An Expectation Maximization Approach to Detecting Compromised Remote Access Accounts

Kevin Gold, Ben Priest, and Kevin M. Carter
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420

Abstract

We present a method for detecting when a user's remote access account has been compromised in such a way that an attacker model can be learned during operations. A Naive Bayes model is built for each user that stores the likelihood for each remote session based on a variety of features available in the access logs. During operation, we leverage Expectation Maximization on new data to update both the user and attacker models, based on the likelihood of the observed session, and perform a model comparison to test for compromise. The system scales linearly with the number of users in computation and memory. We present experimental results on a medium-sized enterprise network of over two thousand users, performing "masquerade detection" in which the activity of one user is discovered within another user's logs.

Introduction

Just as credit-card companies are able to detect aberrant transactions on a customer's credit card, it would be useful to have methods that could automatically detect when a user's login credentials for Virtual Private Network (VPN) access have been compromised. A compromised VPN account is dangerous because it allows an adversary to slip past the network firewall, allowing remote adversaries to attack the network as if they were insiders. In cases where standard methods such as physical tokens have been compromised – as in the recent RSA security breach (Coviello 2012) – it can be useful to have tools that determine the likelihood that an account has been compromised based on its activity alone. We present here a novel method for detecting that a VPN account has been compromised, in a manner that bootstraps a model of the second unauthorized user. Since we do not have records of actual breaches, we model the problem as determining when a user's activity logs have been mixed with the logs of another user.

Several challenges lie in the way of using machine learning to detect account compromise (Sommer and Paxson 2010). First, security professionals expect an extremely low false positive rate from any practical network security tool – hence the popularity of "whitelist" and "blacklist" approaches, which do not generalize well. When scaled to an enterprise of two thousand users, a 1% daily false alarm rate by user translates to 20 false alarms a day, resulting in a tool that will soon be ignored.

Second, when an account is compromised, compromised logins are typically sparse and mixed with good behavior, in such a way that an algorithm or human operator may miss bad behavior amongst the preponderance of good logins. The Expectation Maximization (EM) approach presented here handles this problem by treating the compromised model as a two-user model, in which sessions may either be produced by the original user or a new user; the particular approach used here causes benign sessions to fall out of the likelihood calculations, so that they do not sway a mix of good and bad sessions toward being evaluated as good overall. (We define a "session" as a successful login that ends with either logout or timeout, and a "bad" session is any session in which the user logging in is not the person to whom the account belongs.)

Third, it is easy to detect the presence of threats from known malicious sources, but difficult to defend against new threats. Researchers studying "masquerade detection," or the detection of one user on another user's account, can use an agglomeration of the other users in the training set as an attacker model, then test whether the injection of a specific user's activity can be detected using such a model (Maxion and Townsend 2002; Maxion 2003); but this is a much easier problem than detecting attackers from novel sources, about whom nothing is known a priori. We tackle a harder problem than this prior work by not allowing attacker information to leak into the training data.

Fourth, machine learning algorithms typically assume draws of features from a static distribution, but attackers can change their behavior to emulate good users. Ideally, anomaly detection methods should make it difficult to evade detection even when the algorithm is known to the adversary. Our model makes it difficult for an attacker to emulate a user because doing so would increase the likelihood that an attacker logs in at the same time as the target user, which itself triggers an alarm in the model.

Fifth, rarely is the question of scalability addressed in machine learning applied to masquerade detection; tests on small samples of users can not only hide the intractability of an algorithm at large scales, but can also fool researchers into overfitting their models to their data.

This is not only the first published study of masquerade detection for VPN, but to our knowledge it is also the first work using the masquerade detection methodology run on a user base in the thousands, with machine learning and experimentation methodologies that scale to this number of users. It is the first paper to suggest and implement the EM methodology for extracting attacker models from test data, in a way that bootstraps both the attacker model and the compromised sessions. We evaluate the contributions of the features that can be used to identify users, itself a novel contribution that can aid the design of future login compromise detection schemes. And finally, we report results in a practical sub-1% false positive range.

Related Work

Statistical intrusion detection was introduced as a subject of inquiry in (Denning 1987), which suggested using Chebyshev bounds that detected when user statistics fell outside a standard operating range. "Masquerade detection," or detecting when one user's data had been substituted for another, was introduced in (Schonlau et al. 2001) as a proxy for intrusion detection, with a focus on characterizing users by command line activity. We follow a variant of the masquerade detection evaluation methodology established in (Schonlau et al. 2001), using real users' activity logs as a proxy for attacker compromise.

Probabilistic methods have been used with some success in other masquerade detection and anomaly detection domains, such as command line masquerade detection (Schonlau et al. 2001; Yeung and Ding 2003; Maxion and Townsend 2002; Maxion 2003), web behavior anomaly detection (Xie and Yu 2006), and traffic anomaly detection (Min and Shun-Zheng 2006). In many domains, a common approach is to treat the sequence of observed behavior as a Markov model or hidden Markov model, in which a given event's properties are conditioned on the previous event in the sequence (Schonlau et al. 2001; Min and Shun-Zheng 2006; Xie and Yu 2006). Expectation Maximization (Dempster, Laird, and Rubin 1977) is a general strategy for bootstrapping model parameters in a variety of model structures, though our particular application of it is novel.

Our idea of using Autonomous System Numbers (ASNs) to generalize across user IP addresses for masquerade detection is from (Borders and Oehler 2012). Overlap and session length have been found to be useful in other fraud detection schemes (Bolton and Hand 2002).

Algorithm

The method consists of performing Expectation Maximization (EM) on a set of untrustworthy data to pull out the most likely two-user model, where the user models are Naive Bayes models, as well as probabilities that each session belongs to each user. The likelihood of this two-user model is then compared to the likelihood of a single-user model. We shall go into detail about the Naive Bayes features used first, then describe the EM process.

Naive Bayes Features

Time of day and day of the week: We bin time by hour and estimate $P(Hour|User)$ and $P(Day|User)$ by simply counting occurrences in the data, following "Laplace's Law" of adding one for smoothing (Manning and Schütze 1999).

Overlap probabilities: VPN sessions that overlap are rare enough in the data to be a reasonably reliable signal of compromise. Like the time of day and day of week, we can estimate the probability of detecting an overlapping session from a user by counting how often it occurs in the training data, adding one, and dividing by $N + 1$, where N is the number of sessions in the training data. This is almost always simply $1/(N + 1)$.

Bytes read: Inspection of the distribution of individual users' "bytes read" for a session revealed an exponential-like distribution, with many values near 0 and a decreasing probability of session sizes thereafter, with some outliers in the tens of gigabytes. We model this with an exponential distribution of mean $1/\lambda$, and take the cumulative distribution function (CDF) to the right the number of bytes b, $e^{-b\lambda}$, to be the likelihood of the feature for a session. In other words, our feature is having "at least as many" bytes as what is observed, so that we alarm on bytes read that are extreme, and have a likelihood of 1 for this feature when the observed bytes read are close to 0. The CDF is floored at $\epsilon = 10^{-10}$ to avoid having outliers dominate the likelihood calculation. Each distribution is initialized with one pseudovalue corresponding to 1 megabyte.

IP address and ASN: A user's ISP may assign a different IP address to a user on a regular basis; however, users logging in from home are likely to be coming from the same Autonomous System, a subnetwork that can be determined by looking up the IP address in a database. We used the publicly available MaxMind[1] database to determine which Autonomous Systems own which blocks of IP addresses.

We model IP address as conditionally dependent on ASN in the following manner. For each user U, we first calculate $P(ASN|U)$ using the IP addresses in the user's training data and a lookup table; "unknown" is treated as a special ASN.

We also reserve some probability mass α for unseen ASNs that increases with the number of ASNs observed from a user, and maintain separate estimates for the probability that a new ASN is seen depending on whether it is in the same geographic region – in the United States, the same state – as the enterprise location. (This is obtainable from the "region" field of the Maxmind database.) $P(ASN = \text{"new in-state"}) = \alpha = \gamma|A| + \epsilon$, with $|A|$ the number of unique ASNs seen from within the same geographic state and smoothing parameter $\gamma = 0.01$, and $P(ASN = \text{"new out-of-state"}) = \beta = \gamma|B| + \epsilon$, where $|B|$ is the number of unique out-of-state or unidentifiable ASNs seen for this user. (The epsilons create a very small but nonzero likelihood if the user has never been out-of-state, or more rarely, never in-state.) Then

$$P(ASN = a_j|User) = \begin{cases} \frac{|ASN=a_j|}{\alpha+\beta+\sum_i |ASN=a_i|} \text{if } a_j \in A, \\ \frac{\alpha}{\alpha+\beta+\sum_i |ASN=a_i|} \text{if new local ASN,} \\ \frac{\beta}{\alpha+\beta+\sum_i |ASN=a_i|} \text{otherwise.} \end{cases} \tag{1}$$

[1]http://www.maxmind.com/

Given an ASN, the frequency with which an ASN assigns a particular IP address can be used to calculate $P(IP|ASN, U)$ for each IP address in the training data. Like ASN, a small probability mass within each ASN is reserved for unseen IP addresses from the same ASN, and this probability mass $\beta = \gamma|I_a|$ is proportional to the total number of unique IP addresses I_a that the ASN a has assigned to the user. Let $a(s)$ be the ASN produced on IP address s on lookup, while A remains the set of unique ASNs.

$$P(IP = s|ASN = a) = \begin{cases} \frac{|IP=s|}{|ASN=a(s)|+\gamma|a(s)|} \text{if } s \in I_a, \\ \frac{\gamma|a(s)|}{|ASN=a(s)|} \text{if } s \notin I_a. \end{cases} \tag{2}$$

The total contribution from these features to a Naive Bayes calculation of likelihood is then

$$P(IP = s) = P(IP|ASN = a(s))P(ASN = a(s)) \tag{3}$$

with conditioning on the user implicit.

Naive Bayes summary

The probability of a session being produced by a particular user U is proportional to the product of the probabilities described above:

$$P(U) \propto P(A)P(I|A)P(D)P(H)P(O)P(S) \tag{4}$$

where A is the ASN, I is the IP address, D is the day of the week, H is the hour of day, O is boolean overlap feature, and S is the size of the data, and the features are conditioned in the usual manner on the identity of the user.

Expectation Maximization

Given the framework established above, we could perform direct comparisons on sessions to determine which user is more likely to have generated them – but our target use case is identifying when a *novel* user has injected sessions into a benign user's activity pattern. The following method uses Expectation Maximization to infer the probability that each session is compromised from the session features. The procedure alternates between estimating user and attacker parameters, and re-estimating the likelihood that each session is compromised. The resulting two-user model is then compared in likelihood to the single-user model (Figure 1).

We wish to decide whether a sequence of session activity contains injected activity from a different user from the account's intended user. Let $P(B_t)$ denote the probability that session t in the data is "bad" – that is, it belongs to a different user from the one we trained on for this account. We initialize $P(B_t) = 0.1$ for all sessions in the unlabeled test data, while $P(B_t) = 0$ implicitly for all sessions in the training data.

Now we re-estimate the original user's model and also create an attacker model, using $P(G_t) = 1 - P(B_t)$ as a weight on the new data for the "good" user and $P(B_t)$ as the weight for the attacker model. In the Naive Bayes calculations, all event counts become weighted sums, where the

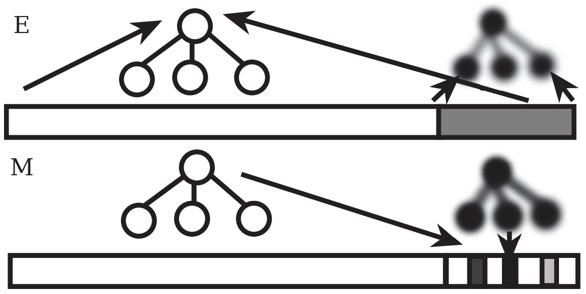

Figure 1: EM bootstraps and "clarifies" the attacker model at runtime by alternating between the E and M steps. *E step.* A user and attacker model are inferred from the user's training data (left) and sessions of uncertain trustworthiness (right), producing a revised user model and a new attacker model. *M step.* Probability of compromise is revised for each session based on the new models.

weights are the probabilities of session trustworthiness. We use the notation $P_B(X)$ as shorthand for $P(X|Attacker)$, with X an arbitrary variable. The weighted equations are:

$$P_B(Hour = h) = \frac{1 + \sum_{t:Hour=h} B_t}{24 + \sum_t B_t} \tag{5}$$

$$P_B(ASN = a) = \frac{\sum_{t:ASN=a} B_t}{\alpha + \beta + \sum_t B_t} \tag{6}$$

The equations are similar for IP address, day, bytes read, and overlaps: observations are weighted by the likelihood they belong to the attacker, and the smoothing parameters remain calculated in the same manner as the unweighted case. The counts of unique IP addresses per ASN $|I_a|$ and ASNs $|A|$ for the attacker are over the sessions such that $P(B_t) > \epsilon$. The user's model is also updated in the same way, treating the weights on the training data as 1 and the weights on the novel test data as $1 - P(B_t)$.

With a new attacker and defender model, the likelihood of each novel session belonging to one or the other classification can be recalculated. The session likelihood is calculated for the attacker model and the defender model for each session, using the Naive Bayes features and a prior $P(B)$ estimated from the test data:

$$P(B) = \frac{\sum_t B_t}{T} \tag{7}$$

Here, T is the number of sessions in the test sequence. The probability of a given session being compromised is then:

$$P(B_t) \propto P(B)\ell_B(t) \tag{8}$$

where $\ell_B(t)$ is the likelihood of the data according to the attacker's inferred Naive Bayes model. A similar calculation is made for the original user to calculate likelihood $\ell_G(t)$, and the two probabilities are scaled to sum to 1:

$$P(B_t) = \frac{P(B)\ell_B(t)}{P(B)\ell_B(t) + (1 - P(B))\ell_G(t)} \tag{9}$$

This process is repeated for each session in the test sequence.

This process can now be repeated: the new likelihoods are used to reweight the data, and the models are again retrained. The retrained models are used to estimate the likelihood that each session is compromised, and so on. Each EM pass generally increases the log likelihood of the test sequence until convergence.

The likelihood of each session under this two user model is the weighted sum of its likelihood given the session is benign, and its likelihood given the session is compromised:

$$\ell_t^{comp} = B_t \ell_B(t) + (1 - B_t)\ell_G(t) \tag{10}$$

The log likelihood of the sequence is then the sum of the individual log likelihoods:

$$\log \ell_{1 \ldots T}^{comp} = \sum_t \log \ell_t^{comp} \tag{11}$$

This likelihood calculation serves two purposes. First, if the log likelihood compared to the last EM iteration is either worse or within a small threshold (we used 1.0) of the last log likelihood, iteration has converged and the EM process stops. Second, this log likelihood can be used as the overall likelihood of the hypothesis that there exists some compromised data within the sequence. This can then be compared to the null hypothesis log likelihood, $\log \ell^{null} = \sum_t \log \ell_t^{null}$, which is simply the log likelihood under the assumption that only the benign user is producing the data, with an alarm raised if the log likelihood is greater by more than a threshold.

Key to the success of the method is the fact that the components of ℓ^{comp} that have a very high probability of being benign are also terms in ℓ^{null}. If EM has assigned probabilities $P_G(t)$ close to 1 to the clearly benign sessions, those sessions will drop out of the equation, leaving the terms corresponding to the sessions with a non-negligible chance of being compromised. As mentioned in the introduction, this prevents a situation in which large amounts of benign user activity could conceal a few malicious logins, which would occur for most Markov methods. If (10) were instead simply $\ell_t^{comp} = \ell_B(t)$, a compromised sequence with many benign sessions would overall produce a lower log likelihood for the benign model than the attacker model.

Though the idea of iterating until convergence as a *test* procedure may seem alarming, in practice, this usually requires no more than a few passes and a few milliseconds on commodity hardware.

Complexity

Each EM pass, including initial training, requires only counting, hashing, and database lookups for each session, and the number of passes required can be bounded by a low constant without loss of performance (the expectation is roughly 5). Running the analysis on the full user base is therefore $O(US)$, where U is the number of users and S is the maximum number of sessions. We point out that the problem is itself $\Omega(US)$ if the administrator wishes to check each user and each session for evidence of compromise, so our algorithm is tight at $\Theta(US)$.

The memory required for the algorithm is also $O(US)$, which is not tight, and there is probably room for improvement in the information sharing between user models. However, space was not particularly an issue in practice – the Java implementation ran at default memory settings, with the user model IP/ASN hash tables requiring only 20-30 entries in the usual case.

Rules

Besides the Naive Bayes approach, we evaluate the contribution of two rules that an operator might naturally impose to solve the problem. If the test data contains only known IPs for a user in the sequence, it is considered benign (the "whitelist" rule); and if the data contains an overlap, it is assumed that a compromise exists in the sequence (the "overlap" rule). Note that the second rule does not make the contribution of the overlap feature irrelevant, because the absence of overlap can provide some evidence that a session or sequence of sessions is good.

These rules naturally exclude certain compromise cases from analysis altogether, but not unreasonably. The whitelist prohibits the algorithm from detecting logins from a trusted machine that is compromised – but if a user's machine is compromised, even a normal login will expose the network, and intrusion detection is beside the point. The "no overlaps" rule requires enforcing a policy from normal users that no more than one machine can VPN in at a time, but this seems a minor annoyance in exchange for a heuristic that can weed out many compromise cases.

Experiment: Masquerade

Methodology

The following experiment used a corpus of VPN log messages for all 2197 VPN users of an enterprise environment from June 14, 2011 to May 3, 2012. The experiment was run on a single processor of an Intel Xeon X5675 CPU (3.07GHz). The logs contained data from 20 different time zones, reflecting the diversity of remote locations from which users could log in; these were a mix of employees permanently working remotely and employees traveling. A sliding window of 261 training days followed by 7 test days was run for each of 64 runs of the experiment.

We used Hadoop[2] to parallelize the regular-expression-based parsing that turned hundreds of thousands of VPN event messages into Accumulo[3] tables of event descriptions, indexed by user id. Once the data had been ingested in this way, it was small enough to run on a single machine.

For each run, the first 100 users in the database with at least five logins during the test week were selected to emulate "intruders" in the other users' data. All users with at least one login during the training period and at least one login during the test week were used as benign users. 100 mixed logs were created for each benign user by taking the union of the events recorded for the original user and the "intruder." The mixed logs did not necessarily contain any

[2]`http://hadoop.apache.org/`

[3]`http://accumulo.apache.org/`

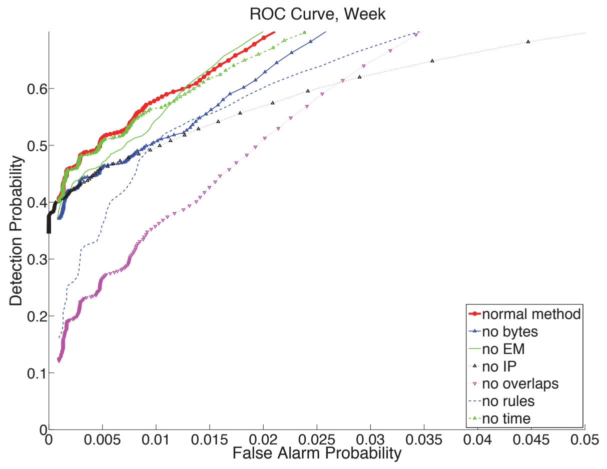

Figure 2: Average true and false alarm rates for the 0-5% false alarm range when detecting injected user activity over a week, compared to performance of the method with various features or steps omitted.

sessions that directly overlapped; in fact, this was relatively uncommon, since most logs were sparse. The EM procedure described above was then performed on each artificial test log, using the trained model for the original user, to calculate an overall likelihood of intrusion for the test period. The system was blind to the injected user's training data during each test, and was only allowed to bootstrap a tentative model for the injected user from the mixed test period data. The number of successful conclusions of compromise determined the true positive rate.

To determine the relative importance of the features, the same experiment was repeated with each feature in turn removed from the calculations. We also experimented with using a flat attacker model and no EM, with uniform probability of login time and day, a byte distribution mirroring the user, an extremely low (10^{-11}) probability of matching the user's IP, a probability matching the user probability for the same ASN and a new IP, and the rest of the IP/ASN probability mass divided evenly between the "out-of-vocabulary" symbols.

Results

A ROC curve for performance is presented in Figure 2 (note the labels on the axes, as we concentrate on the practical region of the curve). The alarm rates are not on a per-session basis, but per-user, with an alarm reported if the algorithm reports any intrusion within the target time period; and they are per-week. Thus, a 1% false alarm rate means that in a week of operation, it is expected that 1 in 100 benign and active users will trigger an alarm. With roughly 1000 users active in any given week, this corresponds to 10 users falsely flagged in any given week.

When the prior is not weighted toward believing users are benign at all (log likelihood threshold of 0), the algorithm achieves 94% true positive rate for a 12% false positive rate (not shown). For a more practical 1% false positive rate, the method achieves at most a 59% true alarm rate. For a very conservative false positive rate of 0.1%, the algorithm still achieves a 44% true alarm rate. The lowest false alarm rate we tested was 40.5% true alarm rate for a 0.095% false alarm rate. (The true alarm rate here is the same as the recall, whereas the precision is one minus the false alarm rate; i.e., 44% recall is achieved for a 99.9% precision.)

The most useful feature appeared to be the detection of overlapping activity between users. As a followup to determine whether overlaps were doing the bulk of the work in detection, we measured how many overlaps would be detected in a sample day (March 1), and found that 16% of the alarms could be sounded through overlap alone. Thus, while it is the most useful feature, it is not solely responsible for our true alarms.

The two rules – whitelist known IPs, and alarm on overlaps – appear to provide a useful function above and beyond their use as Naive Bayes features, especially for achieving extremely low false alarm rates. These rules can be seen as asserting that some events have zero probability, an assertion that usually does not work well in log likelihood methods because one can't take the log of 0. Removing the whitelist and overlap rules, we achieve only 50% true alarms at 1% false positives, and only 16% at 0.1% false positives. Thus, removing our two assumptions (no users allowed to have multiple sessions, no compromise of user machines instead of credentials) has only slight impact when there are some false positives allowed, but extremely high precision requires these assumptions for good performance.

The EM method appears most useful for achieving very low false positive rates, and in that range is nearly as useful as IP address. It appears to become less useful as the operating point is more forgiving of false positives, at which point a flat attacker model works as well or better. This suggests the importance of testing intrusion detection methods on the sub-1% false positive operating range, since this strictness can change the evaluation of whether a technique is useful.

Bytes read and IP appear roughly equally useful in the sub-1% false positive operating range, while time of day and day of the week appear to be relatively unhelpful.

Discussion

Masquerade detection is in general a hard problem; though we know of no prior VPN masquerade work to which we can compare our results, when masquerade detection is performed on command line traces, most reported results fall below 70% true alarm rates (Salem and Stolfo 2011). Benign and "malicious" activity simply may not look very different, because in fact there is nothing inherently suspicious about the injected activity except that it may not look like the user's normal pattern. We believe that the results we present here are reasonably strong given the demographics of the user base, who for the most part log in from very similar locations at very similar hours, and yet occasionally will log in from some never-seen-before location as the result of travel. Examination of a sample day revealed 18% of the 661 users with any activity for that day logged in from an IP address that they had not used during the previous 261 days (our training window size), and 11% of the 661 users logged in from a novel city. Perfect performance is not to be expected under such conditions.

The number of users logging in from novel IP addresses and novel ASNs on an arbitrary day in the VPN traffic was surprising to us, and made achieving sub-1% false alarm rates very challenging. For extremely low false alarm rates, using IP addresses as information can apparently cause more harm than good. We also found that the extreme variability of the natural activity levels of users meant there were many cases in which a very infrequent user was injected into the trace of a very active user, prompting us to develop the EM method for extracting these needles in the haystack.

One possible attack on this method would have the attacker slowly shifting the good user's model away from its true values – a "frog-boiling attack" (Chan-Tin et al. 2011). The overlap detection is our principal defense in this model against this kind of attack; a malicious user attempting to look very much like the real user should eventually be caught by virtue of logging in at the same time.

While tackling this problem, it became clear that training a mixture of users as an adversary model leaks an unrealistic amount of information about the attacker. Our original results on a 30 user sample with that approach produced 98% true positives and no false positives. As we scaled, performance dropped, and it became clear that the detection was benefitting from having already seen the attackers. Had we not performed the experiment at scale, we would not have caught this methodological error, and it calls into question any other masquerade work that uses mixtures of users as the attacker model. We re-emphasize that our attacker model is now learned on the fly during testing.

We have discovered that achieving very low false positive rates on detecting account masquerade is possible, but user variability makes this an extremely challenging problem when attempting to scale; problems can appear in the sub-1% false alarm rate that are not at all apparent from small sample sizes, and seemingly reasonable methodology can prove flawed when scaling to realistic user bases. Focusing on any one feature to characterize a user seems unlikely to succeed. Nevertheless, the problem of detecting likely compromised sessions and accounts remains an important challenge for network security, and our work suggests that a human operator attempting to judge whether sessions are anomalous from source location and time alone is likely to have a still higher false alarm rate.

Acknowledgments

The authors thank Charles Wright, Bill Streilein, Gabe Wachman, and Josh Haines for their valuable comments and suggestions, and Kevin Borders and Michael Oehler for graciously sharing their unpublished manuscript. This work is sponsored by the Department of Defense under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

References

Bolton, R. J., and Hand, D. J. 2002. Statistical fraud detection: A review. *Statistical Science* 17(3).

Borders, K., and Oehler, M. 2012. Detecting hijacked credentials for internet services. Unpublished manuscript.

Chan-Tin, E.; Heorhiadi, V.; Hopper, N.; and Kim, Y. 2011. The frog-boiling attack: Limitations of secure network coordinate systems. *ACM Transactions on Information and System Security (TISSEC)* 14(3).

Coviello, A. 2012. Open letter to RSA customers. http://www.rsa.com/node.aspx?id=3872, retrieved 4/18/2012.

Dempster, A. P.; Laird, N. M.; and Rubin, D. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B* 39(1):1–38.

Denning, D. E. 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering* 13(2).

Manning, C., and Schütze, H. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Maxion, R., and Townsend, T. 2002. Masquerade detection using truncated command lines. In *International Conference on Dependable Systems and Networks*.

Maxion, R. 2003. Masquerade detection using enriched command lines. In *International Conference on Dependable Systems and Networks*. IEEE.

Min, L., and Shun-Zheng, Y. 2006. A network-wide traffic anomaly detection method based on HSMM. In *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, volume 3, 1636 –1640.

Salem, M. B., and Stolfo, S. J. 2011. Modeling user search behavior for masquerade detection. In *Recent Advances in Intrusion Detection*, volume 6961 of *Lecture Notes on Computer Science*. Springer.

Schonlau, M.; Dumouchel, W.; Ju, W.; Karr, A. F.; Theus, M.; and Vardi, Y. 2001. Computer intrusion: detecting masquerades. *Statistical science* 16:58–74.

Sommer, R., and Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*. IEEE.

Xie, Y., and Yu, S.-Z. 2006. A dynamic anomaly detection model for web user behavior based on HsMM. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, 1 –6.

Yeung, D.-Y., and Ding, Y. 2003. Host-based intrusion detection using dynamic and static behavioral models. *Pattern recognition* 36(1).