A Novel AI-based Methodology for Identifying Cyber Attacks in Honey Pots

Muhammed AbuOdeh[1], **Christian Adkins**[1], **Omid Setayeshfar**[2], **Prashant Doshi**[1], **Kyu H. Lee**[2]
[1] THINC Lab,
[2] Institute for Cyber Security and Privacy
Department of Computer Science, University of Georgia, Athens GA 30606
pdoshi@uga.edu

Abstract

We present a novel AI-based methodology that identifies phases of a host-level cyber attack simply from system call logs. System calls emanating from cyber attacks on hosts such as honey pots are often recorded in audit logs. Our methodology first involves efficiently loading, caching, processing, and querying system events contained in audit logs in support of computer forensics. Output of queries remains at the system call level and is difficult to process. The next step is to infer a sequence of abstracted actions, which we colloquially call a *storyline*, from the system calls given as observations to a latent-state probabilistic model. These storylines are then accurately identified with class labels using a learned classifier. We qualitatively and quantitatively evaluate methods and models for each step of the methodology using 114 different attack phases collected by logging the attacks of a red team on a server, on some likely benign sequences containing regular user activities, and on traces from a recent DARPA project. The resulting end-to-end system, which we call Cyberian, identifies the attack phases with a high level of accuracy illustrating the benefit that this machine learning-based methodology brings to security forensics.

Introduction

Automatically identifying the phases of a cyber attack on a host is of significant import. It facilitates automated forensics, which leads to faster attack discovery, damage assessment, and ultimately prevention. It also represents a key step toward better understanding the intent of the attacker. This paves the way for the defender to engage the attacker in more effective cyber-deception techniques on honey pots. However, this capability is made difficult by the fact that the individual attack steps are often common between different types of attacks, and the attack activity, if logged, is buried within massive system call logs that are hard to sift through.

A step toward addressing this challenge is development of host-based intrusion and anomaly detection systems (Chevalier 2019), which alert defenders about anomalies in system behavior that may indicate malicious activity. For example, a recent technique named DeepLog (Du et al. 2017) demonstrated feasibility of using deep learning to detect anomalous behavior based on low-level log data. However, DeepLog and similar systems generally lack the capability to identify attack phases, which is usually left to a human security analyst.

We present a novel machine learning (ML) based methodology that automatically identifies phases of cyber attacks on a single host system such as a honey pot. Our methodology involves the following general steps: (i) Efficiently load, cache, process, query, and display system events from audit logs to support computer forensics. Output of queries should be provenance graphs, which can be processed for further analysis. (ii) Resulting trace is often still hard to parse and in need of further abstraction to facilitate analyses. Utilize a latent-state probabilistic model, which allows us to infer the most likely sequence of higher-level actions, which we call an *attack storyline*, while modeling system calls as observations. (iii) Finally, we seek to identify the attack phase based on the sequence of high-level actions inferred in the previous step. We view this step as a multi-class classification problem where each label is a phase of an attack.

Classifying cyber attacks has been explored before. In early work, Lippmann et al. (2000) analyzed intrusion detection performances of various rule-based systems on network log data containing both benign and malicious activity, which included labeling segments corresponding to low-level attack types. The results highlight the limitations facing rule-based detection toward attack identification. Bolzoni, Etalle, and Hartel (2009) paired anomaly detection with machine learning to classify attack types based on n-gram analysis of attack payloads. These and other similar works classifying types of attacks tend to be rule based and ignore the significant overlap between various attacks, which often share steps. Many face the well-documented rule-based limitation of being inflexible to changes in both benign and malicious behaviors in the dynamic realm of cyber security. Others focus on classifying an entire attack campaign (instead of intermediate attack phases) where accurate identification of later-occurring phases relies on successful identification of earlier phases.

Another system named HOLMES (Milajerdi et al. 2019) focuses on detecting anomalies as phases of advanced and persistent threats. HOLMES also uses a rule-based system for classifying attack phases and relies on a large amount of benign log training data to reduce false positives in the testing data. In contrast to the rigidity of such rule-based systems, our ML-based methodology can learn and adapt to identify several attack phases as long as some system call logs for these phases exist to use during training. This is enabled by a key innovation of the methodology: the inference of attack storylines as an intermediate step.

To translate our methodology to practice, we qualitatively or quantitatively evaluate each step with candidate methods or models. An *independent* red team simulated several types of attacks on a server acting as a honey pot. This yielded a total of 114 attack phases, which included asset discovery and data exfiltration, among others. The pipeline of selected models lead to a system, which we call Cyberian, that performs very well in identifying the attack phases as measured by an F1-score of 90.31 ± 8.44. Additionally, we test on a few attack sequences discovered in a recent data set released by DARPA. An out-of-class evaluation of the learned models to test how well Cyberian can distinguish between malicious sequences and those that are likely benign is also performed.

Attack Phase Abstraction and Classification

Our methodology is useful to areas of cybersecurity called *cyber forensics* or *cyber crime investigation*. Tools for forensics predominantly rely on log records to understand behavior of attackers, whether at application level (web server logs) or at system level (system call logs). A key challenge for analyzing logs is their enormity – a typical computer produces more than 10K low-level logged events each minute. While higher level logging may produce more understandable events, these sacrifice detail and introduce heterogeneity due to diverse events emitted by different programs in the system. On the other hand, lower level system logs provide detail and homogeneity, but extracting meaningful abstractions from them has been difficult. Our initial focus is to perform the forensics on *honey pots*, which are hosts masquerading as important systems in a subnet intended to deceive attackers by consuming their attack resources. As these hosts are not in regular use, there is less benign activity and the log entries largely capture unauthorized intrusions and subsequent activity.

Provenance of System Calls

The first objective is to extract an attack-focused sequence of system calls from low-level system call logs and generate the provenance graph. Each sequence represents an attack phase, such as data exfiltration or privilege escalation. Generating the provenance graphs in this step is crucial to the rest of the methodology. We identify several desiderata that contribute toward ways for successfully generating these graphs:

- Schema flexibility by allowing varied log formats
- Online pattern matching or live monitoring and tracking on streaming log data
- Backward and forward tracing on provenance – especially on long sequences along with a graphical representation
- Support for a granularity smaller than the process
- An intuitive query language and interface that goes beyond simple keyword-based filtering to isolate key events in logs
- Automated anomaly detection capabilities
- Open source version available to facilitate wider adoption

We utilize these desiderata to qualitatively compare several more prominent log analysis systems that are currently available. These include systems such as Carbon Black's LiveOps that are proprietary and commercially available only as well as Plaso and Elastic (Project 2019), AIQL (Gao et al. 2018b), SAQL (Gao et al. 2018a), and SOF ELK (Lewes 2019). Table 1 shows this comparison in a concise format; more details about each of the systems is available in its reference.

Title	S	PG	F	O	G	Q	A
AIQL	×	□	×	×	×	■	×
Carbon Black LiveOps	■	□	×	×	×	□	■
GrAALF	■	■	■	■	■	■	×
NoDoze	×	□	×	×	×	×	■
Plaso + Elastic	×	×	□	□	×	□	■
SAQL	■	□	×	×	×	■	×
SOF ELK	×	×	□	□	×	□	■

Table 1: We denote **S** for streaming analysis, **PG** for provenance graph forward and backward tracing, **F** for flexibility in schema, **O** for open-source, **G** for supporting granularity smaller than process, **Q** for querying ability of the graph, and **A** for intelligent anomaly detection. × shows lack of support, □ shows limited support, and ■ shows full support.

GrAALF (Setayeshfar et al. 2019), a new general-purpose query system, offers good support on several desiderata but lacks advanced anomaly detection which makes some human involvement necessary. Regardless, it may serve well to instantiate this step of the methodology using GrAALF.

Inferring Attack Storylines

System call logs, even extracted by GrAALF, tend to be extensive and incomprehensible. For example, some large attack phase files contain 10,154 records occupying up to 2.2MB. This makes manually parsing and understanding attack steps tedious. Resulting traces are often still hard to parse and need further abstraction to facilitate analysis. *Logged system calls may be perceived as a sequence of observable signals noisily emitted by a dynamic system as an adversary performs an attack.* Thus, we may infer the most likely sequence of higher-level actions given these observations. This suggests modeling the probabilistic action recognition problem using classical latent-state graphical models. The sequence of abstracted actions is the most-likely explanation (MLE) inferred by the model for the observed sequence of system calls.

We note that the same system calls often appear in different attack sequencesand perform similar functions. For example, the sequence involving *sh* executing a temporary process which then executes another shell is present in attack phases such as system reconnaissance and persistence. This sequence is indicative of the attacker's shell launching a temporary process which then starts another shell. Furthermore, we may not distinguish between various temporary processes for the purpose of generally understanding and identifying attack phases. Consequently, a relatively small number of distinct abstracted actions suffice to comprehensibly explain various attack phases at a higher level. These abstracted actions form hidden states of latent models, and a categorized listing of some candidate states is given in Table 2. For a given (cleaned) sequence of system calls, we refer to the sequence of states inferred by the model as its *storyline*.

Start states	Bash_execute_process Execute_process (by non bash process) Init_server_daemon
System operation states	System_operation Generic_process_read_write Generic_process_operation Netapp_operation Bash_operation Server_daemon_operation File_operation
Information states	System_information Library_state
Network states	Process_upload_download Netapp_upload_download Daemon_upload_download Server_download Server_upload

Table 2: Categorized list of candidate states for latent-state probabilistic model representing abstracted actions. Column four of Table 4 relates some states to system call events.

Probabilistic graphical models such as hidden Markov models (HMM) (Rabiner 1989) and conditional random fields (CRF) (Lafferty, McCallum, and Pereira 2001) can infer abstracted actions corresponding to system calls as these models allow a sequence to be tagged with labels based on context. While HMMs and linear-chain CRFs share many similarities, HMMs tend to be simpler to learn from data and do not require handcrafted feature functions characteristic of CRFs.

Prior use of probabilistic models such as HMMs in cybersecurity has mainly addressed intrusion detection (Liu et al. 2018). For example, Garcia et al. (2012) use an HMM and k-means for detecting malicious activity, while Wang, Guan, and Zhang (2004) use an HMM to determine whether a sequence exceeds a certain threshold, thus classifying it as an attack. In Radhakrishna, Kumar, and Janaki (2016), the aim is to identify intrusions using temporal pattern mining. Intrusion detection usually does not involve analyzing specific, actionable observations and identifying the attack phases.

Identifying Attack Phases

Each storyline details the logical steps comprising an attack phase. We may view storylines as time series' and the problem of identifying the corresponding attack phase as multiclass classification. Subsequently, storylines serve as input for a machine learning classification model. Storylines are composed of sequential contexts with temporally extended dependencies. Such dependence is likely a factor within such complex, lengthydata as attack logs. Therefore, our primary model hypothesis is a Long Short-Term Memory (LSTM) neural network (Hochreiter and Schmidhuber 1997).

As alternative candidate classification models, a 1-dimensional convolutional neural network (CNN) and a simpler linear support vector machine (SVM) may be explored. Sequential contexts like storylines often contain very similar repeated sub-sequences common among instances within each class. 1D convolution is a common method in time-series data analysis and has frequently been used in various previous works related to classification (Lee, Yoon, and Cho 2017; Zhang, Zhao, and LeCun 2015), and serves as one performance baseline for the classification stage. Joachims (1998) discusses suitability of SVMs for text classification, for which they have been used extensively. SVMs, being simpler models, benefit from fewer parameters thereby requiring less manual design. Indeed, the only parameters that usually require testing are kernel (linear vs non-linear, with some variations) and maximum number of iterations for training.

A classification model trained to perform multi-class classification on each attack storyline represents the last step of the methodology. The resulting output from this step is the identified attack phase. This model's performance ultimately demonstrates the instantiated method's ability to extract high-level information from the granular and extensive logs.

Attack phase	Count
system reconnaissance	39
persistence	12
privilege escalation	14
asset discovery	16
data exfiltration	14
network discovery	19

Table 3: Distribution of various attack phases in our data set.

Performance Evaluation

To realistically evaluate our methodology, an independent red team of cyber security researchers assisted by Metasploit (Rapid7 2020), a well-known penetration testing tool, engaged in several attacks on a Linux server over a few days. These attacks yielded 114 phases, each of which is one of *six* popular types (based on sequence of system calls executed by attacker in each phase). Attack phase types, informed in part by MITRE's ATT&CK matrix (Strom et al. 2018), are:

1. **system reconnaissance**: gather information about system, including OS version and user account information;
2. **persistence**: attacker implements measures to maintain access even after a system restart;
3. **privilege escalation**: attacker attempts to gain root or higher access on the victim machine;
4. **asset discovery**: search for assets such as sensitive files on a system;
5. **data exfiltration**: attacker transfers data out;
6. **network discovery**: attacker explores different connections made to/from the victim machine.

The distribution of these types is shown in Table 3.

Each attack is logged using Sysdig on the honey pot server. GrAALF automates online monitoring by pre-defining a set of query templates similar to predefined rules. Once the user specifies the system and sensitive files to be monitored, a set of queries derived from templates are deployed by GrAALF. For instance, for monitoring a modification of a sensitive file, the following query is used by GrAALF *"back select write from file where name is X"* where 'X' represents file name of interest. To monitor for potential access to IP ranges outside the local network *"back select * from soc where not*

System calls generated by GrAALF	from_name, evt_type, and to_name extracted from system calls	Processed system calls, in the form *from_name* → *evt_type* → *to_name*	Abstracted actions (HMM states)
{ ”sequence_number”: 12125, ”user”: ”www-data”, ”from_id”: 2398 , ”from_name”: ”sh”, ”evt_type”: ”exec”, ”to_name”: ”perl” , ”to_id”:2399 , ”count”: 1}	*from_name*:sh *evt_type*: exec *to_name*: perl	bash → exec → perl	Bash_execute_process
{ ”sequence_number”: 12530, ”user”: ”www-data”, ”from_id”: 2399 , ”from_name”: ”perl”, ”evt_type”: ”exec”, ”to_name”: ”perl” , ”to_id”:2400 , ”count”: 1}	*from_name*: perl *evt_type*: exec *to_name*: perl	perl → exec → perl	Execute_process
{ ”sequence_number”: 12577, ”user”: ”www-data”, ”from_id”: 1782 , ”from_name”: ”apache2”, ”evt_type”: ”shutdown”, ”to_name”: ”10.0.2.8:44933 →10.0.2.10:80” , ”to_id”:13 , ”count”: 2}	*from_name*: apache2 *evt_type*: shutdown *to_name*: ”10.0.2.8:44933 → 10.0.2.10:80”	apache2 → close → remote_address	Server_daemon_operation
{ ”sequence_number”: 12578, ”user”: ”www-data”, ”from_id”: 1782 , ”from_name”: ”apache2”, ”evt_type”: ”read”, ”to_name”: ”10.0.2.8:44933→10.0.2.10:80” , ”to_id”:13 , ”count”: 1}	*from_name*:apache2 *evt_type*: read *to_name*: 10.0.2.8:44933 → 10.0.2.10:80	apache2 → read → remote_address	Server_download

Table 4: Raw system calls extracted from log files and processed. Column 1 shows system calls recorded by auditing software and output by GrAALF. Column 2 gives information extracted from system calls, and the third shows format of sequences as passed on to HMM. A latent-state model may view sequences as being emitted from corresponding states shown in fourth column.

name has 172.16.” where a range of local IP is ‘172.16.*.*’. GrAALF provides further query templates for process-based monitoring. For example, ‘nc’ and ‘scp’ processes are frequently used by adversaries to plan a backdoor or to exfiltrate sensitive data. Query *“back select * from * where name is nc or name is scp”* can monitor detailed behavior of such processes, and their remote hosts can be identified by — *“forward select * form * where name is nc or name is scp”*. Output of GrAALF’s templated queries is a focused provenance graph. The engagement and GrAALF’s analysis yielded 25 sequences of system calls which did not belong to any of the six phases and may be viewed as benign. We utilize these for a deeper evaluation of learned models. All attack data including log files and GrAALF’s output is available for download at https://tinyurl.com/yy9stomv.

Data cleaning On receiving system call sequences from GrAALF, individual system calls, such as those shown in the first column in Table 4, are cleaned to be handled by the latent-state model. A simple pre-processing step removes some fields from each log entry as these fields do not contain information essential to understanding behavior. In particular, Linux audit logging includes fields such as *sequence number*, *user name*, and various *IDs* in addition to *from_name*, *evt_type*, and *to_name*. Values in the first grouptypically do not speak about the action that was performed, and are dropped from further analysis. Remaining fields that contain valuable information for the HMM are *from_name*, *evt_type*, and *to_name* (see second column of Table 4 for illustration).

We adopt straightforward rules to trim the set of distinct observations. For instance, diverse shell processes such as *sh*, *bash*, or *zsh*, are merged into *bash* as it is the most popular shell. In some sequences, we observe processes that create temporary child processes with randomly generated process names; we rename them to *temp_process*. Table 4 shows a real audit log and processed logs.

Additionally, we filter out obviously irrelevant and unhelpful log events. For example, if the underlying audit system fails to extract system or process information, this appears as ⟨NA⟩, and we filter these out as they do not contain useful information. We also exclude routine cache and library file access events (i.e., accesses of .lib and .so files), and unnamed pipe accesses (i.e., “NULL” and “pipe”).

To preserve readability, we provide the observation in the triple format *from_name* → *evt_type* → *to_name*, where the *from_name* in the triple is the parent process, the *evt_type* is what the parent process performs, and *to_name* is the child process on which the event is performed. For example, *bash* → *exec* → *nc* means that bash executes the process netcat. This is the third and final step in converting system call sequences into observations. The third column in Table 4 shows the observation for each system call, respectively.

HMM inference of attack storylines We utilize a standard HMM for inference. Observations to the HMM are system calls in an attack phase sequence, cleaned and in triple format as described above To infer attack storylines automatically, our aim is to learn transition and emission probability tables of the HMM from sequences of system calls in attacks annotated with abstracted actions using a learning algorithm such as Baum-Welch (Baum et al. 1970). This requires relating each state to observed system call triples emitted by the state. For example, the state Daemon_upload_download emits the system call *nmbd* → *sendto* → *some_socket* and state Server_daemon_operation emits the system call *proftpd* → *close* → *some_socket*.

A storyline is then the most likely explanation inferred by the trained HMM for a given sequence of system call observations. An example short attack storyline inferred for the asset discovery phase is: (Execute process *ircd*, Server daemon operation *ircd*, Execute process *perl*, Execute process *perl*, System information, Execute process *perl*, Execute process *temp_process*, Execute process *temp_process*, System information). The lengths of

Fold	Mean LL	Mean LL ratio
0	-4.909 ± 0.996	0.938 ± 0.084
1	-4.406 ± 1.566	0.956 ± 0.05
2	-4.586 ± 1.338	0.942 ± 0.071
3	-5.091 ± 0.974	0.95 ± 0.067
4	-4.579 ± 1.549	0.911 ± 0.097

Table 5: Mean and standard deviation of log likelihood and log likelihood ratio per test fold generated by HMM. Mean log likelihood across all folds is **-4.714**, mean ratio is **0.939**.

storylines vary from 1 step to more than 5,000.

Cleaned sequences of system calls from GrAALF for 114 attack phases yielded 1176 distinct observations, for which we utilized an HMM with 17 states as defined in Table 2. We implemented the HMM using the Pomegranate package (Schreiber 2017). We evaluate the HMM's performance using 5-fold cross validation on the annotated sequences in the 114 + 25 sequences. The HMM is trained using four folds, which involves learning the transition and emission probabilities using Baum-Welch with a count-based initialization (Laan, Pace, and Shatkay 2006) and pseudo counts. The HMM's fit and inference of the storyline is evaluated using the fifth fold.

We report log likelihoods of system call sequences in the five test folds. Each observed attack sequence is passed to the Viterbi algorithm, which yields the sequence of states that most likely explains observations. Likelihood is the probability that this sequence of states emits the observations. Table 5 shows mean log likelihood (divided by number of steps in the phase to account for highly varying lengths) for each fold, and also presence of some sequences of system calls that could not be predicted with high probability leading to noticeable standard deviations. In some test folds, the learned HMM encountered some sequences with previously unseen system calls. However, small non-zero values at initialization of transition and emission probabilities and Baum-Welch's use of pseudocounts permitted generalization to these sequences.

Having evaluated the model's fit, next we evaluate correctness of the inferred storylines. We compare the previous mean log likelihoods with the mean of log probabilities of observed sequences in each test fold given the *ground truth* assignment of states for a sequence. In Table 5, we also report the means and standard deviations of likelihood ratio $\frac{\text{LL of MLE}}{\text{LL of ground truth}}$ while noting that a ratio of 1 is desired. While no fold gave a perfect ratio of 1, the mean ratio for most folds is above 0.9 indicating that generated storylines were mostly correct. Table 6 contains likelihood ratios decomposed by attack phase. Storylines pertaining to network discovery yield lowest likelihood ratio because some system calls commonly seen in several network discovery sequences were consistently mispredicted by the HMM. As network discovery sequences are among the shortest, few errors have a significant impact. Finally, storylines are passed on for identification.

Model	Asset disc.	Sys. recon.	Exfil	N/w disc.	Persist	Priv. Escal.
HMM	0.957	0.973	0.873	**0.846**	1.0	0.986

Table 6: Mean log likelihood ratio of the HMM decomposed by attack phases. The lowest performance is highlighted.

Model	Weighted-mean F1	
	with HMM	without HMM
SVM	84.14 ± 12.82	64.79 ± 15.75
CNN	86.32 ± 11.46	14.51 ± 16.30
LSTM	**90.31 ± 8.44**	47.55 ± 26.03

Table 7: Weighted-mean F1-score (%) and weighted standard deviation for models with and without storylines. Statistics obtained by weighting phases' F1-scores with class sizes.

Classifier evaluation to identify attack phases We design the recurrent neural network model to embed tokenized and padded input sequences (length of 5,465), pass resulting vectors through two LSTM layers (100 memory units, dropout of 0.05, recurrent dropout of 0.1), and finally use a softmax layer to arrive at a probability distribution over attack phase labels for each input sequence. The CNN takes encoded and padded storylines through two convolution blocks (32 and 64 filters respectively, kernel sizes of 3, max pooling of 2, followed by dropout of 0.05), followed by a fully-connected layer, and finally a softmax output layer producing classification probabilities. The support vector classifier uses a simple linear kernel. Various parameters for each model were explored to determine best configurations, such as different dropout rates and training epochs and use of class weights.

Our neural network models were implemented in Keras with TensorFlow backend, while scikit-learn's SVM implementation was used. Experiments were performed on a Linux system with 4 Intel Xeon Skylake processors, 32GB RAM, and 1 NVDIA P100 GPU. To demonstrate utility of HMM's abstraction toward identifying high-level attack characteristics, we evaluate classifiers in two ways: The first experiment trains for up to 100 epochs (maximum iterations of 20K for the SVM) on ground truth sequences and evaluates each model's performance on storylines comprised of states that the HMM produced for each attack phase. A ground truth labeled sequence is the sequence of manually annotated states. In the second, input sequences are system calls directly coming from GrAALF without being processed by the HMM; as there are no storylines for this raw data, 5-fold cross validation was used to evaluate models on this data set (confusion matrices for each fold were combined to get final results). Relative results of these tests show how well HMM predictions reflect true characteristics of each attack phase.

Table 7 gives the weighted-mean F1-score of the classification by each of the models in both experiments. First, notice that the use of storylines as input to the classifiers improves their accuracy significantly compared to using sequences of low-level system calls; this improvement is especially large for the CNN. Among the various classifiers, the LSTM model achieves a better mean F1-score than the SVM or CNN. The LSTM operating on attack storylines is able to accurately identify the type of about **90%** of the attack phases. However, the paired F1-score differences between the LSTM and the other methods are not statistically significant. Clearly,

Model	Asset disc.	Sys. recon.	Exfil	N/w disc.	Persist	Priv. Escal.
SVM	74(43)	90(61)	93(50)	**62(58)**	89(74)	97(36)
CNN	**64(90)**	90(99)	100(98)	91(99)	89(100)	79(98)
LSTM	**73(91)**	92(99)	100(99)	94(99)	86(99)	93(99)

Table 8: F1-score (%) of HMM-aided models by attack phases, and mean confidence on correct classifications (true positives). Lowest performance for each model is highlighted.

Model	Asset disc.	Sys. recon.	Exfil	N/w disc.	Persist	Priv. Escal.
SVM	**68**	90	88	90	80	93
CNN	89	88	100	100	80	**68**
LSTM	79	92	100	100	**75**	93

Table 9: Precision (%) of HMM-aided classification models by attack phases. Lowest performance for each is highlighted.

the process of inferring the most likely explanation of the observed system calls is very valuable and identification of the attack phases benefits from reasoning about the context.

Precision and Recall We analyze models' performances by reporting classification F1-scores by attack phase. Table 8 shows that LSTM's and CNN's weakest performance is on asset discovery sequences, where no model achieves a high score. This significantly lower F1-score is due, in part, to sequences of that class having low average lengths compared to others. Asset discovery sequences give the models less information to learn from and inform the classification.

As recall is the percentage of instances of a class that are correctly classified, in this multi-class setting, the per-class recall corresponds to the classification accuracy for each class. In Tables 9 and 10, we see that no model achieves high recall on the *asset discovery* phase, with the CNN's recall being the highest score for that class (75%). The *exfiltration* phase is evidently the easiest to classify, with perfect recall and at least 82% precision for each model. The LSTM model achieves the worst single score in both precision and recall.

Out-of-Distribution Evaluation While our aim is to make the log analysis step highly precise in identifying attack provenance, we also consider the realistic scenario where it misidentifies possibly benign sequences of system calls as attack phases. As sequences are composed of system calls that are likely to be shared with attack phases, the HMM continues to generate storylines for likely benign sequences.

Model	Asset disc.	Sys. recon.	Exfil	N/w disc.	Persist	Priv. Escal.
SVM	81	90	100	**47**	100	100
CNN	**50**	92	100	84	100	93
LSTM	**69**	92	100	89	100	93

Table 10: Recall (%) of HMM-aided classification models by attack phases. Lowest performance for each is highlighted.

Model	Asset disc.	Sys. recon.	Exfil	N/w disc.	Persist	Priv. Escal.	Benign
SVM	35	84	44	41	100	0	38
CNN	58	92	53	88	89	72	0
LSTM	69	92	53	94	86	93	0

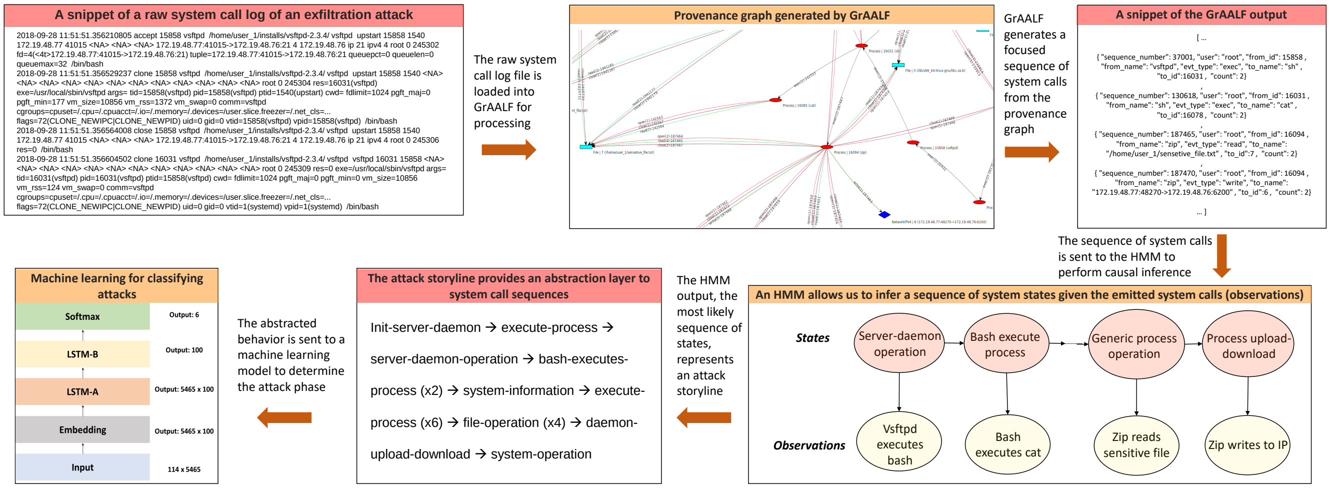
Table 11: F1-score (%) of each HMM-aided model by attack phase and benign class, based on confidence threshold of 0.5.

How do trained classifiers perform on out-of-class instances? 25 likely benign storylines were tested alongside 114 attack phase storylines. As models are trained on instances belonging to six attack phases only, each prediction was initially for one of these classes. We simply rely on prediction confidences to discriminate benign sequences. Ideally, if benign instances are significantly different from attack phase instances, their predictions should have distinctly lower confidences. For these tests, any prediction made with a confidence value less than a threshold T was changed to a prediction of the benign class for the corresponding instance. This change would make the prediction correct if the instance is truly benign and incorrect for any of the original 114 instances.

Table 11 shows results of this experiment with $T = 0.5$. The SVM suffers most from this thresholding approach due to confidence values being relatively lower and more varied than those of the other models. It does achieve the highest F1-score on benign sequences out of the models (at $T = 0.5$ and at other values), but its overall performance on the attack phases is more severely diminished than those of the CNN or LSTM. The CNN and LSTM achieve high enough confidence values on the majority of their predictions so that this simple thresholding method did not change any predictions to benign. Thus, the models achieved an F1-score of 0% on that class.

Furthermore, isolating predictions by each model on the likely benign instances did not reveal significant differences in mean prediction confidence from that of true attack phases. On benign instances, mean confidence is near 51% for SVM, 88% for CNN, and 96% for LSTM. Comparatively, on the 114 attack phases, the SVM has the most varied confidence values with a mean confidence near 54%. The CNN and LSTM reach consistently high confidence values, with means of 92% and 97% respectively. As such, the mean confidences of the latter two models are similar. We did not find a value for T that could correctly classify a high percentage of benign instances without severely reducing the performance on the attack phases. This suggests that a more sophisticated approach is needed for out-of-class instance identification.

Confidence and Additional Phases Assessing the confidences each model achieves during classification can reveal attack phases lacking clear discriminating features. The CNN and LSTM have high average confidence levels among correct and incorrect classifications, with almost all confidences near 99% for the LSTM. The SVM has more dispersion in these scores, and has better distinction in confidences among right and wrong classifications. Interestingly, across all three models' incorrect classifications, if the true class was *privilege escalation* then the predicted class was most likely to be


A snippet of a raw system call log of an exfiltration attack
The raw system call log file is loaded into GrAALF for processing
Provenance graph generated by GrAALF
GrAALF generates a focused sequence of system calls from the provenance graph
A snippet of the GrAALF output
The sequence of system calls is sent to the HMM to perform causal inference
An HMM allows us to infer a sequence of system states given the emitted system calls (observations)
States
Server-daemon operation
Bash execute process
Generic process operation
Process upload-download
Observations
Vsftpd executes bash
Bash executes cat
Zip reads sensitive file
Zip writes to IP
The HMM output, the most likely sequence of states, represents an attack storyline
The attack storyline provides an abstraction layer to system call sequences
The abstracted behavior is sent to a machine learning model to determine the attack phase
Machine learning for classifying attacks
Softmax
Output: 6
LSTM-B
Output: 100
LSTM-A
Output: 5465 x 100
Embedding
Output: 5465 x 100
Input
114 x 5465

```
2018-09-28 11:51:51.356210805 accept 15858 vsftpd  /home/user_1/installs/vsftpd-2.3.4/ vsftpd  upstart 15858 1540
172.19.48.77 41015 <NA> <NA> <NA> 172.19.48.77:41015->172.19.48.76:21 4 172.19.48.76 ip 21 ipv4 4 root 0 245302
fd=4(<4t>172.19.48.77:41015->172.19.48.76:21) tuple=172.19.48.77:41015->172.19.48.76:21 queuepct=0 queuelen=0
queuemax=32  /bin/bash
2018-09-28 11:51:51.356529237 clone 15858 vsftpd  /home/user_1/installs/vsftpd-2.3.4/ vsftpd  upstart 15858 1540 <NA>
<NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> root 0 245304 res=16031(vsftpd)
exe=/usr/local/sbin/vsftpd args= tid=15858(vsftpd) pid=15858(vsftpd) ptid=1540(upstart) cwd= fdlimit=1024 pgft_maj=0
pgft_min=177 vm_size=10856 vm_rss=1372 vm_swap=0 comm=vsftpd
cgroups=cpuset=/.cpu=/.cpuacct=/.io=/.memory=/.devices=/user.slice.freezer=/.net_cls=...
flags=72(CLONE_NEWIPC|CLONE_NEWPID) uid=0 gid=0 vtid=15858(vsftpd) vpid=15858(vsftpd)  /bin/bash
2018-09-28 11:51:51.356564008 close 15858 vsftpd  /home/user_1/installs/vsftpd-2.3.4/ vsftpd  upstart 15858 1540
172.19.48.77 41015 <NA> <NA> <NA> 172.19.48.77:41015->172.19.48.76:21 4 172.19.48.76 ip 21 ipv4 4 root 0 245306
res=0  /bin/bash
2018-09-28 11:51:51.356604502 clone 16031 vsftpd  /home/user_1/installs/vsftpd-2.3.4/ vsftpd  vsftpd 16031 15858 <NA>
<NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> root 0 245309 res=0 exe=/usr/local/sbin/vsftpd args=
tid=16031(vsftpd) pid=16031(vsftpd) ptid=15858(vsftpd) cwd= fdlimit=1024 pgft_maj=0 pgft_min=0 vm_size=10856
vm_rss=124 vm_swap=0 comm=vsftpd
cgroups=cpuset=/.cpu=/.cpuacct=/.io=/.memory=/.devices=/user.slice.freezer=/.net_cls=...
flags=72(CLONE_NEWIPC|CLONE_NEWPID) uid=0 gid=0 vtid=1(systemd) vpid=1(systemd)  /bin/bash
```

```
[ ...

{ "sequence_number": 37001, "user": "root", "from_id": 15858 ,
"from_name": "vsftpd", "evt_type": "exec", "to_name": "sh" ,
"to_id":16031 ,  "count": 2}
,
{ "sequence_number": 130618, "user": "root", "from_id": 16031 ,
"from_name": "sh", "evt_type": "exec", "to_name": "cat" ,
"to_id":16078 ,  "count": 2}
,
{ "sequence_number": 187465, "user": "root", "from_id": 16094 ,
"from_name": "zip", "evt_type": "read", "to_name":
"/home/user_1/sensetive_file.txt" , "to_id":7 ,  "count": 2}
,
{ "sequence_number": 187470, "user": "root", "from_id": 16094 ,
"from_name": "zip", "evt_type": "write", "to_name":
"172.19.48.77:48270->172.19.48.76:6200" , "to_id":6 ,  "count": 2}

... ]
```

Init-server-daemon → execute-process → server-daemon-operation → bash-executes-process (x2) → system-information → execute-process (x6) → file-operation (x4) → daemon-upload-download → system-operation

Figure 1: Component steps in Cyberian's pipeline. System call logs containing possible attacks (here, we show a data exfiltration attack) is loaded into a graphical tool. Templated queries help identify possible attack phases and HMM based inference on these traces generates a comprehensible 'storyline', which facilitates identification of the attack phase using an LSTM classifier.

asset discovery; the converse was also true. Notice that *asset discovery* is the worst classified attack phase overall. From the low F1-scores combined with the shared high confidence between these two classes, we may reason that the storylines for these two phases have high similarity. Misclassified instances of *network discovery* were always predicted to be *asset discovery*, although the converse was rarely true.

To test this methodology's adaptability to honeypots in different settings, we evaluate system call logs from the DARPA Transparent Computing program (Computing 2020). GrAALF discovered six attack phases from Engagements 2 and 4 of this program, containing 380 and 625 million records respectively. Five represent activities falling within our chosen attack phases, while one (which we label as 'drop file' phase) does not. We create ground truths for the first 5 new sequences and train new instances of our classifiers with those and the 114 original traces. Then we test the learned models on storylines for all 6 DARPA traces. The SVM correctly classifies two traces, while the LSTM classifies one correctly. Confidencesfor both models' incorrect predictions are significantly lower than their mean values. The exception is the LSTM's prediction on the drop file phase, which is *exfiltration* and has very high confidence. *The CNN correctly classifies all but the drop file phase*; notably that trace is confidently predicted to be *system reconnaissance*, the class with the most instances and representing the broadest activity category. As differences between F1-scores of the models are not significantly different in Table 7 and the CNN demonstrates better ability to incorporate new data in this methodology, the CNN may be the better approach for future deployment.

Concluding Remarks

Through raw system call analysis using GrAALF, storyline generation through an HMM, and CNN-based machine learning, this pipeline of candidate methods and models, which we name Cyberian, achieves an attack phase classification accuracy approaching 90% (Figure 1). Thus, Cyberian is effective in identifying attack behavior on a host based on distinct steps of various attacks. Our results demonstrate Cyberian's ability to identify actionable phases of attacks from large system logs collected in honey pots, for further use.

One limitation of the evaluation is the relatively small number of distinct attack phases analyzed (though corresponding logs are extensive). However, attacks tend to be infrequent and results show that significant accuracy is attainable from the overall methodology even when relatively few phases are available for training. One benefit of this general methodology is that it can follow the same procedure to identify additional phases when system call logs for these other phases exist to use during training. The framework is not restricted to only using the phases in the reported experiments. A future direction of research is to experiment with more sophisticated attacks to increase dataset and observation diversity; it is likely that some of these additional logs will come from more sophisticated methods of privilege escalation and data exfiltration as well as new phases such as drop files.

In addition, a more sophisticated method is needed to handle the possibility of encountering storylines that do not belong to known attack phases (either benign or an unknown phase); open set recognition is a promising avenue of research for this purpose. This approach seeks to prepare models to effectively deal with classes unseen during training while accurately classifying seen classes (Geng, Huang, and Chen 2020). A more robust version of Cyberian with such abilities could identify mistakenly generated benign storylines or storylines representing attack phases beyond the known set.

Acknowledgments

This research was supported, in part, by a grant from the Army Research Office under grant number W911NF-18-1-0288. We also acknowledge discussions with Prof. Munindar Singh's group at NCSU, which helped shape this research.

References

Baum, L. E.; Petrie, T.; Soules, G.; and Weiss, N. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics* 41(1): 164–171.

Bolzoni, D.; Etalle, S.; and Hartel, P. H. 2009. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *International Workshop on Recent Advances in Intrusion Detection*, 1–20. Springer.

Chevalier, R. 2019. *Detecting and Surviving Intrusions: Exploring New Host-Based Intrusion Detection, Recovery, and Response Approaches*. Ph.D. thesis, CentraleSupélec.

Computing, D. T. 2020. Transparent Computing Engagement 5 Data Release. https://github.com/darpa-i2o/Transparent-Computing. [accessed 26 August 2020].

Du, M.; Li, F.; Zheng, G.; and Srikumar, V. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 1285–1298. New York, NY, USA: ACM. ISBN 978-1-4503-4946-8. doi:10.1145/3133956.3134015. URL http://doi.acm.org/10.1145/3133956.3134015.

Gao, P.; Xiao, X.; Li, D.; Li, Z.; Jee, K.; Wu, Z.; Kim, C. H.; Kulkarni, S. R.; and Mittal, P. 2018a. {SAQL}: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *27th USENIX Security*, 639–656. USENIX.

Gao, P.; Xiao, X.; Li, Z.; Xu, F.; Kulkarni, S. R.; and Mittal, P. 2018b. {AIQL}: Enabling Efficient Attack Investigation from System Monitoring Data. In *2018 USENIX Annual Technical Conference*.

Garcia, K. A.; Monroy, R.; Trejo, L. A.; Mex-Perera, C.; and Aguirre, E. 2012. Analyzing log files for postmortem intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(6): 1690–1704.

Geng, C.; Huang, S.-j.; and Chen, S. 2020. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Joachims, T. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, 137–142. Springer.

Laan, N. C.; Pace, D. F.; and Shatkay, H. 2006. Initial model selection for the Baum-Welch algorithm as applied to HMMs of DNA sequences. *Queen's University, Kingston, ON, Canada* .

Lafferty, J.; McCallum, A.; and Pereira, F. C. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001*.

Lee, S.-M.; Yoon, S. M.; and Cho, H. 2017. Human activity recognition from accelerometer data using Convolutional Neural Network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 131–134. IEEE.

Lewes, T. C. 2019. SOF-ELK® Virtual Machine Distribution. https://github.com/philhagen/sof-elk/blob/master/VM_README.md. [Online; accessed 25 May 2019].

Lippmann, R. P.; Fried, D. J.; Graf, I.; Haines, J. W.; Kendall, K. R.; McClung, D.; Weber, D.; Webster, S. E.; Wyschogrod, D.; Cunningham, R. K.; et al. 2000. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, 12–26. IEEE.

Liu, M.; Xue, Z.; Xu, X.; Zhong, C.; and Chen, J. 2018. Host-Based Intrusion Detection System with System Calls: Review and Future Trends. *ACM Computing Surveys (CSUR)* 51(5): 98.

Milajerdi, S. M.; Gjomemo, R.; Eshete, B.; Sekar, R.; and Venkatakrishnan, V. 2019. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, 1137–1152. IEEE.

Project, P. 2019. Plaso (log2timeline). https://plaso.readthedocs.io/en/latest/. [Online; accessed 25 May 2019].

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2): 257–286.

Radhakrishna, V.; Kumar, P. V.; and Janaki, V. 2016. A Novel Similar Temporal System Call Pattern Mining for Efficient Intrusion Detection. *J. UCS* 22(4): 475–493.

Rapid7. 2020. metasploit. https://metasploit.com/. [Online; accessed 20 Jan 2020].

Schreiber, J. 2017. Pomegranate: fast and flexible probabilistic modeling in python. *The Journal of Machine Learning Research* 18(1): 5992–5997.

Setayeshfar, O.; Adkins, C.; Jones, M.; Lee, K. H.; and Doshi, P. 2019. GrAALF: Supporting Graphical Analysis of Audit Logs for Forensics. *arXiv preprint arXiv:1909.00902* .

Strom, B. E.; Applebaum, A.; Miller, D. P.; Nickels, K. C.; Pennington, A. G.; and Thomas, C. B. 2018. Mitre Att&ck: Design and Philosophy. Technical report, MITRE Corp.

Wang, W.; Guan, X.; and Zhang, X. 2004. Modeling program behaviors by hidden Markov models for intrusion detection. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 5, 2830–2835 vol.5. doi:10.1109/ICMLC.2004.1378514.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.