Heat-ray: Combating *Identity Snowball* Attacks Using Machine Learning, Combinatorial Optimization and Attack Graphs

John Dunagan
Microsoft Research
jdunagan@microsoft.com

Alice X. Zheng
Microsoft Research
alicez@microsoft.com

Daniel R. Simon
Microsoft
dansimon@microsoft.com

Abstract

As computers have become ever more interconnected, the complexity of security configuration has exploded. Management tools have not kept pace, and we show that this has made *identity snowball* attacks into a critical danger. Identity snowball attacks leverage the users logged in to a first compromised host to launch additional attacks with those users' privileges on other hosts. To combat such attacks, we present Heat-ray, a system that combines machine learning, combinatorial optimization and attack graphs to scalably manage security configuration. Through evaluation on an organization with several hundred thousand users and machines, we show that Heat-ray allows IT administrators to reduce by 96% the number of machines that can be used to launch a large-scale identity snowball attack.

1 Introduction

The past decade has witnessed a plague of remote exploits that could be launched by any machine on the Internet against any other machine with a given vulnerability [61, 32, 40, 39]. To combat these attacks, the research community has developed a large number of defensive techniques: address space randomization [5], stack canaries [13], compartmentalized web browsers [19], self-certifying alerts [12], runtime dynamic dataflow analysis [36], and many others. Despite these advances, it seems unlikely that machine compromises can be completely eliminated; computer system defenders must expect that some small fraction of machines are compromised, either due to insider attacks [44], social engineering [57], or a more traditional vulnerability.

Over this same period of time, computers have become ever more interconnected. It is commonplace for organizations today to run *single-sign-on* identity services (e.g., using Kerberos [51]) for hundreds of thousands of users, while Internet identity services support hundreds of millions of users (e.g., Microsoft's Live ID [30]). Emerging federation technologies [46] are further expanding the scope of these identity services. For example, cloud applications running on EC2 [15] can already recognize both the user *alice@aol.com* according to *AOL* and the user *bob@yahoo.com* according to *Yahoo!*, and the applications can then implement access checks involving these users.

Unfortunately, the ability to authenticate users and set access policies has far outpaced the ability to manage these security policies. We show in this paper that the aggregate complexity of security configuration has made *identity snowball* attacks into a pressing danger. We introduce the term identity snowball attack to describe an attack launched after an initial machine compromise where the attacker leverages the identities of users currently logged in to the first compromised machine to compromise additional machines. If the currently logged in users have administrative privileges on these other machines, the additional compromises are likely trivial for the attacker. The attacker may then iterate this process of successive compromise. We explain the mechanics of such attacks in detail in Section 2. Such iterative use of identities obtained from compromised hosts dates back to the first Internet worm [50] and was more recently used to compromise machines across a number of institutions [49, 43]. However, to the best of our knowledge there has been no prior work analyzing the potential for identity snowball attacks within large enterprises.

The threat of identity snowball attacks is that they magnify other dangers: a single initial compromise can lead to a large number of compromised machines. We quantify this threat in Section 8 by analyzing security configurations in a single large organization containing several hundred thousand users and machines. We show that an attacker who compromises almost any machine in this organization can proceed to compromise many other machines. Given our expectation that some small fraction of machines within the organization will be com-

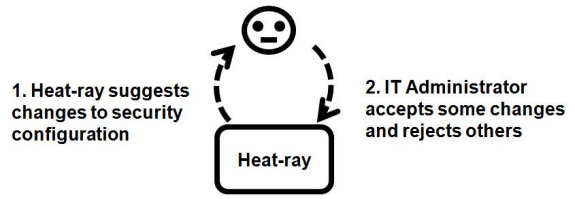

Figure 1: *An IT administrator uses Heat-ray iteratively to identify desirable security configuration changes.*

promised, this is absolutely unacceptable. We have only analyzed one organization in detail because of the sensitivity of the needed data (it is a map for how to compromise these machines). However, we expect that many organizations are similarly vulnerable for the following reasons:

- Granting additional privileges is frequently an easy way to enable administrative tasks on Windows [8], and Windows is prevalent in many large organizations. Industry white papers have also described similar issues on Unix [45, 53].
- Large organizations do not have the tools to understand when the aggregation of locally reasonable decisions to grant additional privileges have led to unacceptable global risk.

Although, to the best of our knowledge, identity snowball attacks have not been launched against the internal networks of large enterprises, the potential damage of such an attack motivates the development of defensive measures now. To this end, we present Heat-ray, a system designed to empower IT administrators to manage security configuration in large organizations. The operation of Heat-ray is depicted in Figure 1. On a periodic basis, Heat-ray presents a small number of high impact security configuration changes to an IT administrator. The IT administrator selects the changes they want to make, and the changes they prefer not to make. Heat-ray incorporates this feedback and learns the kinds of changes most acceptable to this administrator. The process repeats, with Heat-ray proposing additional security configuration changes until the IT administrator is satisfied with the new security configuration. Figure 2 shows the desired results.

Heat-ray identifies the most desirable set of configuration changes on each iteration through a combination of machine learning, combinatorial optimization and attack graphs. In Section 3 we explain how attack graphs capture the potential paths through which an attacker who has compromised one machine can compromise additional machines, and how configuration changes map to removing edges in this graph. In Section 4 we describe how Heat-ray integrates into a system for scalably collecting the data needed to create the attack graphs. In Section 5 we describe how Heat-ray applies sparsest cut, a combinatorial optimization technique related to min-cut, to find small sets of high impact configuration changes. As we show in Section 8, because sparsest cut prioritizes configuration changes by the impact they will have on the security of the organization as a whole, Heat-ray significantly outperforms heuristics that propose configuration changes based purely on local properties.

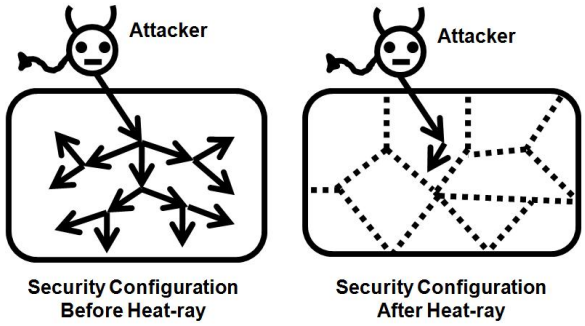

Figure 2: *After using Heat-ray, the initial compromise no longer results in an identity snowball attack.*

In Section 6 we describe how Heat-ray uses the Support Vector Machine technique from machine learning to address one final challenge, proposing configuration changes that are both high impact and *implementable*. An implementable configuration change is one that does not prevent users from accomplishing their jobs. For example, removing an IT administrator's ability to upgrade the software on a certain server is most likely not an implementable change. Due to the scale of the attack graph, it is impossible to explicitly label the implementability of each potential configuration change. Instead, Heat-ray treats the IT administrator's decision to accept or reject any proposed configuration change as implicitly indicating the implementability of the configuration change. Machine learning is used to generalize from this implicit feedback and re-estimate the implementability of other potential configuration changes. These revised estimates are incorporated into the sparsest cut algorithm as new edge costs in the attack graph. This causes future iterations of the sparsest cut algorithm to do a better job selecting configuration changes that are both high impact and implementable by the IT administrator. Finally, in Section 7, we describe how Heat-ray groups related configuration changes together and ranks its recommendations to reduce the burden on the IT administrator further still.

To the best of our knowledge, Heat-ray is the first system for defending against identity snowball attacks in large organizations. Prior work on improving the security configuration in a network of machines has required substantial manual effort by the IT administrator [2, 52, 48, 38, 35]. In particular, these systems may repeatedly propose configuration changes that are not

implementable, or they may require the IT administrator to specify particular high-value machines that must be defended. This manual burden renders these systems very difficult to apply in large organizations. In contrast, we show in Section 8 that Heat-ray allows an IT administrator to spend only modest effort (e.g., a few hours of their time) and identify desirable configuration changes in an organization with several hundred thousand users and machines. The identified configuration changes reduce by 96% the number of machines that can be used to launch an identity snowball attack reaching a large fraction of the organization.

2 Identity Snowball Attack Mechanics

We explain the mechanics of an identity snowball attack in the context of Kerberos, a widely deployed identity service. However, the attack we have defined is not exploiting a weakness in Kerberos. It is typical for modern identity services to entrust a computer with the authority to make requests on behalf of a logged in user. Once this trust has been granted, it is available to an attacker if the machine is compromised.

Figure 3 depicts the mechanics of Kerberos. In step 1, Alice provides her machine a secret, either by entering a password, by using a smartcard, or by some other method. Alice's machine uses this secret to obtain a Ticket Granting Ticket (TGT) from the Kerberos Key Distribution Center (KDC) – the TGT grants Alice's machine the right to perform actions on Alice's behalf. In step 2, Alice's machine stores the TGT locally, avoiding the need to repeatedly ask Alice for her secret. In step 3, Alice's machine presents the TGT to the KDC and obtains a Service Ticket (ST). In step 4, Alice's machine presents the ST to Bob's machine as part of a request on Alice's behalf (e.g., writing a file on Bob's machine that is marked "writable by Alice"). The ST proves to Bob's machine that Alice's machine has the authority to perform actions on Alice's behalf. Cryptographic techniques in Kerberos prevent Bob's machine from later using this same ST to convince other machines that it has the authority to perform actions on Alice's behalf.

Figure 4 depicts an identity snowball attack. In step 1, an attacker compromises Alice's machine. If Alice is already logged in, or if Alice then arrives at work and logs in, the TGT is stored somewhere on the computer, and the attacker can use it to generate STs at will.

In step 2, the attacker can attempt to compromise every machine where Alice has administrative privileges. A great variety of techniques are possible here because, by design, Alice has privileges that allow her to arbitrarily modify these machines. We enumerate a few such techniques here, including examples from both Windows and Unix. To find machines where Alice might have administrative privileges, the attacker might query some organizational directory service, scan all of Alice's email for machine names (e.g., "*"), scan the "/etc/hosts" file, snoop local broadcast traffic, or simply monitor outgoing and incoming TCP connections. To perform actions with Alice's ST, the attacker might read Alice's TGT out of memory, start a new process under Alice's login session, or possibly modify the parameters in some system call before it is executed by the OS on Alice's behalf. The actions performed with Alice's ST could include trying to write some security critical file or registry key on the remote machine, installing a new application, or configuring a system service insecurely so that it becomes a backdoor. We have verified a subset of these approaches ourselves in a controlled environment. Similar approaches have also been described previously [23]. Although individual circumstances may sometimes prevent compromise (e.g., because of network segmentation or firewall policies), the many available techniques suggest that in most cases, it is not safe to assume that an attacker will have difficulty exploiting these additional machines.

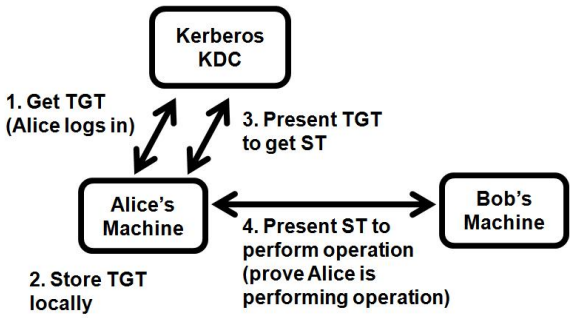

Figure 3: *How machines authenticate on a user's behalf in Kerberos.*

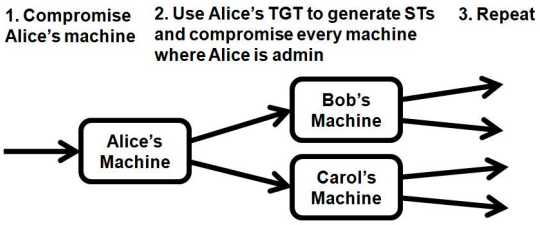

Figure 4: *An identity snowball attack.*

In step 3, the attacker repeats the process. The attacker scans newly compromised machines for TGTs, perhaps lying in wait for additional users to log in. The attacker then uses these new identities to compromise still more machines.

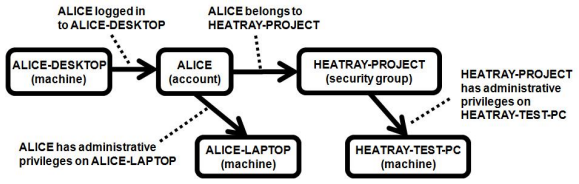

Figure 5: *An attack graph capturing the logins, security group memberships, and administrative privileges that allow the attacker to launch an identity snowball attack.*

3 Attack Graphs

An attack graph provides a convenient representation for how the security configuration in a network of machines may be vulnerable to attack. Attack graphs are a general formalism for capturing a wide variety of threats, but for our purposes it will suffice for the nodes to be machines, accounts, and security groups. Figure 5 depicts such an example attack graph. A directed edge in the graph means that the from-node can control (or "speak for" [29]) the to-node. For example, machine ALICE-DESKTOP can perform any action available to the account ALICE because ALICE logged in to ALICE-DESKTOP. The other edges represent other types of control relationships: ALICE can perform any action available to HEATRAY-PROJECT because she is a member of that security group; ALICE can perform any action available to ALICE-LAPTOP because she has administrative privileges on that machine; and similarly HEATRAY-PROJECT is a security group that has administrative privileges on HEATRAY-TEST-PC. The attack graphs analyzed by Heat-ray consist of exactly these four types of edges. As detailed in Sections 3.2 and 4, the login edges are collected over a brief (e.g., week-long) observation window, while the other edges are largely static across this same observation window.

Now consider an attacker who compromises the machine ALICE-DESKTOP. Looking at the attack graph, the identity snowball attack described in Section 2 corresponds to first traversing the edge from ALICE-DESKTOP to ALICE, and then traversing additional edges to arrive at ALICE-LAPTOP and HEATRAY-TEST-PC. This illustrates how the attack graph captures the identity snowball threat: if there is a path from a first node to a second node in the attack graph, then an attacker who compromises the first node can also compromise the second node.

Figure 5 also illustrates how changes to security configuration can prevent such attacks. Suppose that the account ALICE is removed from the security group HEATRAY-PROJECT, preventing an attacker who compromises ALICE-DESKTOP from continuing on to compromise HEATRAY-TEST-PC. In the attack graph, this change corresponds to removing the edge between ALICE and HEATRAY-PROJECT, and the attack graph representation then appropriately reflects that HEATRAY-TEST-PC is no longer reachable from ALICE-DESKTOP.

This correspondence between changes in security configuration and removing edges in the attack graph is what allows Heat-ray to apply combinatorial optimization and machine learning techniques. For example, the sparsest cut algorithm (Section 5) is applied to the attack graph to calculate how removing certain edges will decrease the number of nodes that can be reached from various starting nodes. Mapping this back to the security configuration modeled by the attack graph, each removed edge corresponds to some configuration change. Furthermore, decreasing the number of nodes reachable from various starting nodes means that an attacker who compromises a machine corresponding to an initial starting node will be unable to compromise many other machines. Similarly, machine learning (Section 6) is used to estimate the costs of removing various edges in the attack graph. Because edge removal corresponds to configuration change, the learning technique is actually estimating the implementability of the configuration changes.

3.1 Implementable Configuration Changes

Heat-ray is designed to uncover high impact security configuration changes that are easily implementable. As mentioned in the Introduction, we consider a change implementable if it does not interfere with users accomplishing their work. In the attack graph, these implementable changes correspond to edges that can be removed. By way of contrast, the edges in Figure 5 may all be necessary for Alice to perform her job. For example, ALICE may need administrative privileges on ALICE-LAPTOP in order to install or upgrade software. We worked with IT administrators in the studied organization and identitied several important classes of edges that can be easily removed:

- **Removing out-of-date privileges:** Over time, individuals in an organization change roles, e.g., moving from being the administrator for one set of servers to being the administrator for another set of servers. Sometimes, the privileges on the old set of servers are never revoked. The easy configuration change is to remove the out-of-date privileges.
- **Removing overly-large group privilege assignments:** Sometimes a machine may be considered one that "everyone needs to access." For example, consider a machine that will be used to present some new technology, where the presenter has not yet been decided. An IT administrator might proactively grant privileges on this machine to an ex-

isting large security group, and then forget to remove these privileges later. The easy configuration change is to remove the privileges if the presentation is over, or to create a smaller and more targeted security group if the presentation is still pending.

- **Preventing unnecessary logins with powerful accounts:** If an account has administrative privileges on a large number of machines, it may be important for it to retain all these privileges in case they are needed to deal with unforeseen circumstances. However, this powerful account should be careful about where it logs in, lest its powerful TGT be exposed on an insecure machine. Sometimes TGTs are exposed as part of accomplishing some task that would have been trivial to perform in a more secure manner. For example, an administrator who logs in to a machine to update a local security group on the machine could easily have modified the security configuration using only an ST (which would have been secure) using standard remote management tools. The easy configuration change is for the powerful account's owner to pick a small number of secure machines that he or she typically uses, and to modify the configuration at the KDC to forbid powerful account logins at other servers (mechanisms to securely enforce such a policy already exist [55]). For many administrators with powerful accounts, taking care with where they log in is already a known responsibility.
- **Securing automated script execution:** The task mentioned in the previous example, modifying local security group membership, is easy to perform using an ST. Many more elaborate tasks are encoded into scripts that need to run on a large number of target machines. These scripted tasks often are designed to execute as part of a login with a TGT on the target machine, and they also require administrative privileges (e.g., the scripted tasks upgrade software or audit security critical files). The most straightforward way of automating these tasks is to log in to every machine with a powerful account that has administrative privileges on all of them. However, this is highly insecure: if a single one of these machines is compromised, all the machines can be compromised using the powerful account.

  Fortunately, there is a secure approach to automating such tasks. The powerful account can create a temporary local account on each of the machines, grant each local account administrative privileges on its respective machine, log in to the remote machine using the temporary account, execute the scripted task, and then delete the temporary account, all with only an ST from the powerful account. Though slightly more involved than the other configuration changes, this successfully runs the tasks without ever exposing the powerful account's TGT on any of the other machines. We have verified that changing an existing script in this manner required less than 20 lines of new code, but the exact amount of work may be script-dependent.

Based on these insights, we formulated a simple set of guidelines for evaluating changes proposed by Heatray; we expect that similar guidelines will work for other organizations. Initially, we remove overly-large group privilege assignments, adding back in privileges only for accounts that actually logged in to the machines. We also secure (using the mechanism just described) all automated scripts that log in to many machines. After these steps, we judge accounts that still have administrative rights on many systems to be "powerful accounts" where it is reasonable to ask them to log in only to a single secure machine. Because it is not always clear from the attack graph whether an account login is due to an automated script, we use the heuristic that any account that logs in to more than 10 machines is doing so as part of a script. During our work, we also encountered a small number of cases where we could not apply these guidelines, e.g., a particular powerful account that had to log on to tens of machines.

These guidelines underscore the importance of combinatorial optimization and machine learning. The sheer amount of configuration in a large organization makes it difficult to sift through manually. Furthermore, "best practices" for security configuration are often either imprecise or unnecessarily burdensome. For example, limiting all accounts to log on to a few machines leaves open the interpretation of "few," while limiting all accounts to a single machine is highly restrictive and not actually necessary – we show in Section 8 that restricting logins for a relatively small number of powerful accounts can significantly reduce the number of machines that can be used to launch a large-scale identity snowball attack. Also, even trivial changes requires some investment of time, if for no other reason than to check with all the relevant parties before applying standard management tools [54]. Combinatorial optimization can automatically select a small number of configuration changes that offer a large improvement, minimizing the required investment of time.

Similarly, it is easy to understand from the context given in these examples that the changes are implementable. However, this context may be hard to represent explicitly without adding a significant burden to the IT administrator, e.g., keeping annotations describing every account, security group, and scripted task. Machine learning allows estimating change implementability without detailed annotations, thereby avoiding this

administrative burden.

3.2 Modeling Issues

Modeling security configuration using attack graphs necessarily involves modeling approximations. For example, any particular login occurs at some point in time, and the TGT will be destroyed after the user logs out. If the edge is represented as only existing during this time window, it can be used to estimate the rate at which an identity snowball attack can proceed. However, estimating the future benefit of preventing this login requires some prediction of whether the login will happen again.

Heat-ray treats this issue conservatively by discarding time of login in the attack graph. Intuitively, this is a worst-case assumption that people will repeatedly log in to the same machines as part of their work, and so any machine compromised after a user has logged out will eventually see that same user's TGT in the future. By reducing the threat of identity snowball attacks under this worst-case assumption, Heat-ray guarantees that it has also improved the actual case, where not all logins repeat. A more elaborate predictive model of future login behavior might yield a greater reduction in the identity snowball threat for any given amount of effort changing security configuration. However, in Section 8 we show that identity snowball attacks would be a significant threat even under a best-case assumption that logins never repeat outside the time window we evaluated.

In practice, some potential configuration changes include both edge addition and removal. For example, a desirable configuration change might modify a machine to remove the administrative privileges of a security group containing many accounts, but then add back in administrative privileges for a small number of the accounts. To deal with this, Heat-ray focuses on selecting edges to propose for removal, but then allows the IT administrator to add back in edges as part of approving the proposed edge removals.

4 Collecting Data

Heat-ray is a system for analysis, and it relies on an external database to provide the needed data about security configuration. In our implementation, this external database provides the complete security configuration of the organization. This may contain attack paths that an attacker would not discover using the techniques described in Section 2, and so the database should itself be secured in a manner similar to the data collection systems that provide the data about security configuration. However, the IT administrator cannot rely on attack paths being hard to discover (this would be "security through obscurity"), and so must defend all the potential attack paths.

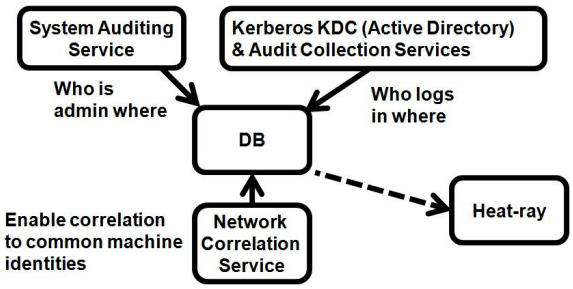

Figure 6: *How Heat-ray integrates with a data collection system.*

The external database is itself supplied by three data collection systems, as shown in Figure 6. These systems were developed in-house by the studied organization, but we note that commercial security products also help collect this information [18].

The first source of data is "Audit Collection Services." Audit Collection Services can collect event information from the Kerberos KDC implemented by the Active Directory application. The event information includes all grants of TGTs and STs over a given period of time. The TGT and ST events include the IP adress of the machine, the name of the account for which the ticket was granted, and the time at which the grant was made.

The second source of data is the "System Auditing Service." This is a tool that scans all machines within the organization and reports back the accounts and security groups that have administrative privileges on each machine. These logs also contain the MAC address, machine name, and the time at which the scan was done.

The third source of data is the "Network Correlation Service." This service collects Address Resolution Protocol (ARP) logs [17]. The ARP logs allow the IP addresses in the Audit Collection Services logs to be correlated with the MAC addresses and machine names in the System Auditing Service logs.

The components in the current data collection system collectively represent multiple man-years of software engineering effort, and they have been through significant validation to assure the accuracy of the data being collected. However, there are several places where inaccurate inferences might arise in the current data collection architecture: the correlation from MAC address to IP address may be incorrect if the common network time service being used is failing to synchronize; the administrative privilege assignments on each machine are polled periodically, and so may not always reflect recent additions or deletions; the enumeration of accounts in a security group is done when the data is inserted into the

database, and it too may change over time; individual machines may have multiple MAC addresses, possibly causing them to be described ambiguously by the data; and finally, if a computer is compromised, it may spoof its MAC address, or it may provide incorrect information about which accounts have administrative privileges on the compromised machine.

Some of these problems are artifacts of the data collection system that Heat-ray currently leverages. For example, correlating IP and MAC addresses using the Network Correlation Service could be entirely avoided if the Audit Collection Service additionally recorded MAC addresses. The problem of missing changes to various security groups could be solved by recording changes at Active Directory, and then feeding these changes into the database. Fortunately, these issues have not been a problem for the current system, and indeed, this is not surprising: network time synchronization within a single organization is generally quite accurate; security group membership changes rarely; and client machines tend to be connected to the network using only one MAC address at a time (e.g., most laptops we observed were either using a wired or a wireless interface, not both at the same time).

The problem of a compromised machine misrepresenting its security configuration is more fundamental, but this problem does not prevent Heat-ray from accomplishing its goal. Heat-ray is designed to prevent uncompromised machines from becoming compromised. Regardless of how a compromised machine reports its own configuration, Heat-ray will still try to remove edges that point from the compromised machine to accounts (i.e., the edges due to logins). These logins are reported directly by the KDC through the Audit Collection Service when it grants TGTs, and thus this data is not exposed to tampering by the compromised machine.

The work we have done with IT administrators has also served as an end-to-end validation for part of the input data. For the edges that the IT administrators removed, we have independent confirmation that the edge did exist, and hence that this part of the data was correct.

5 Applying Sparsest Cut

Heat-ray is designed to scale to the massive attack graphs that are needed to model real-world large organizations. Such graphs can have hundreds of thousands of nodes and millions of edges. To find the high impact edge removals in these graphs, Heat-ray models the problem as an instance of *sparsest cut*, a well-studied problem in combinatorial optimization. Sparsest cut finds the smallest set of edges in the attack graph whose removal splits the graph into two large components. This is visually depicted in Figure 7. On each iteration, Heat-ray strives to identify such edge sets to remove, thereby splitting the graph into many small components.

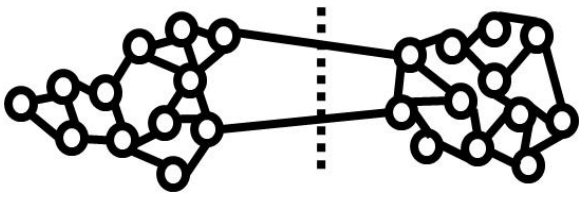

Figure 7: *A sparse cut is a small set of edges whose removal separates the graph into two large components.*

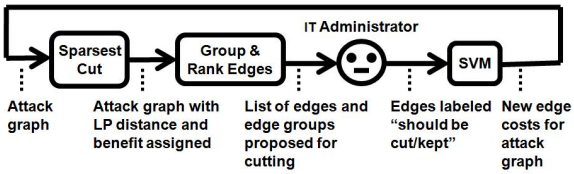

Figure 8: *Heat-ray algorithmic workflow.*

The overall role of the sparsest cut algorithm in Heat-ray is shown in Figure 8. The sparsest cut algorithm approximates every edges' distance and benefit, terms we define later in this section. These calculations are used to rank and group the edges for presentation to an IT administrator. Based on the IT administrator's feedback, the edge weights in the attack graph are updated using a machine learning algorithm (SVM) before the next iteration of sparsest cut.

The integer programming version of sparsest cut is NP-hard, but a large body of work in the theoretical computer science community has looked at how to efficiently compute approximate solutions to the linear relaxation of this problem. We discuss this related work in more detail in Section 9. Heat-ray only solves the linear relaxation, and it takes an approach similar to Young's algorithm [59]. The remainder of this section describes how Heat-ray exploits its particular problem domain to deviate from Young's algorithm, and for the sake of brevity it assumes the reader is already familiar with Young's algorithm and some other work on sparsest cut. The paper following this section can be read without the details presented in the remainder of this section.

To define the directed sparsest cut problem precisely, we use the notation of Hajiaghayi and Räcke [33] as shown in Figure 9. The inputs are the vertex set V, the edge set E, the edge costs $c(e)$, and the demand set $\text{dem}(i)$. In the Heat-ray context, there is a unit demand between every pair of machines (s_i, t_i). P_{uv} refers to the set of paths connecting u to v. The output of the optimization is a set of edge distances $d(e)$. In the NP-hard integer programming version, an edge distance of 1 indicates that the edge should be cut, and an edge distance of 0 indicates that the edge should not be cut. In the lin-

ear relaxation, these edge distances $d(e)$ are allowed to take on fractional values between 0 and 1, and the $x(u, v)$ are forced by the optimization and equation 3 to take on the shortest path distance between u and v. Equation 4 constrains all distances to be positive. Equation 2 represents a normalization for the relative sizes of the separated portions of the graph – a feasible solution either separates many pairs (s_i, t_i) by a small distance $x(s_i, t_i)$ or it separates a few pairs by a large distance. The edge costs are all initially set to 1, and they are updated in subsequent iterations as described in Section 6. Young's algorithm solves this linear program by iteratively finding shortest paths between vertices and increasing the edge distances $d(e)$ on these paths by an amount proportional to the edge costs.

Heat-ray has to deviate from Young's algorithm because it needs to be faster: Heat-ray is designed for interactive use, and thus it needs to complete within minutes even on a graph with millions of edges. Fortunately, Heat-ray has significantly more flexibility in its goal than Young's algorithm: Heat-ray does not need to estimate the actual optimal value of the objective function, but can settle for estimating the relative importance of edges to the objective function. For example, edges that cause many accounts to be able to compromise many machines are very bad. As long as these edges are identified as important to cut, the exact value of the objective function is not important, allowing Heat-ray to compute a looser approximation than Young's algorithm.

Heat-ray exploits this additional flexibility in two ways, which we initially describe at a high level. First, Heat-ray applies stochastic gradient descent rather than classic gradient descent. This allows the use of sampling, dramatically reducing the time to compute the gradient at some cost in accuracy. Second, Heat-ray uses a change of variables to eliminate the inequality constraints present in Young's formulation, which in turn eliminates constraints on the step size in gradient descent. This makes the problem non-linear, which makes the convergence guarantees more complicated. However, Heat-ray's reduced need for accuracy means it does not need to run to convergence, and so avoiding expensive calculations around step size is worthwhile.

We perform the aforementioned change of variables in two steps. First, we make the problem non-linear by moving the constraint of equation 2 into the objective function. The revised formulation is shown in Figure 10. Note that this re-formulation has not changed the underlying model: an optimal solution to the re-formulated problem can be trivially converted to an optimal solution to the original problem through scaling, and vice versa. Second, we substitute in expressions using new variables $u(e)$ for the original variables $d(e)$. The expressions are given in equation 5. These new variables $u(e)$ are left unconstrained, but because of the expressions used in the substitution, the non-negativity constraints on $d(e)$ can be dropped.

$$
\begin{array}{lll}
\text{minimize} & \Sigma_{e \in E} c(e) d(e) & (1) \\
\text{subject to} & \Sigma_i x(s_i, t_i) \cdot \text{dem}(i) = 1 & (2) \\
& \forall (u, v) \in V \times V, \forall p_{uv} \in P_{uv} : & \\
& \Sigma_{e \in p_{uv}} d(e) \geq x(u, v) & (3) \\
& d(e) \geq 0, x(u, v) \geq 0 & (4)
\end{array}
$$

Figure 9: *Hajiaghayi-Räcke sparsest cut formulation.*

$$
\begin{array}{lll}
\text{minimize} & \ln \frac{\beta}{\gamma} & (6) \\
& \beta = \Sigma_{e \in E} c(e) d(e) & (7) \\
& \gamma = \Sigma_i x(s_i, t_i) \cdot \text{dem}(i) & (8) \\
\text{subject to} & \forall (u, v) \in V \times V, \forall p_{uv} \in P_{uv} : & \\
& \Sigma_{e \in p_{uv}} d(e) \geq x(u, v) & (9) \\
& d(e) \geq 0, x(u, v) \geq 0 & (10)
\end{array}
$$

Figure 10: *Sparsest cut formulation used by Heat-ray.*

$$
d(e) = \begin{cases} u(e) & \text{if } u(e) > 1 \\ e^{u(e)-1} & \text{if } u(e) \leq 1 \end{cases} \quad (5)
$$

Heat-ray deals with the constraints of equations 3 and 9 in the same way as Young's algorithm. Rather than represent the $x(u, v)$ directly, Heat-ray simply computes shortest paths on the variables $d(e)$ (now expressions involving $u(e)$ because of our change of variables).

As in Young's algorithm, Heat-ray begins with an initial uniform assignment of edge distances, and then iteratively refines this assignment using steps of gradient descent. To estimate the gradient using sampling, Heat-ray chooses a small number of nodes, and from each node it conducts a bounded-horizon search both forwards and backwards using Dijkstra's algorithm [14]. Each tree constructed by Dijkstra's algorithm is interpreted as a set of shortest paths for the purpose of estimating the gradient. In particular, the shortest path yields the distance $x(s_i, t_i)$ that contributes to the numerator of the objective function. The number of shortest paths that each edge e appears on yields the contribution to the denominator of the objective function, and can be interpreted as the benefit $b(e)$ assigned by the linear program to cutting this edge:

$$b(e) = \#\text{ shortest paths crossing } e.$$

To avoid degeneracies when there are multiple shortest

paths with the same distance between two given nodes, we perturb each edge distance by a small random multiplicative factor (between 0.95 and 1.05) before each shortest path computation.

The algorithm used by Heat-ray has a number of parameters, and in future work we hope to explore the tradeoffs among these parameters more thoroughly. However, we found that the following parameter values suffice for our purposes: a gradient descent step size of 1.0, a bound of 1 on the number of iterations, a bound of 1,000 on the size of each shortest-path tree, and a bound of 1,000 on the number of shortest-path trees. With these parameters, Heat-ray's algorithm for sparsest cut completes in just under a minute on a dual-processor AMD Opteron server with 10 GB of RAM. In contrast, running just Heat-ray's implementation of shortest path in the configuration required by Young's algorithm (i.e., without sampling 1,000 starting nodes and 1,000 shortest path trees) would require over 4 orders of magnitude more running time, simply because many more nodes have to be visited in the graph.

6 Learning Edge Costs

As mentioned in Section 3, the implementability of removing different edges in the attack graph can vary dramatically. The sparsest cut algorithm can incorporate such differences as different edge costs, but the number of edges in the graph makes it infeasible to set all their costs manually.

After considering heuristics for setting edge costs, we instead decided to use a machine learning algorithm derived from Support Vector Machines (SVM) to learn the costs based on feedback from the IT administrator about their willingness to make a security configuration change. After each application of sparsest cut, we present the administrator with a list of proposed edge cuts. The IT administrator can label each edge as "should be cut," "should be kept," or "no opinion." These decisions give us implicit feedback on the relative magnitude of the cost: If an edge is marked as worth keeping, then the cost must be larger than the benefit of cutting it, while if an edge is marked as worth cutting, its cost must be less than the benefit. The machine learning algorithm generalizes from the feedback on individual edges to re-estimate the costs of all edges on every iteration. In this way, Heat-ray learns over time to propose primarily configuration changes that the administrator is interested in implementing. This learning approach has the advantage that the recommendations are tailored to each organization, and no a priori assumptions are required about any given usage pattern being correlated with edge cost (e.g., whether or not a mostly unused administrative privilege must be kept around for unusual events). As in Section 5, the paper following this section can be read without the details presented in the remainder of this section.

Heat-ray defines a set of features on each edge from which it tries to learn the best approximation to the true edge costs. The set of features are 12 basic graph properties of each edge: the number of accounts, security groups and machines pointing in to the start node of the edge (3 features), the number of accounts, security groups and machines that the start node points to (3 more features), and the corresponding numbers for the end node of the edge (6 more features). Let f_e denote the feature vector for edge e. The edge cost $c(e)$ is modeled as a linear function of these features, i.e.,

$$c(e) = w^T f_e + w_0,$$

where w_0 and the vector w are the parameters to be learned.

Heat-ray uses the "cut/kept" edge labels to generate constraints that the cost function should satisfy with a certain margin. For every edge that is marked as "should be cut," Heat-ray creates a constraint that this edge's cost is less than the benefit assigned by the sparsest cut linear program for cutting this edge. For every edge that is marked as "should be kept," Heat-ray creates a constraint that this edge's cost is greater than the benefit assigned by the linear program to cutting this edge. Thus, the set of labeled edges translate into a set of linear constraints on the cost:

$$b(e) - c(e) \leq -1 \quad \text{if } e \text{ is to be kept,} \tag{11}$$
$$b(e) - c(e) \geq 1 \quad \text{if } e \text{ is to be cut.} \tag{12}$$

Heat-ray uses the linear SVM framework [11] to learn the parameters of the cost function. The constraints in Eqns. (11) and (12) can be translated into a classification problem as follows. Let y_e encode the label on edge e: $y_e = 1$ if the edge is to be cut, and $y_e = -1$ if it is to be kept. The problem of learning the cost function boils down to learning a separating hyperplane (specified by the normal vector $\mathbf{w} = [w\ w_0]$), such that the cost of each labeled edge falls on the correct side of the benefit $b(e)$. SVM approaches this problem via the large-margin principle, with the aim of finding not only a separating hyperplane but one that provides the largest margin (separation) between the positive and the negative classes. In cases where it is impossible to find a linear hyperplane to separate the data, the soft-margin version of SVM allows for some slack. This translates into the following optimization problem:

$$\min_{\mathbf{w}} \sum_e \max(0, 1 - y_e(b(e) - c(e))) + \lambda \mathbf{w}^T \mathbf{w}, \tag{13}$$

where the value of λ is chosen to minimize the 5-fold cross-validation error. The function $\max(0, 1 - y_e(b(e) -$

$c(e)))$ is called the hinge loss; it is zero if the cost estimate of edge e falls on the correct side of the benefit, otherwise it is equal to the error. However, the hinge loss function can be difficult to optimize. Instead, Heat-ray applies the scaled logistic loss, which is differentiable everywhere and closely approximates the hinge loss [56].

At each iteration, Heat-ray obtains more labeled edges from the IT administrator and incorporates them into the training set for SVM. The features and cost-benefit constraints for kept edges are updated in subsequent iterations, while those for cut edges are frozen at the values in the iteration they are cut since they henceforth disappear from the graph. The optimal value of $\mathbf{w}$ is found via gradient descent, and it is then used to re-estimate the cost of every edge before the next iteration of the linear program of Section 5.

7 Proposing Edge Groups To Cut

Early in the development of Heat-ray, we discovered that IT administrators commonly want to apply a single action to an entire group of edges with a common start or end node. For example, an IT administrator will sometimes want to remove a group's administrative permissions on all machines – this corresponds to removing every edge where this group is the start node. In another example, an IT administrator will sometimes want to require a given powerful account to only log in to one machine – this corresponds to removing all but one edge where a machine is the start node and the particular powerful account is the end node.

To make it easier to act on edge groups, Heat-ray presents administrators with two kinds of groupings: all outgoing edges of some start node or all incoming edges of some end node. Heat-ray also proposes specific edges. Edges and edge groups are then ranked by a function of their cost and benefit. In the case of edge groups, the cost and benefit are defined to be the sum of the costs and benefits of all the edges in the group. Intuitively, edges and edge groups should be ranked highly if they have very high benefit, or if they have modest benefit but low cost. The ranking function $\max\{b(e), -c(e)b(e)\}$ captures this intuition, and we found that it worked well in our experiments (indeed, it performed better than either the benefit $b(e)$ or the distance $d(e)$ alone).

Finally, we applied a filter to the edge groups based on our experience with IT administrators: IT administrators never want to remove every login from a particular machine without looking at the identity of the accounts logging in. Therefore, we do not propose cutting edge groups where the specified node is a machine, though we do allow these same edges to be cut as part of other edge groups or as individual edges.

The IT administrator should only accept or reject some modest number of configuration changes in each round. After making these decisions, the IT administrator should re-run the algorithm (a matter of a few minutes), so that the algorithm can incorporate the feedback from this round and produce new recommendations that better reflect the IT administrator's notion of implementability. Through a small amount of experimentation, we found that presenting 900 items at a time worked well: Heat-ray recommends many worthwhile configuration changes, yet the IT administrator can scroll through the entire set with ease. The 900 items consist of 300 incoming edge groups, 300 outgoing edge groups, and 300 individual edges.

Each edge group may correspond to a very large number of individual edges, and the learning algorithm benefits little from large numbers of similar examples. Because of this, we marked only a subset of the edge groups that were not cut as explicitly "should be kept" – most were marked as "no opinion" – and we sampled the individual edges in each group before using them as input to the learning algorithm. We labeled on average 10 groups of outgoing and incoming edges as "should be kept" on each iteration, and we sampled 3 individual edges for learning from all labeled edge groups. We found this to be sufficient for the learning algorithm.

8 Evaluation

We begin by presenting statistics about the data set that we used to evaluate Heat-ray, and quantifying the severity of the threat from identity snowball attacks in the status quo. Then in Section 8.1, we show that Heat-ray quickly and effectively identifies a set of high-impact and implementable configuration changes, and that these changes significantly reduce the threat of identity snowball attacks. In Section 8.2, we examine these configuration changes. In Section 8.3, we compare Heat-ray's algorithmic approach to potential heuristics for proposing configuration changes.

We evaluated Heat-ray on a data set from the database described in Section 4. This data set covers a period of 8.5 days, from 2:00:27 AM on Monday March 5th, 2007 to 1:28:40 PM on Tuesday March 13th, 2007. Table 1 shows the number of each type of entity that comprises a node or edge in the attack graph (as explained in Section 3). The set of unique logins is derived by filtering the total set of logins to disregard the time at which the login occured, only retaining logins that differ in either the machine or account involved.

Figure 11 depicts the number of each node after grouping by the number of other nodes it points to in the attack graph. The lesson of this figure is that all four different kinds of edges in the attack graph have a similar distribution. Figure 11(a) shows that most accounts log in to

Machines	197,631
Accounts	91,563
Security Groups	62,725
Total Nodes	351,919
Unique Logins	130,796
AccountIsAdminOnMachine	309,182
SecurityGroupIsAdminOnMachine	380,320
AccountIsMemberOfSecurityGroup	3,695,878
Total Edges	4,516,176

Table 1: *Number of each entity in Heat-ray evaluation data set.*

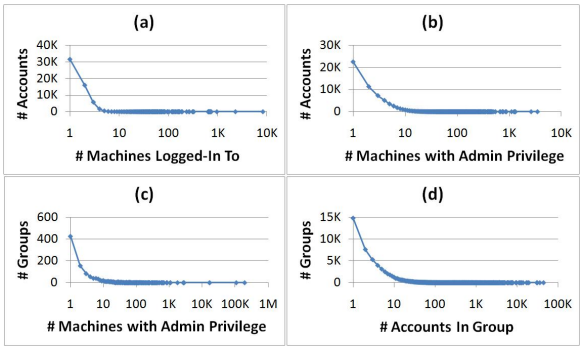

Figure 11: *Relative quantity of each node by number of other nodes it points to.*

a very small number of machines (approximately 30,000 log in to only one machine), but a small number of accounts log in to a very large number of machines. Figure 11(b) shows that most accounts have administrative privileges on a small number of machines, but a small number of accounts have administrative privileges on a large number of machines. Figure 11(c) shows a similar phenomenon for the administrative privileges of security groups. Figure 11(d) shows that most security groups are small, but a few security groups are very large.

This distribution suggests that a small number of accounts, security groups or machines may play a large role in the exposure of an organization to identity snowball attacks. For example, granting administrative privileges on a machine to a large group poses a far greater risk than granting administrative privileges to a small group. In such cases, Heat-ray may have the opportunity to propose a small number of configuration changes that produce a large reduction in the exposure of the organization to identity snowball attacks. The results we present in Section 8.2 show that this is indeed the case.

We now justify the statement in Section 3.2 that identity snowball attacks pose an acute threat even under the best-case assumption that no login we observe in our measurement window ever reoccurs. We randomly sampled 100 machines from the set of machines where at least one user logs in over the course of our observation period. From each machine, we calculated the number of machines reached using an identity snowball attack as a function of time. We assumed that TGTs are destroyed immediately after login, minimizing the attacker's window of opportunity.

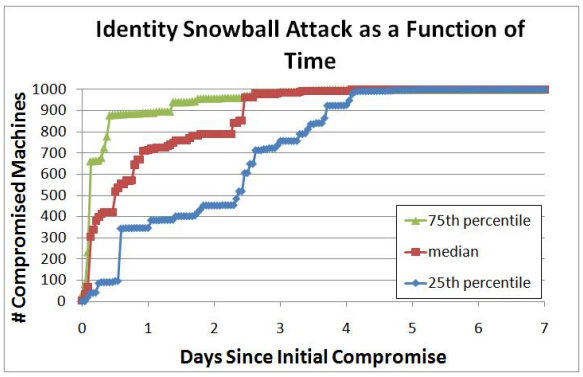

Figure 12: *Compromises from a random starting node as a function of time under the assumption that TGTs are quickly destroyed after login.*

Figure 12 shows the results of this experiment. The curves show the 25th percentile, median and 75th percentile for the number of machines reached via an identity snowball attack under this defender-favorable assumption. In the median curve, the attacker can compromise over 700 machines within 1 day, and over 1,000 machines within 4 days. The 75th and 25th percentile curves show similarly rapid rates of compromise. For reasons of confidentiality, we cap the number of reachable machines we report at 1,000. These results demonstrate the threat of identity snowball attacks even under best-case assumptions about TGT lifetime for the defender.

8.1 Heat-ray Effectiveness

Heat-ray is designed to be run periodically on an organizations's security configuration (e.g., to review configuration changes made in the past week). However, our evaluation focuses on the initial use of Heat-ray to reduce the threat of identity snowball attacks in an organization that has not previously been running Heat-ray.

Figure 13 shows the effectiveness of running Heat-ray for ten iterations. To produce each curve, we randomly sampled 1,000 machines from the set of machines where at least one user logs in over the course of our observation period, and we used these as points of initial compromise. We resampled the set of initial compromises at each iteration. We then calculated the number of machines that could be reached by launching an identity

snowball attack from each different initial compromise. For confidentiality reasons, we do not display the exact number of machines that could be compromised when that number is in excess of 1,000.

The curve labelled "Original" shows the graph before running Heat-ray. For 981 out of the 1,000 sampled machines, it is possible to launch an identity snowball attack that results in the compromise of over 1,000 machines. Applying the configuration changes identified in the first iteration of Heat-ray reduces this to only 453 out of the 1,000 sampled machines being able to launch an identity snowball attack that reaches over 1,000 other machines. The second iteration shows a similar sharp drop. Progress slows in the subsequent iterations, but by the tenth iteration, only 41 out of the 1,000 machines (4.1%) can be used to launch an identity snowball attack that compromises over 1,000 machines. The reduction from 981 to 41 is a decrease of 96%.

After 10 iterations of Heat-ray, there still remains a small set of machines that can be used to launch a large-scale identity snowball attack. Though we would like to drive the number of such machines to zero, it is not clear whether this is feasible: accounts with important administrative responsibilities in an organization must continue to log in somewhere.

The number of initial compromises that can lead to a large number of other compromises is only one of many reasonable metrics. The right metric may vary for each organization: an organization may want to focus on the number of initial compromises that can threaten even a small number of other machines (e.g., 50), or it may want to minimize some weighted sum of the number of machines threatened by each initial compromise. Figure 13 shows that Heat-ray reduces the number of machines reachable from most initial compromises, and hence improves a wide range of reasonable metrics, including the examples given above.

We now turn to the effort required from the IT administrator over these ten iterations. Heat-ray proposed a total of 9,000 edges and edge groups over these ten iterations. These proposals are drawn from a total of 308,576 potential edge groups, and millions of potential edges. Out of the 9,000 proposals, 1,745 were implementable changes under the criteria given in Section 3.1, and were thus marked as "should be cut." This is a manageable number of configuration changes for an organization with almost a hundred thousand accounts. All ten iterations together took less than three hours on a dual-processor AMD Opteron 250 with 10 GB of RAM. Less than half an hour was spent waiting on Heat-ray to make proposals; the rest of the time was spent examining the proposals. This demonstrates that Heat-ray meets its goal of allowing an IT administrator to identify desirable security configuration changes in a large organization with

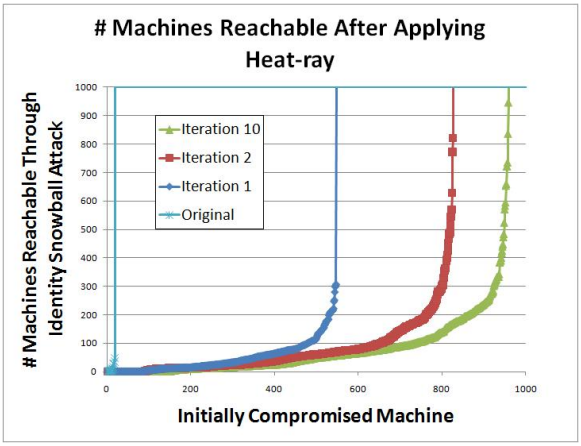

Figure 13: *Graph showing threat reduction after running Heat-ray for ten iterations, measured on 1,000 randomly selected machines as points of initial compromise. A point (x,y) indicates that, out of the 1,000 initial compromises, x of them could reach y or fewer machines using an identity snowball attack. Further to the right is better; the "Original" curve is on the far left of the graph.*

modest effort. The IT administrators can then apply standard management technologies [54] to implement these changes.

For comparison purposes, we also ran a version of Heat-ray with the machine learning aspect disabled, instead consistently setting edges to have unit cost – we refer to this approach as UnitCost. We evaluated the difference in threat reduction between the two approaches using the same methodology as used to produce Figure 13. We found that UnitCost almost always produced a smaller reduction in the number of machines that can be used to launch an identity snowball attack reaching over 1,000 machines. However, the benefit of machine learning was very uneven. In the first 2 iterations, the edges that need to be identified are quite obvious to both algorithms, and UnitCost averaged only 2.6% more such launching point machines compared to Heat-ray. In the next 6 iterations, the edges became less obvious and UnitCost started to do much worse: UnitCost averaged 21.4% more launching point machines than Heat-ray, and peaked at 27.1% more such machines. In the last 2 iterations, the total number of remaining implementable configuration changes decreased. Because there were so few changes for Heat-ray to find, UnitCost started to catch up with Heat-ray: it averaged only 7% more launching point machines in the final 2 iterations. From this experiment, we conclude that on the whole, machine learning does indeed allow Heat-ray to do better at proposing configuration changes that can actually be implemented, though when the number of im-

plementable changes becomes sufficiently small, the advantage of machine learning decreases.

Examining the performance of the SVM cost learner in more detail, we find that, averaged across the ten iterations of Heat-ray, its misclassification rate on the training sets is 4.05% and on the testing sets 19.7%. (An edge is misclassified if the learned cost falls on the wrong side of the benefit, e.g., if the cost of an edge labeled as "cut" is greater than the benefit.) The low training error rate shows that SVM successfully fits the training data. A significant reason for the high testing misclassification rate lies in the sampling procedure for creating the training and testing data sets. As discussed in Section 7, the training set contains 3 individual edges selected from each edge group, regardless of group size. The testing set contains the rest of the labeled edges. As a result, the training set encourages the SVM to prioritize explaining the numerous small edge groups (typically account logins) over the smaller number of large edge groups (typically the administrative privileges of security groups). This manifests itself in the larger testing error.

The patterns learned by SVM change with the feedback given at each iteration. It consistently assigns positive weights to the machine in-degree and the group out-degree of the starting node, and negative weights to the group in-degree of the starting node. It sometimes assigns negative weights to the group in-degree and out-degree of the destination node. The positive weights drive up the cost of removing the administrative privileges or group memberships of individual accounts, whereas the negative weights drive down the cost of removing account logins or the administrative privileges of groups.

8.2 Examination of Changes

The configuration changes identified by Heat-ray on this data set all fall in to the categories described in Section 3.1. Table 2 shows the number of each type of configuration change that was approved on each iteration. The dramatic initial reduction in the identity snowball threat in the first Heat-ray iteration results from changes that fit the "removing overly-large group privilege assignments" category: some machines were granting administrative privileges to security groups containing thousands of accounts. We continued to find group admin privileges to remove in later iterations, though their impact was smaller.

Both the first and second iterations identified a large number of changes from the "securing automated script execution" category. Heat-ray found that a small number of scripted tasks were being carried out through logins by highly privileged accounts; these were marked for conversion to use the mechanism described in Section 3.1.

Heat-ray iteration	Removing large group admin privs	Securing scripts	Preventing unnecessary logins
1	9	69	0
2	63	55	0
3	36	29	228
4	14	24	223
5	5	15	231
6	5	11	235
7	9	8	224
8	1	5	241
9	17	7	228
10	1	5	219

Table 2: *Configuration change statistics by iteration.*

As in the category of group administrative privilege removals, later iterations continued to yield changes in this category but with diminishing importance.

In the third iteration and later, most security configuration changes identified by Heat-ray are from the "preventing unnecessary logins with powerful accounts" category. We found that applying simple policies, such as requiring these powerful administrators to log in only to their own desktops or laptops, was sufficient to eliminate large numbers of potential identity snowball attacks.

8.3 Comparison to Heuristics

In this section, we compare Heat-ray's algorithms to a set of heuristics for enterprise security policy. Note that while these heuristics are specific to stopping identity snowball attacks in enterprise networks, Heat-ray's combinatorial optimization and machine learning algorithms are applicable to arbitrary attack graphs where the graph's edges have arbitrary feature sets.

For this comparison, we design the heuristics to focus on the edge classes identified in Section 8.2. To make the comparison fair, we limit each heuristic to proposing the same number of edges or edge groups as proposed by Heat-ray; this allows the heuristic to use an equal amount of the IT administrator's time. We evaluate the heuristics using the same methodology as used to produce Figure 13, where we found that after 10 Heat-ray iterations, only 41 out of 1,000 randomly selected potential initial compromises would still allow an identity snowball attack to reach over 1,000 other machines. Previewing our results, we find that 10 Heat-ray iterations significantly out-performs the heuristics.

The first heuristic we consider is to identify the security groups that contain the largest numbers of accounts. This heuristic is based on the rule of thumb that large security groups should not have administrative privileges on any machine. Because Heat-ray proposed 300 out-

going edge groups in each of its 10 iterations, we allow this first heuristic to propose 3,000 security groups that should no longer have administratrative privileges on any machine. After running the heuristic, we find that it fails to identify many edge groups to cut for the simple reason that very few large security groups (only 2.4% of the top 3,000) have administrative privileges anywhere.

The second heuristic we consider is designed to address the shortcoming of the first heuristic. The second heuristic ranks security groups by the product of their in- and out-degrees in the attack graph (i.e., the number of accounts that belong to the group times the number of machines where the group has administrative privileges). We find that this does identify 167 security groups where administrative privileges can be removed, more than the 160 identified by Heat-ray, and we proceed to remove the privileges of every one of these 167 groups. After doing this, we find that out of 1,000 potential initial compromises, over 503 would still allow an identity snowball attack to reach over 1,000 other machines.

The third heuristic we consider focuses on reducing the number of machines where powerful accounts log in. Based on our experience with the second heuristic, we rank accounts by the product of the number of machines where the account logs in and the number of machines where the account has administrative privileges. We then restrict every possible account (including ones not originally considered by Heat-ray) to only log in to one machine. We find that out of 1,000 potential initial compromises, over 776 would still allow an identity snowball attack to reach over 1,000 other machines.

Finally, we evaluate combining the second and third heuristics, allowing 3,000 proposals to remove the administrative privileges of a security group and 3,000 proposals to reduce the number of machines where an account logs in. Even after running this combined heuristic, over 240 out of 1,000 potential initial compromises can still compromise over 1,000 other machines. Thus, all the heuristics we have considered are significantly inferior to 10 Heat-ray iterations. This also suggests that considering the larger impact of a configuration change on the network, as Heat-ray does by using sparsest cut, provides a significant benefit compared to heuristics based on simply considering local properties of a configuration change.

9 Related Work

Much research has focused on preventing anonymous machine compromise (i.e., a compromise launched without a Kerberos ST or any other form of authentication) [5, 13, 19, 12, 36]. Other work has targeted identifying compromises and their subsequent effects once they have occurred [24, 26]. Most intrusion prevention and detection systems (IPS/IDS) fall into these categories. Heat-ray complements this work by containing the damage of any individual compromise that still does occur without relying on being able to detect the compromise.

Singer [49] describes an incident in 2004 where a semi-automated attack with an identity snowball component successfully exploited machines across a number of sites. Schechter et al [43] analyze the feasibility of fully automating this attack and several methods for decreasing the propagation rate by obscuring the addresses of target hosts. Heat-ray differs from the work of Schechter et al in its focus on limiting the propagation of a compromise by proposing implementable changes to security configuration, not by attempting to obscure the set of target hosts.

In the rest of this section, we compare Heat-ray with other closely related prior work grouped by the major techniques in Heat-ray: attack graphs (Section 9.1), combinatorial optimization (Section 9.2) and machine learning (Section 9.3). In Section 9.4, we discuss work on alternative approaches to authentication and authorization.

9.1 Attack Graphs and Analysis

Attack graphs are a very general technique for modelling security in a network of machines. They have been used to model both local and remote elevation of privilege attacks due to software vulnerabilities, insecure Access Control Lists (ACLs), insecure network firewall rules and other issues [2, 52, 48, 38, 35]. Prior work has looked at automating the construction of attack graphs, sophisticated modeling of network features, and graph analysis using both algorithmic and visualization approaches. Heat-ray models only the features necessary for identity snowball attacks, and its algorithmic approach differs from prior algorithmic approaches in two main ways.

First, the algorithmic techniques in prior work have assumed that administrators can wade through every potential configuration change, either assigning costs to the changes a priori, or if they do not assign costs, rejecting a large number of proposed changes that are too high in cost (i.e., infeasible). Heat-ray uses machine learning to estimate the feasibility of configuration changes, thereby avoiding both of these burdens: the administrator does not have to assign costs manually, but because the proposed changes reflect the estimated costs, infeasible changes are proposed less often. This significantly reduces the burden on IT administrators and allows Heat-ray to be applied more easily in a large organization (i.e., one with hundreds of thousands of users and machines).

Second, the algorithmic techniques in prior work generally assume a well-defined set of high-value machines that must be protected, and then use techniques such as

model checking, approximate shortest paths, or the DataLog reasoning engine to find particular attacks from a low-value machine to a high-value machine. Mulval [38] allows an arbitrary DataLog policy to be specified, but the only policy examples they provide are for protecting particular high-value machines or files.

In a large organization, it is not enough to protect particular high-value machines. It is also unacceptable for a large number of "low-value" desktops to be compromised, and identity snowball attacks pose exactly this threat. Because Heat-ray cannot focus on just protecting a small number of high-value machines, none of the algorithmic techniques from prior work are directly applicable. Instead, Heat-ray uses a new algorithm based on sparsest cut.

Visualization is a powerful technique for understanding data, but it has been difficult to apply to large attack graphs. A recent paper on improving attack graph visualization only demonstrated scaling to 16 machines [37]. Though techniques have been developed for visualizing massive graphs [34], no prior work has evaluated whether these techniques can succeed at illuminating the small sets of high impact and implementable configuration changes needed by the IT administrator.

Prior work on attack graphs has not specifically focused on identity snowball attacks, and to the best of our knowledge, our work is the first to measure the severity of this threat in a large organization. Though this analysis could have been done in frameworks proposed by earlier work, Heat-ray's more directed focus made this measurement easier. For example, compared to prior work that examined the ACLs on every file system and registry object [35], Heat-ray needed to collect far less data per machine. This allowed Heat-ray to more easily scale to the hundreds of thousands of machines and users in the large organization we studied. Nonetheless, it is an interesting area of future work to understand whether Heat-ray can help with the classes of attack graphs considered in prior work or the even larger attack graphs that arise from including additional attack vectors, such as local elevation of privilege exploits.

9.2 Combinatorial Optimization

Heat-ray's sparsest cut algorithm leverages prior work on this problem. Recent work has made significant progress in the sparsest cut approximation ratio [4, 1, 33]. There is also a large body of work on efficiently computing these approximations [47, 31, 27, 3]. Heat-ray borrows most directly from Young's algorithm [59], but Heat-ray exploits its greater flexibility around approximating the objective function. Section 5 explains in detail how Heat-ray exploits this flexibility.

Heat-ray uses random sampling to estimate the gradient in its sparsest cut algorithm. Prior work has sometimes referred to this general approach as stochastic approximation, stochastic optimization, or stochastic gradient descent [22, 6]. Though we are not aware of any prior work using Heat-ray's sampling strategy, Heat-ray's primary contribution is its overall technique for managing security configuration, not the particularities of its sparsest cut algorithm.

9.3 Machine Learning

SVM [11] is a widely-used machine learning algorithm for classification and regression. The cost learning algorithm we present in Section 6 is subtly different from the usual application of SVM to classification: rather than learning a linear function that is bounded above +1 or below -1, we learn a linear function that is greater or less than the benefit assigned by the linear program by some margin. SVM with varying offsets has also been applied to learn ranking functions [20, 10], solving for both relative constraints (e.g., item 1 should be ranked above item 2) and for optimal ranking boundaries (in our setting, this would mean estimating $b(e)$). In contrast, our algorithm operates over absolute constraints (e.g., cost of edge 1 should be higher than benefit of edge 1, a fixed number).

9.4 Alternative Approaches to Authentication

A large amount of research has focused on the development of alternative authentication and authorization technologies. Much research has focused on decentralized systems, such as GSI [7], SFS [25] and SDSI/SPKI [42, 16]. Applying Heat-ray to such systems would require centralizing the data needed to create the attack graphs, a task made harder by these systems' decentralized nature.

One area that has seen significant adoption is multi-factor authentication, e.g., combining smartcards or biometrics with passwords [41, 21]. These techniques effectively prevent password stealing, but they do not prevent hijacking of a Kerberos TGT by a compromised system. As mentioned in Section 2, modern identity systems entrust computers to perform actions on behalf of a user. Heat-ray is designed to prevent this trust from being abused if a computer is compromised.

Other areas of active research in authentication and authorization include restricted delegation, multi-principal systems, and information flow security. In restricted delegation, a user Alice can empower another user Bob to perform only particular actions on her behalf (as opposed to classic Kerberos delegation, which is more analogous to a blank check) [29]. In a multi-principal system, ACLs can incorporate more than just a user's identity,

e.g., restricting a file's access to Alice, and further requiring Alice to access the file only through a particular program [58]. Information flow security enables access checks based on the flow of information between processes or other entities [28, 60, 9].

All these systems increase the information available for security decisions, e.g., by taking more than just the user's identity into account. Compared to Heat-ray, these systems require significantly more effort to deploy; Heat-ray works on the security configuration of existing systems. Furthermore, even if these alternative systems were widely deployed, it seems unlikely that administrators would perfectly configure system security. In turn, Heat-ray could potentially be used in these alternative systems to identify the security configuration changes that were both high impact and implementable.

10 Conclusion

Computers are becoming ever more interconnected, both in the enterprise and in the emerging world of cloud computing. This paper has focused on analyzing Kerberos and Windows, but the importance of interconnection is common to other identity systems and other operating systems. This suggests that identity snowball attacks will pose an ever more acute threat unless we take defensive measures.

Heat-ray is the first system to defend against identity snowball attacks in large organizations. We have measured the threat of such attacks in a single large organization, and demonstrated the effectiveness of Heat-ray in addressing this threat. Heat-ray accomplishes this goal by applying novel machine learning and combinatorial optimization techniques to attack graphs. We are optimistic that the techniques introduced by Heat-ray can be applied to other types of attack graphs, further enhancing our ability to secure distributed systems.

11 Acknowledgments and Responsible Disclosure

We have many people at Microsoft to thank for their support of this project, but we would particularly like to thank Eric Fitzgerald, Ted Hardy, Cam McLeery, Dave Steeves, and Greg Hartrell. We would also like to thank the anonymous reviewers and our shepherd, Stefan Savage, for their helpful feedback on the paper.

We have also greatly enjoyed working with the Microsoft Forefront team on the problem of identity snowball attacks. As of July 2009, the "Stirling" version of Microsoft's Forefront Security Suite is available as a public beta, and this beta version automates collecting security configuration relevant to Heat-ray.

The organization studied in this work was extremely helpful in working with us to analyze their security configuration and fix problems as they were identified. We express our heartfelt gratitude.

References

[1] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. O ($\sqrt{\log n}$) Approximation Algorithms for Min UnCut, Min 2CNF Deletion, and Directed Cut Problems. In *STOC*, 2005.

[2] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, Graph-Based Network Vulnerability Analysis. In *ACSAC*, 2002.

[3] S. Arora, E. Hazan, and S. Kale. $O(\sqrt{\log n})$ Approximation to SPARSEST CUT in $\tilde{O}(n^2)$ Time. In *FOCS*, 2004.

[4] S. Arora, S. Rao, and U. Vazirani. Expander Flows, Geometric Embeddings and Graph Partitioning. In *STOC*, 2004.

[5] S. Bhatkar, D. DuVarney, and R. Sekar. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *USENIX Security*, 2003.

[6] O. Bousquet, U. von Luxburg, and G. Rtsch. *Advanced Lectures on Machine Learning*. Springer, 2003.

[7] R. Butler, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, and C. Kesselman. National-Scale Authentication Infrastructure. *COMPUTER*, 33(12):60–66, 2000.

[8] S. Chen, J. Dunagan, C. Verbowski, and Y. Wang. A Black-Box Tracing Technique to Identify Causes of Least-Privilege Incompatibilities. In *NDSS*, 2005.

[9] S. Chong, K. Vikram, and A. Myers. SIF: Enforcing Confidentiality and Integrity in Web Applications. In *USENIX Security*, 2007.

[10] W. Chu and S. S. Keerthi. Support vector ordinal regression. *Neural Computation*, 19(3):792–815, 2007.

[11] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20, 1995.

[12] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-End Containment of Internet Worms. In *SOSP*, 2005.

[13] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *USENIX Security*, 1998.

[14] E. Dijkstra. A Note on Two Problems in Connexion with Graph Theory. *Numerische Mathematik*, 1(269-271):1, 1959.

[15] EC2. http://aws.amazon.com/ec2.

[16] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory, 1999.

[17] Ethernet Address Resolution Protocol. http://tools.ietf.org/html/rfc826.

[18] Forefront. http://www.microsoft.com/forefront.

[19] C. Grier, S. Tang, and S. King. Secure Web Browsing with the OP Web Browser. In *IEEE Security and Privacy*, 2008.

[20] R. Herbrich, T. Graepel, and K. Obermayer. Support Vector Learning for Ordinal Regression. In *International Conference on Artificial Neural Networks*, 1999.

[21] Jain, A. and Flynn, P. and Ross, A. Eds. *Handbook of Biometrics*. Springer, 2007.

[22] James C. Spall. *Introduction to Stochastic Search and Optimization*. Wiley, 2003.

[23] J. Johansson and S. Riley. *Protect Your Windows Network: From Perimeter to Data*. Addison-Wesley, 2005.

[24] A. Joshi, S. King, G. Dunlap, and P. Chen. Detecting Past and Present Intrusions through Vulnerability-specific Predicates. In *SOSP*, 2005.

[25] M. Kaminsky, G. Savvides, D. Mazieres, and M. Kaashoek. Decentralized User Authentication in a Global File System. In *SOSP*, 2003.

[26] S. King and P. Chen. Backtracking Intrusions. *ACM Transactions on Computer Systems (TOCS)*, 23(1):51–76, 2005.

[27] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster Approximation Algorithms for the Unit Capacity Concurrent Flow Problem with Applications to Routing and Finding Sparse Cuts. In *SIAM JOURNAL ON COMPUTING*, 1994.

[28] M. Krohn, M. Brodsky, M. Kaashoek, and R. Morris. Information Flow Control for Standard OS Abstractions. In *SOSP*, 2007.

[29] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. In *ACM Transactions on Computer Systems (TOCS)*, 1992.

[30] Live ID. http://winliveid.spaces.live.com/.

[31] M. Luby and N. Nisan. A Parallel Approximation Algorithm for Positive Linear Programming. In *STOC*, 1993.

[32] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. In *IEEE Security and Privacy*, 2003.

[33] MT. Hajiaghayi and H. Räcke. An $O(\sqrt{n})$-approximation algorithm for directed sparsest cut. In *Information Processing Letters*, 2006.

[34] T. Munzner. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, 2000.

[35] P. Naldurg, S. Schwoon, S. Rajamani, and J. Lambert. NETRA: Seeing Through Access Control. In *ACM Workshop on Formal Methods in Security*, 2006.

[36] J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *NDSS*, 2005.

[37] S. Noel and S. Jajodia. Managing Attack Graph Complexity Through Visual Hierarchical Aggregation. In *ACM Workshop on Visualization and Data Mining for Computer Security*, 2004.

[38] X. Ou, S. Govindavajhala, and A. Appel. MulVAL: A Logic-based Network Security Analyzer. In *USENIX Security*, 2005.

[39] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost In The Browser: Analysis of Web-based Malware. In *HotBots*, 2007.

[40] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC*, 2006.

[41] W. Rankl and W. Effing. *Smart Card Handbook*. Wiley, 2004.

[42] R. Rivest and B. Lampson. SDSI - A Simple Distributed Security Infrastructure. Crypto, 1996.

[43] S. Schechter, J. Jung, W. Stockwell, and C. McLain. Inoculating SSH Against Address Harvesting. In *NDSS*, 2006.

[44] E. Schultz. A framework for understanding and predicting insider attacks. In *Conference on Computers and Security (CompSec)*, 2002.

[45] Secure4Privilege White Paper on Unix Root Accounts. http://www.s4software.com/PDF/s4privilege_whitepaper.pdf.

[46] Security Assertion Markup Language. http://saml.xml.org.

[47] F. Shahrokhi and D. Matula. The Maximum Concurrent Flow Problem. In *JACM*, 1990.

[48] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *IEEE Security and Privacy*, 2002.

[49] A. Singer. Tempting Fate. *USENIX login*, 30(1):27–30, 2005.

[50] E. Spafford. The Internet Worm Program: An Analysis. *ACM SIGCOMM Computer Communication Review*, 19(1):17–57, 1989.

[51] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *USENIX*, 1988.

[52] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-Attack Graph Generation Tool. In *DARPA Information Survivability Conference and Expo (DISCEX)*, 2001.

[53] Symark White Paper on Unix Root Accounts. http://www.symark.com/downloads/whitepapers/Symark_Privileged_Access_Control.html.

[54] System Center Operations Manager. http://www.microsoft.com/systemcenter/operationsmanager.

[55] User-Workstations Attribute. http://msdn.microsoft.com/en-us/library/ms680868(VS.85).aspx.

[56] G. Wahba. *Support Vector Machines, Reproducing Kernel Hilbert Spaces, and Randomized GACV*. MIT Press, 1999.

[57] I. Winkler and B. Dealy. Information Security Technology?... Don't Rely on It: A Case Study in Social Engineering. In *USENIX Security*, 1995.

[58] T. Wobber, A. Yumerefendi, M. Abadi, A. Birrell, and D. Simon. Authorizing Applications in Singularity. In *EuroSys*, 2007.

[59] N. Young. Sequential and Parallel Algorithms for Mixed Packing and Covering. In *FOCS*, 2001.

[60] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazieres. Making Information Flow Explicit in HiStar. In *OSDI*, 2006.

[61] C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. In *ACM CCS*, 2002.