Secure "Selecticast" for Collaborative Intrusion Detection Systems

Philip Gross, Janak Parekh and Gail Kaiser
Columbia University
Programming Systems Lab
[phil|janak|kaiser]@cs.columbia.edu

Abstract

The problem domain of Collaborative Intrusion Detection Systems (CIDS) introduces distinctive data routing challenges, which we show are solvable through a sufficiently flexible publish-subscribe system. CIDS share intrusion detection data among organizations, usually to predict impending attacks earlier and more accurately, e.g., from Internet worms that tend to attack many sites at once. CIDS participants collect lists of suspect IP addresses, and want to be notified if others are suspicious of the same addresses. The matching must be done efficiently and anonymously, as most organizations are reluctant to share potentially revealing information about their networks. Alerts regarding external probes should only be visible to other CIDS participants experiencing probes from the same source(s). We term this type of simultaneous publish/subscribe "selecticast." We present a potential solution using the secure Bloom filter data structure propagated over the MEET publish-subscribe framework.

1. Introduction

Increasingly, malicious attackers attempt to avoid tripping an Intrusion Detection System (IDS) by spacing out their probes over long periods of time, and interleaving probes among multiple sites over these extended periods [1]. Most current IDS's have a limited memory window, or employ thresholding before signaling an alert in order to avoid generating a flood of false alerts. In contrast, Collaborative Intrusion Detection Systems (CIDS) attempt to correlate low-level alerts from multiple enclaves in the same organization or across organizations over a long period of time in order to detect these stealthy scanners prior to an actual attack. Participating sites submit lists of suspicious addresses, including those that fall below their own thresholds for escalating alerts (e.g., to human administrators). Each site compares others' lists against its own, looking for matches, which are then escalated to a higher level of suspicion and monitoring.

One approach to implementing CIDS is a centralized repository that receives all the "watchlists" and sends back augmented watchlists to each participant. However, this creates a central point of failure and a tempting target for denial-of-service attacks. It also requires revealing all the data to the central site, but the organizations may only want to share with others that are targets of the same prospective attacker(s). The nature of the latter communication suggests a content-based messaging system, but in a context where participants (particularly when from independent institutions) may demand that their data be anonymized. That is, if participant A has no low-level alerts in common with participant B, it will learn nothing about participant B's network structure or traffic from the data set. Only if there is an alert in common will the information of that particular alert, and only that alert, be revealed.

Such a peer-to-peer system should ensure that the data streams remain private, using some form of encryption. However, content-based messaging systems are inherently difficult to use for encrypted message streams. Content-based routing (CBR) implies that routers inspect the contents of every packet at each routing point, which is unacceptable if routers aren't trusted. Encryption is not a problem in channel-based routing, since routers along the path don't need to inspect content, and only privileged subscribers can decrypt the events. However, channels are not well suited for this problem domain, as each IP probe source would need its own channel, which is potentially a huge number. Encryption is also less of an issue if the content-based routers can be trusted, as each message could safely be decrypted and inspected within the router (although this is computationally expensive). Additionally, trusted routers could enforce access control policies (e.g., [2] or [3]) to guarantee that events are forwarded only to subscribers whose security credentials match those acceptable to the publishers.

However, ensuring that all intermediate routers are trustworthy will not generally be possible. We present a new approach that supports a restricted form of content-based event routing with only minimal trust required of routers. In particular, we trust the routers to forward

messages to subscribers without forging, altering, or discarding them, and to implement our specialized algorithms correctly. Rogue routers that do attempt to interpret the events themselves for malicious purposes, that forward false matches, or forward to additional entities pose little or no threat, as explained below. The main restriction on our approach is that content filtering is limited to equality and inequality, as opposed to other comparisons on event content such as less than, greater than, substring of, etc.

Our main insight is to employ *Bloom filters* for representing hashed alert sets. A Bloom filter [4], described further below, is essentially a compact representation of a set of hashes. A router receives sets of hashed values from participants (publishers), which are then checked against others' (subscribers') Bloom filters. Matched values are sent back to the submitter and to the matched participants; we call this symmetric form of publish-subscribe "selecticast". The router then converts the submitted values to a much smaller Bloom filter, and discards the originals. There is a small possibility of false matches, which can be decreased by increasing the size of the Bloom filters and/or the number of hashes per input value [5]. For the CIDS application domain, false positives are not a major issue, assuming the rate is sufficiently low, as it merely implies that a small percentage of addresses will be temporarily flagged for closer monitoring despite being innocent.

We present one such system, including its methods for minimizing false matches. We first describe the collaborative intrusion detection system motivating this work. We then explain how Bloom filters operate and criteria for selecting the hashes. We compare several alternative approaches to CBR based on secure hashes. We briefly discuss the system's integration into a modular event system and its extension to a distributed implementation. Finally, we survey related work, and conclude with the current status and real-world evaluation plans.

2. Motivation

The current emphasis on security has led to the development and deployment of sophisticated traffic analysis tools, honeypots, and intrusion detection systems. A fundamental limitation of such systems is the single-point perspective on suspicious activity they offer: such activity can only be examined from the point of view of the sensor, which is attached to only one network. Patient attackers can slowly scan several targets in parallel, without creating enough traffic against any one of them to warrant an alert. Such low-frequency events can be easily lost in the sea of alerts generated by IDS's.

A *collaborative* intrusion detection system, such as described in [6], shares IDS alert information among sites within a large organization or across different organizations, thereby enriching the information available to each, and revealing far more detail about the behavior of attackers than would otherwise be possible.

Assume we already have a collection of sites, each hosting an IDS performing surveillance detection, i.e., tracking connections and failed or incomplete connection attempts, and mapping these activities to source IP addresses (or ranges of addresses) as much as memory permits. The Antura Recon detector is a commercial example of such a surveillance-detection-enabled IDS [7]. By identifying such sources, sites progress from detecting attacks as they occur to predicting impending attacks.

In CIDS, we correlate these alerts from IDS's across multiple sites. With sufficient participation in the collaboration, it now becomes possible to detect stealthy scanners who relatively rarely probe any given site but are slowly testing multiple target sites. Collaborative "watchlists" across multiple sites aggregate the activities of source IPs (or ranges of IPs) that would likely fall under the radar at any individual site. A critical concern, however, is that sites be able to participate without revealing confidential or sensitive information about their networks and traffic. By hashing the alerts before transmission and correlating and distributing them with selecticast, we can solve the problem simply and efficiently.

3. Secure Hashes and Bloom Filters

A Bloom filter [4] is a "one-way" data structure, consisting of a bit-string that represents hash hits. It is one-way in the sense that one can test to determine whether a given filter has seen a particular datum before, and the filter will answer with no false negatives and rare false positives. Thus, the Bloom filter does not reveal its contents; it can only confirm whether a specific value is stored.

More precisely, Bloom filters are used to probabilistically and compactly represent subsets of some universe U. A Bloom filter is implemented as an array of m bits, uses k hash functions mapping elements in U to $[0..m)$, and supports two basic operations: **add** and **query**. Initially, all bits in the Bloom filter are set to 0. To **add** $u \in U$ to a Bloom filter, the hash functions are used to generate k indices into the array and the corresponding bits are set to 1. A location can be set to 1 multiple times, but only the first has an effect. A **query** is positive if and only if all k referenced bits are 1. A negative query clearly indicates that the element is not in the Bloom filter, but a positive query may be due to a false positive: the case in which the queried element was not added to the Bloom filter, but all k queried bits are 1 due to other additions. We use k independent indices,

instead of just a single index, to reduce the probability of a such a false positive.

The probability of false positives is an important metric because minimizing it is the key to making effective use of Bloom filters. The analysis proceeds as follows. If p is the probability that a random bit of the Bloom filter is 1, then the probability of a false positive is p^k, the probability that all k hash functions map to a 1. If we let i be the number of elements that have been added to the Bloom filter, then $p = 1-(1-1/m)^{ik}$, as ik bits were randomly selected, with probability $1/m$, in the process of **add**ing i elements. In [8] and [5], it is shown that the probability of false positives is minimized when k is approximately $m/i \ln 2$.

The reason the above analysis is with respect to k, the number of hash functions, is that we do not control i, the number of additions, in our application – it's dictated by network traffic. What we can control is m, the amount of memory used and k, the number of hash functions. The collaborative security participants need to choose m and k so that the probability of false positives is acceptably minimized. Small values of k lead to large values of m, whereas small values of m lead to large values of k. The smaller m, the more compact our Bloom filters, but the smaller k, the faster the implementation – at routers as well as subscribers. As discussed in the next section, hash computation is the dominant cost and it depends on k.

4. CBR With Bloom Filters and Hashes

Bloom filters efficiently represent a set of hash values in a small space. Good results are typically obtainable with filters with only eight times as many bits as there are items being stored [5]. One can merge two or more Bloom filters by simply binary-ORing them together (at the cost of a higher false-positive rate).

Bloom filters also have some disadvantages, most significantly the impossibility of deletion, although alternate schemes allowing deletion (at the cost of less space efficiency) have been proposed [9]. Additionally, as with standard hash tables, the hash values have usually been adjusted modulo the size of the table, so increasing the size requires rehashing the original values.

We will assume that it is undesirable (insecure) for the routers to see raw, unhashed values, which will typically be alerts containing sensitive IP address and port information. There are several different ways to leverage Bloom filters to represent the hash-value sets participants are publishing. We could have clients submit Bloom filter bit-strings representing the hashed addresses of interest, and have routers use Bloom filters internally. We could have clients submit the sets of hash values and have routers organize them into Bloom filters. Or we could have clients submit lists of hash values and have the routers use standard hash tables.

There are a number of costs to consider, and the optimal solution will depend on the specific attributes of our network and the needs of our CIDS "selecticast." Among the important cost metrics are the size of the "selecticast" submissions and notifications in transit, the size of the subscription representations in router memory, and the speed with which the router can compute intersections. Another important variable is the specificity of participant notification: do participants merely need to know that others have seen a particular alert, the number who have seen the alert, or the actual identities of all who have seen an alert?

Plain Hash Tables

If clients submit lists of the hash values, the most straightforward approach for the router is to simply maintain a hash table of all submissions. Each entry in the hash table links to a list of submitters. When a new set is submitted, the router adds each entry to the master hash table. If the submitter list for that entry was non-empty, a notification containing the new submitter list and the hash value is sent to all entries in the list.

This implementation has the advantages that there will be no false positives except for rare hash collisions. It also allows deletion, enabling the submission of updates, as opposed to complete lists. If a hashed alert is flagged for deletion, the submitter will be silently deleted from the submitter list for that entry in the master hash table.

Its main disadvantage compared with Bloom filters is size. The exact size depends on the specific type of hash table constructed and the underlying architecture of the system (e.g., pointer size). Assuming both hash values and pointer size are 32 bits (reasonable for our expected alert set cardinality of 10^5-10^6), an optimal open hash table (load factor 0.50) storing n items will use around $64n$ bits, and an optimal chained hash table (load factor around 0.75) will use around $85n$ bits (64 bit entries, 32 bits for value and 32 for pointer, times 4/3 for optimal load). For either type, if each entry also has a linked list of submitters, add an extra $64n$.

Pure Bloom Filters

At the opposite extreme, we could deal purely with Bloom filters. In this case, participants would submit a Bloom filter representation of their dataset. In order to look for matches, we would directly compare the Bloom filters, counting the number of bits in common. Filters with any matching elements must have at least k bits in common, where k is the number of Bloom filter bits we set for each input value.

Unfortunately, this approach will usually not be practical. The number of bits that match due to random chance will be huge for any typical pair of Bloom filters.

Suppose we have two Bloom filters of size m bits, each storing n distinct values (i.e., no values in common) using k bits per item. A bit in the filter will be set with probability p, roughly $1-(1-1/m)^{kn}$. For optimal Bloom filters, p should be near 0.5, although we can make it much sparser at the cost of increased memory usage.

For our candidate Bloom filter, we can view the case that one of our bits coincidently matches a bit in the other filter as a Bernoulli trial with chance of "success" p. The number of matching bits will then follow a binomial distribution, with the expected number of successes in kn trials equal to knp. Computer simulation confirms the accuracy of this analysis. knp will be vastly greater than k, and thus the number of false positives will be enormous. Even if we try to lower knp by making p extremely small (and making the filter extremely sparse), large values of n will rapidly make the situation unworkable.

For instance, if k is 6, and thus we want our expected number of collisions to be less than 6, we must put fewer than 1200 items into an 8 million bit (1MB) Bloom filter (k=6, n=1183, m=2^{23}, p≈0.0008, knp≈6.0). Note that this cost is only incurred in memory. We can compress the filter during network transmission by a large factor using standard compression tools. Nonetheless, and even given the speed of simply ANDing the two large Bloom filters together, 7000+ bits per item stored is a highly unattractive ratio.

Hybrid Bloom Filter

We can successfully leverage the size advantage of Bloom filters by combining them with the set-of-hash-values approach. Participants submit the list of hash values of interest, representing noted instances of suspicious network activity. The router uses the actual hash values to check against the Bloom filters of the other participants to find matches with reasonable accuracy. If matches are found, the router sends the matching values as a notification to all matching participants. Additionally, the router converts the submitted set of hash values into a Bloom filter of size $8\hat{n}$ bits, where $\hat{n}$ is the estimated total number of values per participant (making all filters the same size, and giving a chance of false positive around 2%). This filter is then stored and associated with the submitter. After all of the submitted hash values have been checked against everyone else's Bloom filters, the router can then discard the submitted hash value list, leaving only the subscriber's (much smaller) Bloom filter.

Thus publishers submit large sets of hash values, which are used to find matches, and then leave behind much smaller Bloom filter "residues" that act as subscriptions. Matching hash value sets (optionally tagged with the identity of their submitter) are sent out as the actual notifications. For this domain, we assume that the number of notifications is very small in relation to the number of values submitted (experiments show correlation rates of 0.01% or lower). If the number of matches is expected to be large, the matching sets could themselves be converted to Bloom filters before being sent as notifications, with the attendant space savings.

Note that the hash values used for Bloom filter generation are much larger than the hash values used by a plain hashtable, even though the resulting filter structure is smaller than the corresponding hashtable. For each item entered, Bloom filters need k indices into an m-bit table, and thus a total of $k\ln m$ bits of hash per item. If m=8n, then $k(3+\ln n)$ bits. For sets of 2,000 to 128,000 items and k=6, this works out to 84-120 bits per item, or a factor of 3-4 increase over the size of the hash values needed for plain hashtables. This would potentially be a problem for the submission of large sets of hashed values in the hybrid case.

However, we can avoid this problem by hashing our alerts to 32 bits for transmission, and then rehashing each to 120+ bits after it arrives at the server (and then splitting up those bits into the k indices of size $\ln m$ that we need), thus making the transmission cost no more expensive than for plain hashtables. Since the original alerts will typically contain less than 32 bits of entropy, no information should be lost with this two-stage hashing process.

Optimization with Two-Stage Compare

In the hybrid case described above, we assumed that the router maintains a separate Bloom filter for each of the C collaborating parties, representing the specific set of alerts seen by that party. When a new set of values is published, it must be compared against each of the C-1 other sets. We can speed processing by entering all of the submitted value sets into a single large "master" Bloom filter in the Router and checking this first.

If we find a match in the master Bloom filter, we must then check each individual filter to discover the specific participants who matched. Despite this, we will show that this approach can offer substantial space efficiencies over the hash table approach, and the speed disadvantage can be reduced.

We can speed our Bloom filter lookups by taking advantage of arithmetic modulo 2^m on binary numbers. Just as a base 10 number modulo 10^m is the least significant m digits of the number, a binary number modulo 2^m is just the bottom m bits of the number, which can be extracted by ANDing the number with an appropriate bit mask (`((1<<m)-1` using the C-language bit operators).

Let n' be the power of 2 closest to n. We create one master Bloom filter of size Cn' and a filter of size n' for each of the C participants. Sizing these at 8 bits per item,

we have a total space of $16Cn'$. The single hash table approach, as described above, will use $128Cn$-$150Cn$ bits to encode the same information. Even if we choose $n'=2n$, the total size of our Bloom filters will be a quarter or less of the size of the hash table solution. To do a lookup, we take our k hash values, compute indices modulo Cn', and do our lookups into the master Bloom filter. If all k indices match, we simply take the bottom $\log_2 n'$ bits of each master table index value, and use these as our search indices into the size n' subtables. Thus we only need to compute a nontrivial modulus once. If C is also a power of 2, both hash resizings become single AND instructions.

Aging

The master Bloom filter still has one major weakness vis-à-vis the master hash table solution. Participants will be publishing new alert lists to the network on a regular basis. While we can easily add new values to the master Bloom filter (just keep setting the appropriate bits), we have no way to delete out-of-date entries, and our master filter will gradually fill up with junk bits, until the probability of a positive response for any input approaches 1.

One solution is to maintain a "shadow" copy of the "primary" master Bloom filter (at a cost of an extra $8Cn'$ bits), and periodically swap the two. At startup, after the primary master Bloom filter is initialized from all the participants' data, the shadow copy is cleared. When participants subsequently publish a new set of values, their individual Bloom filter is replaced, and both the primary and shadow filters are updated with the new values. After all participants have submitted new data (or a preset time interval is elapsed), the shadow table becomes the new primary table, and the old primary table is cleared and becomes the new shadow table.

During the period where many new sets have been added to the current primary table, the number of false positives it returns will increase. However, as the individual participant subtables are always up-to-date, this should not result in a much higher rate of actual false positive messages transmitted back to subscribers, as all of the secondary checks for specific matches will fail. The only effect will be a reduction of the primary table's efficiency in filtering. If value set updates are largely similar to the previous set, the performance degradation will be even smaller.

MEET

The Multiply Extensible Event Transport (MEET) is a modular publish-subscribe system currently under development that allows users to define their own data types and predicates on those types to be used as filters. MEET allows enhancements to the classic publish-subscribe paradigm through the addition of new modules.

In the above discussion, we have assumed a single router node. We can use this extensible system implement a fully distributed solution. We wish to distribute the task of matching values among multiple routers. We can achieve even distribution of the computation by assigning particular hashes to particular routers with a mechanism based on Distributed Hash Table routing. For instance, if we have 16 routers, the first handles all hashed values ending with 0000, the second all hashes ending with 0001, etc.

MEET enables DHT routing and selecticast through the addition of data type, filter, and routing modules.Further discussion of MEET is outside the scope of this paper.

5. Related Work

A number of sophisticated publish-subscribe systems have been developed, including Siena [10], Gryphon [11], JEDI [12], ECho [13], CORBA Events [14], and Elvin [15]. Siena, Gryphon, JEDI, and Elvin are all content-based, where intermediate routers analyze the contents of each packet to determine appropriate forwarding destination(s). Wang et al. [16] examined security issues for CBR, but focused on the (as yet unsolved) problems of evaluating complex filters (i.e., more complex than simple equality testing, e.g., ad-hoc range checking) on encrypted data.

Bloom filters have been studied for a number of applications, including wide-area service discovery [17], IP packet traceback [18], and distributed caching services[8], in addition to being a primary tool for relational database joins [19]. The most germane work is probably by Triantafillou and Economides [20, 21], who use Bloom filters to create "subscription summaries," allowing for radical speedups to standard content-based routing. To our knowledge, no one has proposed using it for management of large numbers of opaque subscriptions in publish/subscribe systems. We also know of no other "selecticast" systems where publications are also subscriptions.

Others have investigated CIDS, e.g., [22] [23] [24] [25], but none have proposed an event infrastructure for data distribution.

6. Status and Conclusions

We believe that our proposed two-level system of Bloom filters will allow efficient and secure correlation of data as required by collaborative intrusion detection systems. The launch of a second generation CIDS, using MEET with Bloom filters extended as discussed here, is planned as a joint project between Columbia, Georgia Tech, Florida Institute of Technology, Syracuse

University, MIT, USC/ISI and the Brookings Institute. We expect to be able to compare our performance data against first generation CIDS trials involving Columbia, George Tech, and the University of Pennsylvania, where the raw data was collected and correlated at a centralized site.

Our extensions to Bloom filters may also prove useful for other secure content-based routing applications where equality/inequality testing of values is sufficient. Publications and subscriptions do not necessarily have to be symmetric, as in our "selecticast", but instead subscriptions could be provided directly as Bloom filters.

7. Acknowledgements

We would like to thank Sal Stolfo and the other members of his Intrusion Detection Systems Lab for developing the collaborative security framework motivating this event system research; Angelos Keromytis, Vishal Misra, Jason Nieh, Dan Rubenstein and Henning Schulzrinne for helpful suggestions; and the other members of our Programming Systems Lab. PSL is funded in part by National Science Foundation grants CCR-0203876, EIA-0202063 and EIA-0071954, and by Microsoft Research.

8. References

[1] CERT Coordination Center. *Module 4 - Types of Intruder Attacks*, *CERT/CC Overview Incident and Vulnerability Trends*. 2003.

[2] Blaze, M., J. Feigenbaum, and A.D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. *Security Protocols International Workshop*. 1998. Springer LNCS.

[3] Keromytis, A.D., et al. The STRONGMAN Architecture. *3rd DARPA Information Survivability Conference and Exposition*. 2003.

[4] Bloom, B.H., Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970. 13(7): p. 422-426.

[5] Mitzenmacher, M., Compressed bloom filters. *IEEE/ACM Transactions on Networking*, 2002. 10(5): p. 604-612.

[6] Locasto, M., et al. *Secure and Efficient Privacy–Preserving Multi–Organization Intrusion Detection*, *Technical Report CUCS-012-04*. 2004.

[7] Robertson, S., et al. Surveillance Detection in High Bandwidth Environments. *3rd DARPA Information Survivability Conference and Exposition*. 2003.

[8] Fan, L., et al. Summary cache: a scalable wide-area Web cache sharing protocol. *SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 1998.

[9] Cohen, S. and Y. Matias. Spectral bloom filters. *ACM SIGMOD International Conference on Management of Data*. 2003.

[10] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001. 19(3): p. 332-383.

[11] Aguilera, M.K., et al. Matching events in a content-based subscription system. *18th Annual ACM Symposium on Principles of Distributed Computing*. 1999.

[12] Cugola, G., E. Di Nitto, and A. Fuggetta, The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 2001. 27(9): p. 827-850.

[13] Eisenhauer, G., F.E. Bustamante, and K. Schwan. Event services for high performance computing. *9th International Symposium on High-Performance Distributed Computing*. 2000.

[14] Harrison, T.H., D.L. Levine, and D.C. Schmidt. The design and performance of a real-time CORBA event service. *ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages and Applications*. 1997.

[15] Segall, B., et al. Content based routing with Elvin. *Australian UNIX and Open Systems User Group*. 2000.

[16] Wang, C., et al. Security Issues and Requirements for Internet-scale Publish-Subscribe Systems. *35th Annual Hawaii International Conference on System Sciences*. 2002.

[17] Czerwinski, S.E., et al. An Architecture for a Secure Service Discovery Service. *Mobile Computing and Networking*. 1999.

[18] Snoeren, A.C. Hash-based IP traceback. *Conference on Applications, technologies, architectures, and protocols for computer communications*. 2001.

[19] Mackert, L.F. and G.M. Lohman. R* optimizer validation and performance evaluation for local queries. *ACM SIGMOD International Conference on Management of Data*. 1986.

[20] Triantafillou, P. and A. Economides. Subscription summaries for scalability and efficiency in publish/subscribe systems. *22nd International Conference on Distributed Computing Systems Workshops*. 2002.

[21] Triantafillou, P. and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. *24th International Conference on Distributed Computing Systems*. 2004.

[22] Yegneswaran, V., P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. *11th Annual Network and Distributed System Security Symposium*. 2004.

[23] Markatos, E. *A European Network of Affined Honeypots*, Private communication, 2004.

[24] Balasubramaniyan, J.S., et al. An Architecture for Intrusion Detection Using Autonomous Agents. *Annual Computer Security Applications Conference*. 1998.

[25] Cuppens, F. and A. Miege. Alert correlation in a cooperative intrusion detection framework. *IEEE Symposium on Security and Privacy*. 2002.