduced TTL value such that while it would be seen at the monitor, it would expire before reaching the victim. This kind of TTL-manipulation has been used in other intrusion scenarios and there are some simple countermeasures for it.

8.3. Integration with Argus

For it to be useful for everyday network security, our algorithm needs to be part of a network traffic monitor operating in realtime. To this end, we are collaborating with Qosient, LLC (http://qosient.com) to imbed the algorithm in their Argus network traffic sensor. Argus is widely deployed in commercial, government, and educational settings. It has been used to monitor extremely high-speed networks in realtime. The bidirectional architecture makes it well-suited as a platform for packet dynamics. Argus has long had features for measuring packet loss, jitter, and round-trip times in monitored traffic.

The most recent version, Argus 3, adds flexible mechanisms which permit additional attributes to be calculated for connections. We leverage this facility for keystroke detection. The prototype code, now in pilot deployment, consisting of a few hundred lines of C. A few dozen lines constitute the heart of the code which runs over a window consisting of only the previous candidate and current packets. The rest consists of setup, debug tracing, and a small queue which accounts for out-of-order packet arrival.

Acknowledgements This work was supported in part by the Army Research Office MURI Program under award W911NF-08-1-0238, the National Science Foundation under award CCF-0937123, and the U.S. Department of Homeland Security under a Center of Excellence award. The article has been prepared in accordance with LLNL Contract DE-AC52-07NA27344.

References

[1] R. Alshammari, P.T. Lichodzijewski, M. Heywood, and A. N. Zincir-Heywood. Classifying SSH Encrypted Traffic with Minimum Packet Header Features using Genetic Programming. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, pages 2539–2546, New York, NY, USA, 2009. ACM.

[2] W.V. Charles, F. Monrose, and G.M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *Journal of Machine Learning Research*, 7:2745–2769, 2006.

[3] A. De Montigny-Leboeuf. Flow Attributes for use in Traffic Characterization. Technical Report CRC-TN-2005-003, CRC, December 2005.

[4] W. Ding, M. J. Hausknecht, S. Hsuan, S. Huang, and Z. Riggle. Detecting Stepping-Stone Intruders with Long Connection Chains. *International Symposium on Information Assurance and Security*, 2:665–669, 2009.

[5] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *Proc. of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 17–35. Springer, 2002.

[6] T. Dunnigan and G. Ostrouchov. Flow Characterization for Intrusion Detection. Technical report, Oak Ridge National Laboratory, 2000.

[7] S. Guha, R.P. Hafen, and W.S. Cleveland. Visualization Databases for the Analysis of Large Complex Datasets. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, 2009.

[8] F. Hernandez-Campos, F. Donelson-Smith, K. Jeffay, and A.B. Nobel. Understanding Patterns of TCP Connection Usage With Statistical Clustering. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 35–44, 2005.

[9] J. Horton and R. Safavi-Naini. Detecting Policy Violations through Traffic Analysis. In *22nd Annual Computer Security Applications Conference*, pages 109–120, December 2006.

[10] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Sigcomm '05: Proceedings of the 2005 Conference On Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 229–240, New York, NY, USA, 2005. ACM.

[11] V. Paxson. End-to-End Internet Packet Dynamics. In *Proceedings of the ACM SIGCOMM Conference*, pages 139–152, 1997.

[12] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[13] Wand Network Research Group. `http://www.wand.net.nz/wits/leipzig/1/`.

[14] H. Xu. A Catalogue of Three-Level Regular Fractional Factorial Designs. *Metrika*, 62:259–281, 2005.

[15] T. Ylonen and C. Lonvick. RFC 4251: The Secure Shell (SSH) Protocol Architecture. `http://www.ietf.org/rfc/rfc4251.txt`, 2006. This is an electronic document. Date of publication: January, 1996. Date retrieved: April 12, 2010.

[16] T. Ylonen and C. Lonvick. RFC 4253: The Secure Shell (SSH) Transport Layer Protocol. `http://www.ietf.org/rfc/rfc4253.txt`, 2006. This is an electronic document. Date of publication: January, 1996. Date retrieved: April 12, 2010.

[17] K. H. Yung. Detecting Long Connecting Chains of Interactive Terminal Sessions. In *Proc. of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 1–16, 2002.

[18] Y. Zhang and V. Paxson. Detecting Backdoors. In *Proceedings of the 9th USENIX Security Symposium*, pages 157–170, 2000.

[19] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of the 9th USENIX Security Symposium*, pages 171–184, 2000.