Automated Design of Network Security Metrics

Aaron Scott Pope
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri
Los Alamos National Laboratory
Los Alamos, New Mexico
aaron.pope@mst.edu

Robert Morning
Los Alamos National Laboratory
Los Alamos, New Mexico
rmorning@lanl.gov

Daniel R. Tauritz
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri
dtauritz@acm.org

Alexander D. Kent
Medtronic
Minneapolis, Minnesota
alex.kent@medtronic.com

ABSTRACT

Many abstract security measurements are based on characteristics of a graph that represents the network. These are typically simple and quick to compute but are often of little practical use in making real-world predictions. Practical network security is often measured using simulation or real-world exercises. These approaches better represent realistic outcomes but can be costly and time-consuming. This work aims to combine the strengths of these two approaches, developing efficient heuristics that accurately predict attack success. Hyper-heuristic machine learning techniques, trained on network attack simulation training data, are used to produce novel graph-based security metrics. These low-cost metrics serve as an approximation for simulation when measuring network security in real time. The approach is tested and verified using a simulation based on activity from an actual large enterprise network. The results demonstrate the potential of using hyper-heuristic techniques to rapidly evolve and react to emerging cybersecurity threats.

CCS CONCEPTS

• **Security and privacy → Network security**; • **Software and its engineering → Genetic programming**; • **Mathematics of computing →** *Graph algorithms*; *Approximation algorithms*;

KEYWORDS

Network security, genetic programming

ACM Reference Format:
Aaron Scott Pope, Robert Morning, Daniel R. Tauritz, and Alexander D. Kent. 2018. Automated Design of Network Security Metrics. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208266

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208266

1 INTRODUCTION

In an age where new software vulnerabilities are discovered and exploited on a daily basis, best practices and fast response are insufficient to secure a large computer network. Administrators need to be able to understand, analyze, and track the level of security in networks they manage. As enterprise computer networks continue to grow in size and complexity, manual methods of analyzing network security are increasingly infeasible. Automated analysis tools are needed to highlight vulnerabilities and allow a pro-active defense strategy.

A common approach to analyzing computer networks is to model the network with a graph representation. Graphs can be used to model the physical or logical connectivity between computers on a network [3]. Alternatively, a graph might be used to represent the communication between networked machines [12]. Attack graphs are an example of a graph-based representation specific to security [24]. These graphs illustrate the potential paths an adversary can take to reach some compromise objective. Authentication graphs are a type of attack graph that can be used to identify the regions of a network an intruder can reach using stolen credentials [27]. Since graphs provide such a natural representation for networks, many network analysis techniques rely on graph-based heuristics.

This work demonstrates the feasibility of using hyper-heuristic techniques to automate the development of novel graph-based network security metrics. User activity data from a large real-world network is used to simulate network attacks that model adversaries traversing the network with stolen user credentials. These simulation results are used to guide the evolution of new graph heuristics that accurately predict attack success. The rest of this paper is organized as follows: Sections 2 and 3 introduce some related background concepts. Related work is discussed in Section 4. Details of the methodology and experimental design is described in sections 5 and 6, respectively. Results are presented and discussed in Section 7. Section 8 offers some conclusions and Section 9 discusses possible future work.

2 NETWORK AUTHENTICATION

Centralized single-sign-on systems, such as Kerberos [21], allow organizations to manage access control on a large scale. The credentials used to access a computer are often stored in a specialized cache on that machine. A variety of methods exist which allow an adversary to retrieve these credentials from a compromised computer [5]. Once the credentials have been obtained, they can be used to access and compromise other computers on the network. This entire process can be applied repeatedly, allowing an intruder to continue to traverse a growing portion of the network. The most notorious example of exploiting stolen credentials, known as *pass-the-hash*, abuses the weakness of reusable password hashes in older networks using Windows NT LAN Manager [11]. However, similar principles make this type of replay attack possible on modern systems as well, such as Kerberos [1].

2.1 Bipartite Authentication Graphs

The computers on a network and the user accounts that access them can be naturally represented as two independent sets of nodes in a bipartite authentication graph (BAG) [14, 15]. An edge in this graph connects a user node to a computer node and represents an occurrence where the user's authentication credentials are used to access the computer. This access can be direct (e.g., a user logging into a workstation) or indirect (e.g., through SSH or a remote desktop session). If the same account credentials are used to authenticate on additional computers, as is common in environments using centralized single-sign-on systems, then the corresponding user node will be adjacent to multiple computer nodes.

This graph representation makes it possible to identify the portions of a network which are vulnerable to credential theft attacks [27]. For example, if the computer $C1$ in Figure 1 is compromised, the credentials for user $U1$ could be stolen. The existing edges of the BAG indicate that the credentials for user $U1$ can also be used to access computers $C2$, $C3$, and $C4$. As a result, an adversary armed with the stolen credentials for user $U1$ would also be able to gain access to these additional computers.

Under normal circumstances, a computer's cache would only contain a subset of the credentials used to access the machine due to limits on the cache size or credentials being periodically removed. If the edges of a BAG incident to a given computer represent the authentication credentials assumed to be currently stored in that computer's cache, then upon compromise, the adversary can gain access to the credentials of all adjacent user nodes in the BAG. These new credentials could then potentially be used to access additional computers on the network. By repeating this process, the adversary can continue traversing the connected nodes, compromising a growing portion of the network.

2.2 Graph Heuristics

Because graphs are a natural way of representing computer networks, there are many examples of network applications that rely on graph heuristics. Minimal spanning tree heuristics are used to control network routing to avoid problematic cycles [23]. Graph partitioning methods are used to segment large computer networks to make it difficult for adversaries to penetrate the network [5, 27]. Path analysis heuristics are used on attack graphs to identify the likely routes attackers will use to compromise network resources [17, 22, 24, 28].

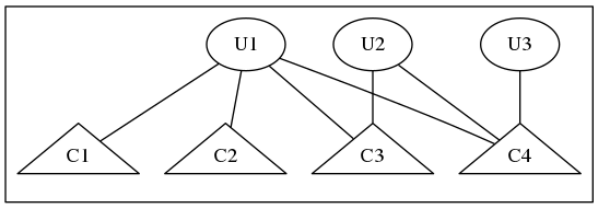


Figure 1: Example BAG with users U1, U2 and U3 and computers C1, C2, C3 and C4. An edge represents an authentication event between a user and a computer.

It is possible to achieve improved algorithm performance by using heuristics that exploit graph characteristics that are common in an application area [25]. Machine learning techniques have been used to automate the process of selecting the best heuristic for a problem from a set of available heuristics with high accuracy [10]. Unfortunately, this approach is limited by the quality and variety of the set of predefined heuristics; an optimal solution to a given problem cannot be selected if it is not already present in the heuristic set. Instead, domain expertise can be exploited to design novel customized heuristics tailored to a specific application. The process of designing new heuristics can be accomplished manually, but this can be difficult and time-consuming, often leading to an incomplete set of optimal heuristics. An alternative approach is to use hyper-heuristic machine learning techniques to automate the design and optimization of novel algorithms [4, 29].

3 GENETIC PROGRAMMING

Hyper-heuristics most commonly employ genetic programming (GP) to search a problem-specific space of algorithmic primitives. In GP, the solutions being evolved typically take the form of programs or heuristics. GP has been shown capable of automatically generating and optimizing heuristics for problems in a variety of domains, including graph algorithm applications [2, 8, 25, 26]. A set of primitive operations is usually constructed by observing the common and essential elements of algorithms which have been designed to solve the intended problem. This primitive operation set is used as algorithmic building blocks by the GP to piece together new candidate algorithm solutions.

4 RELATED WORK

There are many related works that employ graphs as an abstract representation of networks. Network connections, both physical and logical, can be modeled using graphs to visualize the network's topology [3, 23]. Communication between networked machines is also commonly modeled using weighted graphs where the edge weight represents the amount or frequency of communication between machines [12]. In particular, NetFlow communications lend themselves well to graph representations [18]; these provide a high-level, session-based view of the interaction between networked computers. Dynamic graph models have also been used to represent the changes or activity on a network over a period of time [6].

Graphs are particularly useful in network security applications. Graph partitioning methods have been used to segment networks to mitigate the damage potential of an intruder traversing the network [27]. Most graph partitioning techniques minimize the number or weight of edge removals needed to disconnect components of the graph; this feature can be leveraged to minimize the effort needed to segment the corresponding network or limit the impact on network user productivity. Attack graphs are another common abstraction method, this time for representing the avenues an adversary can take during a multi-step intrusion process [24]. Because of their usefulness for risk analysis and network hardening, automated methods of generating and evaluating these attack graphs have been developed [28].

This work builds on previous research that introduced the use of bipartite authentication graphs (BAGs) to model network user activity [14]. BAGs can be used in lieu of traditional attack graphs when the attack model is focused on traversal using stolen user credentials. An advantage of using BAGs over other attack graphs is that they can be constructed without detailed information about the vulnerabilities on individual networked host machines. Authentication graphs can be constructed using logs from centralized authentication systems, which are often already being collected in enterprise networks. BAG representations have been used in previous work to identify anomalous user activity [30] and segment networks by finding minimal access control policy changes [27].

Although there are numerous examples of graph algorithms being applied to network security problems, many of these utilize general-purpose graph heuristics that do not exploit the specific characteristics of graphs that represent computer networks. Hyper-heuristics have been used to tailor heuristics to specific application domains [4], including those involving graph algorithms. Previous work investigated the use of hyper-heuristics to generate and optimize random graph generation heuristics that produce graphs with desirable characteristics, such as specific centrality distributions or community structures [2, 8, 26]. Customized graph partitioning heuristics have also been generated that improve upon the performance of general-purpose algorithms for targetted classes of graphs, including those representing computer networks [25].

5 METHODOLOGY

Network authentication events, consisting of a time stamp, a user account, and a network hostname, are used to construct a dynamic BAG. This graph provides the environment for a randomized credential theft and network traversal attack simulation. The simulation has two configurable settings meant to replicate authentication policy controls. The first parameter controls the duration a credential would be stored in a host's cache after an authentication event occurred. Repeated authentication events refresh this duration on existing credentials. If a credential is not refreshed before the configurable time limit is exceeded, the credential is rendered inactive, removing it from the host's cache. The second parameter controls the maximum size of the credential cache stored on the hosts. If an authentication event would add a credential to a cache that exceeds this limit, the oldest stored credential is removed to make room.

5.1 Lateral Movement Simulation

The attack simulation represents an adversary attempting to traverse the network with compromised user credentials. An adversary is initialized with a single compromised host. In this work, the initial host is chosen at random to represent a network computer inadvertently installing malware, potentially as a result of a phishing campaign. It is assumed that once a host is compromised, the adversary gains access to the credentials active on that computer.

At each subsequent time-step, the adversary will use any credentials they have accumulated to access additional hosts on the network. The simulation assumes the credential must be active on the additional hosts as a result of a legitimate authentication event for the adversary to compromise those computers. This behavior resembles a passive adversary attempting to hide their movement amongst legitimate activity in an effort to avoid detection. If an adversary chose to attempt access to a host not typically used by the impersonated user, the access is more likely to trigger a network intrusion alarm [20].

Whenever a new host is compromised by the adversary, any credentials on that host are added to the adversary's collection. This process is immediate, assuming the adversary is employing scripted exploit methods. Since the adversary is assumed to be automated, their traversal is not limited to a single host at a time. The adversary continuously harvests credentials from all compromised hosts simultaneously, seeking to compromise an ever-growing portion of the network. See Figure 2 for an example of lateral movement to additional network hosts. Once the configurable simulation time limit has expired, the set of compromised hosts is returned as an indication of attack success.

5.2 Compact Graph Representation

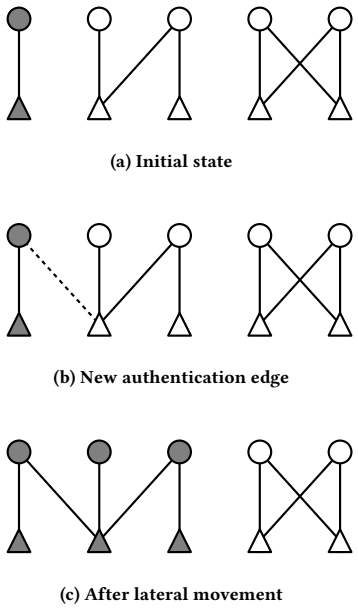
In order for the machine learning process to interact with reasonably sized graph representations, the dynamic bipartite authentication graph is used to produce a series of compact graphs that summarize the activity for 24 hour periods. This compressed representation assigns a weight to each authentication edge that is the count of the number of timesteps that authentication is active during a day. This representation is chosen because it presents the evolutionary process with two interpretation options. The edge weight can be ignored, which would only consider the existence of authentication edges. Alternatively, the weights can be compared to differentiate between edges by their relative probability to be active.

5.3 Hyper-Heuristic Approach

A population of graph heuristics is evolved to predict the chance of simulated attack success given a compact authentication graph model.

Representation: Candidate solution algorithms are represented using strongly-typed genetic programming (GP) parse trees [19].

Initialization: An initial population of parse tree solutions is randomly constructed from the available input and operation nodes. A configurable maximum height parameter is used to limit the size of the initial parse tree solutions. Ramped half-and-half solution generation is used, which produces full parse trees of maximum



(a) Initial state

(b) New authentication edge

(c) After lateral movement

Figure 2: Lateral movement simulation example. Circles represent user credential nodes and triangles represent hosts. Compromised hosts and credentials are indicated by shaded nodes.

height for half the population and variable height trees (up to the maximum) for the remainder.

Evaluation: Solutions are evaluated by comparing their output to the attack simulation results. For each day of activity, the attack simulation result is compared to the return value of the individual heuristic. The absolute percentage differences between these values are summed and normalized by the number of days to find the error rate of the prediction heuristic. This error rate is negated and used as the fitness for the solution, as shown in Equation 1. The evolutionary process attempts to maximize these fitness scores, producing solutions with low error rates. Solutions that fail to return a result within a configurable time limit have their fitness values set to negative infinity to discourage inefficient solutions.

$$fitness = -\frac{\sum_{d \in days} \left| \frac{simulated\_result - predicted\_result}{simulated\_result} \right|}{|days|} \quad (1)$$

Parent Selection: Parents are selected by taking a random sample of size k from the population and choosing the solution from the sample with the best fitness. This is known as k-tournament selection.

Recombination: Due to the destructive nature of parse tree variation operators, offspring are generated using either recombination or mutation, not both [16]. If a pair of parent solutions are selected for recombination, two offspring are produced using random subtree crossover.

Mutation: If recombination is not selected, an offspring is created by cloning a single parent, then performing random subtree replacement.

Survival Selection: Elitist truncation is used for survival selection, simply selecting the solutions with the best fitness.

Termination: Execution of the GP is terminated when a configurable number of generations have passed without any improvement in the average population fitness (convergence threshold).

Parameters: The parameters for the GP can be seen in Table 1. Aside from the convergence threshold, these values were programmatically tuned by a random-restart hill-climbing search attempting to optimize the best fitness found during evolution. This tuning process was also used to choose the parent and survival selection techniques. Other selection methods considered include fitness proportional and uniform random selection. The possible values for these parameters were inspired by previous work evolving graph algorithms [25, 26]. The convergence threshold was hand tuned to ensure termination in the time available.

Table 1: GP Parameter Values

Parameter	Value
Population size	400
Offspring per generation	600
Parent selection tournament size	8
Minimum initial parse tree height	4
Maximum initial parse tree height	7
Recombination probability	70%
Mutation probability	30%
Convergence threshold	10

5.4 Primitive Operations

The following categories of operations make up the set of primitives available to the GP. These operations were inspired by previous work on authentication graph analysis [15, 27] and the automated design of graph-based heuristics [2, 7, 25, 26].

Math operators: Basic addition, subtraction, multiplication, division, modulus, exponentiation, additive and multiplicative inverse. Some of these operators require special attention due to the stochastic nature of the process. For example, if division would produce a division by zero exception, it instead divides by a value very close to zero. These include operations that reduce a set of values to a single value, such as *sum* and *average*.

Numerical constants: Return a constant integer ($\{0, 1, 2, \ldots, 10\}$) or probability ($\{0.001, 0.01, 0.1, 0.2, 0.3, \ldots, 1.0\}$) value randomly chosen once during initialization. These possible values were inspired by previous work evolving graph heuristics [25, 26].

Boolean nodes: True and false constant nodes, as well as a node that randomly returns true according to an input probability.

Control flow: Standard *if-then-else* style conditional branching, as well as *for* and *while* loops.

Global graph metrics: Metrics based on the entire graph, such as average degree, number of nodes, or graph diameter.

Graph elements: Collections of graph edges and nodes.

Graph element metrics: Metrics associated with graph elements,

Table 2: LANL Authentication Dataset Details

Unique Users	10,044
Unique Computers	15,779
Unique (User, Computer) Pairs	124,020
Total Authentication Events	101,918,344
Average Daily Authentication Events	2,547,958.6

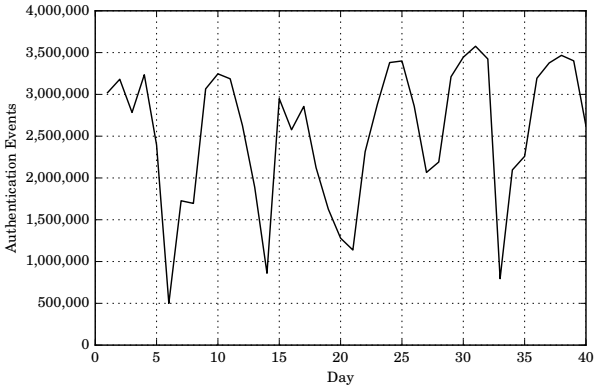


Figure 3: Count of authentication events per day.

such as node centrality values or edge weights.
Maps: Map a collection of elements to their respective metrics. For example, the betweenness centrality of a set of nodes.
Collection manipulation: Manipulate collections, such as concatenation or conditional filtering.
Subgraph induction: Induces a subgraph from a collection of nodes or edges.

6 EXPERIMENT

Dynamic BAGs are constructed from the authentication data produced by Los Alamos National Laboratory (LANL) [13]. One such BAG is produced for every day for the first forty days of activity in the dataset. A summary of this data can be seen in Table 2. Figure 3 shows the daily number of authentication events with an obvious weekly pattern. Presumably, the valleys correspond to weekends, but it is interesting to note that most of the weeks only show a single day with dramatically fewer authentication events. This could be the result of automated processes, such as patch management services, running on a weekly basis during employee downtime to minimize network user impact. Unfortunately, the anonymized dataset does not contain enough information to better explain this pattern.

The lateral movement attack model is simulated for each day for two possible credential policy configurations. The first assumes a maximum cache size of ten, removing the oldest cached entries to make room for new credentials. The second configuration removes credentials from host caches after one hour. Simulations are initialized by populating the graph with twelve hours of network activity. After initialization, the simulated attack begins with

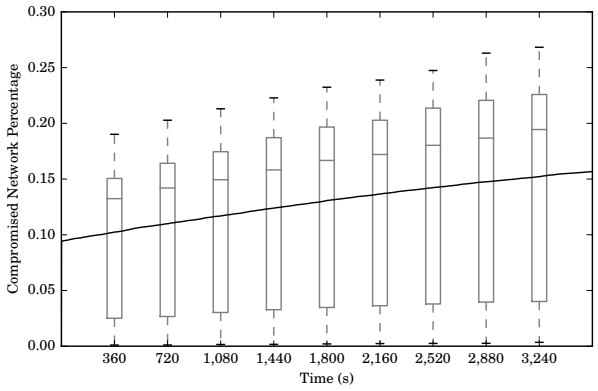


Figure 4: Mean simulation results using 1-hour ticket lifetime policy. The vertical axis shows the percentage of the network compromised, averaged over all adversaries, at each time step. The boxes indicate the variation between different days.

a single, randomly chosen point of compromise. The simulation proceeds for one hour and the degree of success is measured by the percentage of the networked hosts that are compromised upon termination. See Figure 4 for an illustration of how the the percentage of the network compromised tends to grow over the course of the attack simulation. The upward trend shows how the portion of the network compromised grows during the simulation. The high variation and skew are a result of the inclusion of days with very low authentication activity; this dramatically limits an adversary's ability to reach a large portion of the network.

This success measure for each day is averaged over one hundred repeated simulations, each with a different initial point of compromise. Figure 5 shows the final average simulation outcome for each day and each policy configuration. Again, the weekly pattern is obviously present. Compared to the ten credential limit policy, the one hour credential expiration policy consistently reduces the success of the adversary's compromise percentage. Further examination of the graphs produced by this data suggests that this is due to authentication edges that connect computers accessed infrequently by a small number of users; these edges tend to remain active for long periods of time within the simulation.

The first thirty days of the dataset are used to train the GP. A compact graph representation of each day's activity is created as described in Section 5.2. Figure 6 shows the distribution of authentication edge activity levels for both policy configurations. Edges near the lower end of the horizontal axis are only active for short periods during the simulation. Alternatively, edges near the upper end are active for the majority of the simulated period. The dramatic difference between these distributions, especially at high edge activity levels, illustrates the impact of the policy configuration selected. These edge activity levels are available to the evolved heuristics as edge weights, allowing some of the temporal information to be leveraged despite the static nature of the compact graph representation. For each policy configuration, a population of heuristics is evolved to predict the simulation outcomes. Additionally, a third

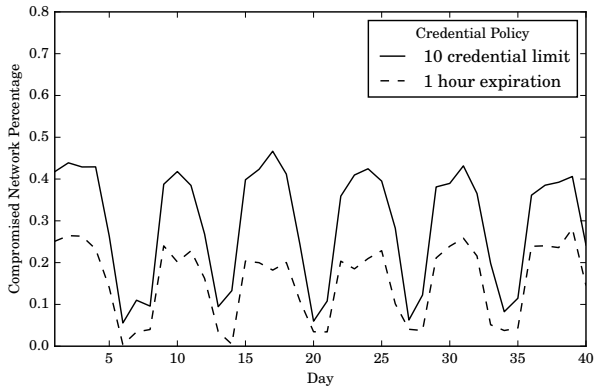
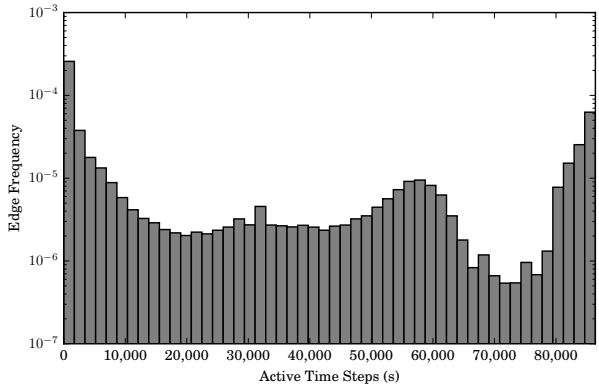
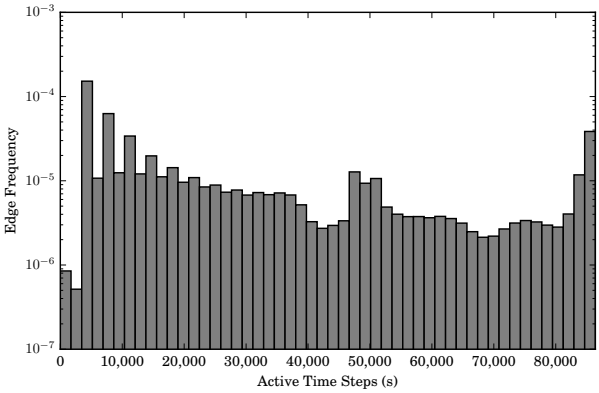


Figure 5: Daily simulation results for both credential policies.



(a) 10 credential limit policy



(b) 1 hour expiration policy

Figure 6: Distribution of authentication edge activity levels for compressed graphs for both credential policies. Low values (to the left) indicate edges that are rarely active in the original dynamic graph. High values indicate edges that are active the majority of that day. Note that the vertical axes are log scaled.

population is trained to predict the simulation outcomes for both policy configuations simultaneously. This is done to examine the benefit of focusing on specific credential policies instead of seeking a more generalized heuristic. The performance of the best final evolved solutions from each population is measured against the simulation results for the final ten days for validation.

7 RESULTS AND DISCUSSION

Although the evolved heuristics are too complex to be included here (the smallest being over 200 lines of code), some functional elements that commonly occur are:

(1) Induce a subgraph with the most active (highest weight) edges
(2) Find the connected components in the induced graph
(3) Filter out the account vertices in each component vertex set
(4) Return a value based on the number of computers in each component relative to the number of computers in the original graph

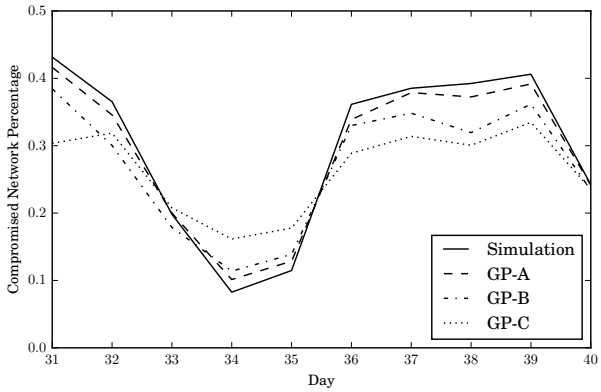
This is not surprising, considering how the connected components represent portions of the network which can be traversed with lateral movement.

Table 3 shows the simulated and predicted compromise percentages averaged over each day of the validation data. **GP-A** refers to the best heuristic trained using the ten credential limit policy. Similarly, **GP-B** indicates the heuristic produced by the one hour expiration policy. **GP-C** is the heuristic evolved to predict the combined simulation results. Figure 7 shows the daily comparison of the simulation results and predicted values for both credential policies. The distance between the predictions of the evolved heuristics and the simulation results is an indicator of the quality of the heuristic. Superior solutions lie closer to the simulated outcomes. In both policy configuration cases, the heuristic evolved to target that configuration (**GP-A** for Figure 7a and **GP-B** for Figure 7b) more accurately tracks the simulation results. The results suggest that it is beneficial to target the particular credential policy in use instead of seeking a more general purpose solution.

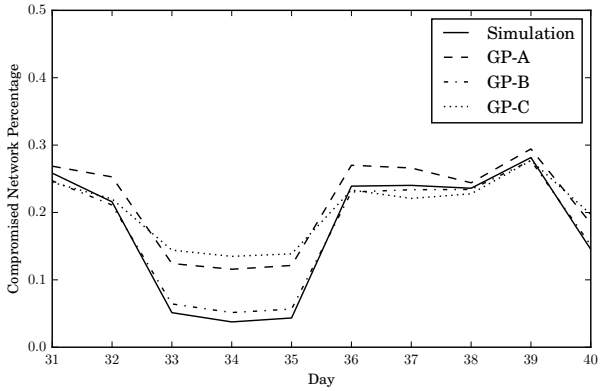
Table 3: Comparison of Evolved Metric Heuristics

	10 Credential Limit		1 Hour Expiration	
Method	**Result**	**Error**	**Result**	**Error**
Simulation	29.797%	N/A	17.484%	N/A
GP-A	29.151%	6.15%	21.411%	60.93%
GP-B	27.093%	14.85%	17.571%	11.23%
GP-C	26.427%	28.00%	20.387%	71.79%

One interesting outcome is that **GP-C**, which represents a "hybrid" approach, tends to perform worse than both of the more targeted heuristics; this is especially evident in Figure 7a. It is understandable for this attempt at a generalized heuristic to be performing worse than the appropriately targetted heuristic, but it is



(a) 10 credential limit policy



(b) 1 hour expiration policy

Figure 7: Comparison of simulated results and predictions from each evolved solution for both credential policies. Higher quality solutions lie closer to the simulation results.

surprising that, in many cases, it is also outperformed by the heuristic targetting the wrong credential policy. One possible explanation is that the differences between the compact graph representations for the two policy configurations made fine-tuned exploitation of specific graph properties difficult. Regardless of the true reason for the degraded performance, the results demonstrate the value of evolving heuristics tailored to the specific policy.

Figure 8 shows the mean population fitness for each generation, averaged over twenty repeated executions of the GP. Progress is indicated by an upward trend. The low initial fitness for the population of **GP-A** is likely the result the higher variation in the simulation results for the 10 credential limit policy. Evolution quickly overcomes this disadvantage, however, and both targeted GPs converge on similar fitness values. Aside from very early generations, **GP-C** consistently has lower population fitness compared to the other GPs.

The results presented here demonstrate the potential of hyper-heuristics to automate the development of novel network security

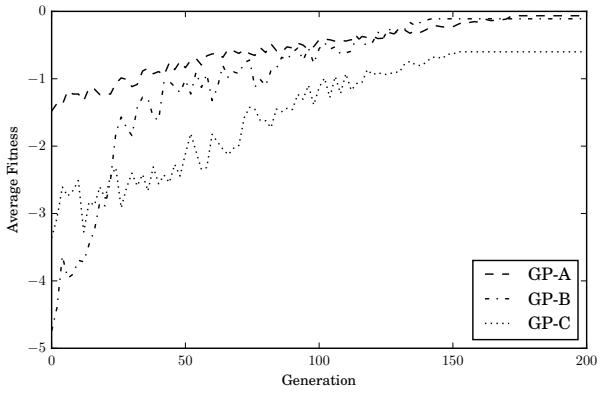


Figure 8: Population fitness value vs. generation for each evolved population. Fitness is averaged over twenty repeated executions of the GP.

metrics. In this work, evolution was guided by an attack simulation, but this could be replaced with data from penetration testing or genuine compromise events. Institutions with large computer networks are likely already collecting the data needed to train these heuristics. The approach presented in this work could provide system administrators a new capability to better leverage this data without relying on expert knowledge of the specifics of an adversary's techniques. This has the potential to reduce the time needed to understand and react to emerging threats.

8 CONCLUSION

In the ever-evolving world of cybersecurity, system administrators need new ways to understand and visualize risks and vulnerabilities. Manual analysis can be prohibitively expensive and time-consuming, limiting our ability to react to new adversary techniques. This work has demonstrated the potential of hyper-heuristic techniques for the automated development of network security metrics. Evolved heuristics were able to accurately predict simulated attacks on network models based on real-world data for a complex network. Automated design can improve our security capabilities enabling us to rapidly react to emergent threats with less reliance on subject matter experts. Although the current results are focused on computer networks, the approach could be easily extended to include physical domain elements for more comprehensive security.

9 FUTURE WORK

The fidelity of the attack simulation could be improved to more accurately model lateral movement attacks, including the addition of physical network components. Simulated results could also be replaced with data from historical compromise incidents or red-team network penetration testing. The entire approach could be applied to additional attack models, either existing models or entirely new attacks as they emerge. The automated nature of this approach has the potential to dramatically reduce the response time needed to react to new cybersecurity threats.

The final heuristics evolved in this work tend to be very large and complex. While some of this complexity is likely needed for

accurate solutions, manual inspection reveals several subtrees that do not contribute to the solutions' functionality. In [9], Helmuth et. al. demonstrate that automated simplification of evolved algorithms can improve their performance on unseen problem sets. While the customized heuristics evolved in this work are not intended to be general purpose, this approach, along with a fitness function that promotes generalization, could improve the results when applying these heuristics to other networks.

Although this work does examine the effect of targeting evolution on a specific credential policy configuration, there are other factors that could impact the performance gained from such tailored heuristics. The forty day time period used for evolution and validation is likely too short to exhibit dramatic changes in user behavior or network structure. These more gradual changes are inevitable over longer time periods as enterprises grow and networks increase in complexity. It is possible such changes will alter characteristics present in the graph representations of the network. If an evolved heuristic relies on characteristics that change over time, it could result in degraded accuracy. This impact should be examined by considering longer periods of time or including ranges of time that include more dramatic structural changes to the network. Doing this could provide insight into how often the evolution process should be repeated to adapt to the new environment.

This work builds on previous research that used graph-theoretic properties to predict and limit the potential impact of adversaries traversing the network with stolen credentials [15, 27]. The previous analytic methods could provide a baseline for comparing the performance of the heuristics evolved in this work. To enable a fair comparison, these methods would need to be adapted to model the dynamic nature of the graphs considered in this work or incorporate the necessary policy configuration details.

ACKNOWLEDGMENTS

This work was supported by Los Alamos National Laboratory via the Cyber Security Sciences Institute under subcontract 259565 and the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20170683ER.

REFERENCES

[1] [n. d.]. Microsoft Windows Kerberos 'Pass The Ticket' Replay Security Bypass Vulnerability. http:// www.securityfocus.com/bid/42435. ([n. d.]). Accessed: 2016-08-10.

[2] Alexander Bailey, Mario Ventresca, and Beatrice Ombuki-Berman. 2014. Genetic Programming for the Automatic Inference of Graph Models for Complex Networks. *IEEE Transactions on Evolutionary Computation* 18, 3 (2014), 405–419.

[3] Andre Broido and K. C. Claffy. 2001. Internet topology: connectivity of IP graphs. In *Scalability and Traffic Control in IP Networks*, Vol. 45. International Society for Optics and Photonics, 172–187. https://doi.org/10.1117/12.434393

[4] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.

[5] John Dunagan, Alice X. Zheng, and Daniel R. Simon. 2009. Heat-ray: Combating Identity Snowball Attacks Using Machine Learning, Combinatorial Optimization and Attack Graphs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY, USA, 305–320. https://doi.org/10.1145/1629575.1629605

[6] Frank Harary and Gopal Gupta. 1997. Dynamic Graph Models. *Mathematical and Computer Modelling* 25, 7 (1997), 79–87. https://doi.org/10.1016/S0895-7177(97)00050-2

[7] Sean Harris, Travis Bueter, and Daniel R Tauritz. 2015. A Comparison of Genetic Programming Variants for Hyper-Heuristics. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 1043–1050.

[8] Kyle Robert Harrison. 2014. *Network Similarity Measures and Automatic Construction of Graph Models using Genetic Programming*. Master's thesis. Brock University.

[9] Thomas Helmuth, Nicholas Freitag McPhee, Edward Pantridge, and Lee Spector. 2017. Improving Generalization of Evolved Programs Through Automatic Simplification. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, New York, NY, USA, 937–944. https://doi.org/10.1145/3071178.3071330

[10] Patricia D. Hough and Pamela J. Williams. 2006. Modern Machine Learning for Automatic Optimization Algorithm Selection. In *Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop*. 1–6.

[11] Chris Hummel. 2009. Why Crack When You Can Pass the Hash. *SANS Institute InfoSec Reading Room* 21 (2009).

[12] Yu Jin, Esam Sharafuddin, and Zhi-Li Zhang. 2009. Unveiling Core Network-Wide Communication Patterns through Application Traffic Activity Graph Decomposition. *ACM SIGMETRICS Performance Evaluation Review* 37, 1 (2009), 49–60.

[13] Alexander D. Kent. 2014. User-Computer Authentication Associations in Time. Los Alamos National Laboratory. (2014). https://doi.org/10.11578/1160076

[14] Alexander D. Kent, Lorie M. Liebrock, and Joshua C. Neil. 2015. Authentication graphs: Analyzing user behavior within an enterprise network. *Computers & Security* 48 (2015), 150–166. https://doi.org/10.1016/j.cose.2014.09.001

[15] Aric Hagberg Alex Kent, Nathan Lemons, and Joshua Neil. 2014. Connected Components and Credential Hopping in Authentication Graphs. In *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS*. 416–423. https://doi.org/10.1109/SITIS.2014.95

[16] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[17] Wei Li and Rayford B Vaughn. 2006. Cluster Security Research Involving the Modeling of Network Exploitations Using Exploitation Graphs. In *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06*, Vol. 2. IEEE.

[18] Pavel Minarik and Tomas Dymacek. 2008. NetFlow Data Visualization Based on Graphs. In *Visualization for Computer Security*, John R. Goodall, Gregory Conti, and Kwan-Liu Ma (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 144–151.

[19] David J. Montana. 1995. Strongly Typed Genetic Programming. *Evol. Comput.* 3, 2 (June 1995), 199–230. https://doi.org/10.1162/evco.1995.3.2.199

[20] Joshua Neil, Curtis Hash, Alexander Brugh, Mike Fisk, and Curtis B Storlie. 2013. Scan Statistics for the Online Detection of Locally Anomalous Subgraphs. *Technometrics* 55, 4 (2013), 403–414.

[21] B. C. Neuman and T. Ts'o. 1994. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine* 32, 9 (Sept 1994), 33–38. https://doi.org/10.1109/35.312841

[22] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaâniche. 1999. Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. *IEEE Transactions on Software Engineering* 25, 5 (1999), 633–650.

[23] Radia Perlman. 1985. An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN. In *ACM SIGCOMM Computer Communication Review*, Vol. 15. ACM, 44–53.

[24] Cynthia Phillips and Laura Painton Swiler. 1998. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms (NSPW '98)*. ACM, New York, NY, USA, 71–79. https://doi.org/10.1145/310889.310919

[25] Aaron S. Pope, Daniel R. Tauritz, and Alexander D. Kent. 2016. Evolving Multi-level Graph Partitioning Algorithms. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8. https://doi.org/10.1109/SSCI.2016.7849930

[26] Aaron S. Pope, Daniel R. Tauritz, and Alexander D. Kent. 2016. Evolving Random Graph Generators: A Case for Increased Algorithmic Primitive Granularity. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8. https://doi.org/10.1109/SSCI.2016.7849929

[27] Aaron S. Pope, Daniel R. Tauritz, and Alexander D. Kent. 2017. Evolving Bipartite Authentication Graph Partitions. *IEEE Transactions on Dependable and Secure Computing* (2017).

[28] Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. 2001. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference & Exposition II, 2001 (DISCEX '01)*, Vol. 2. 307–321. https://doi.org/10.1109/DISCEX.2001.932182

[29] Daniel R. Tauritz and John Woodward. 2017. Hyper-heuristics Tutorial. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, New York, NY, USA, 510–544. https://doi.org/10.1145/3067695.3067710

[30] M. Turcotte, J. Moore, N. Heard, and A. McPhall. 2016. Poisson Factorization for Peer-Based Anomaly Detection. In *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. 208–210. https://doi.org/10.1109/ISI.2016.7745472