"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XV – 2012 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania

A NEW APPROACH TO SECURING THE SETUP AND FLOODING STAGES OF ROUTING ALGORITHMS

Ioan POPOVICIU[1]
[1] Assoc. Prof., PhD, "Mircea cel Batran" Naval Academy

Abstract: *In this paper we describe a method to securing the steps of routing algorithms, using a public-key infrastructure and cryptographic hashing of messages to achieve security. This method can secure several routing algorithms for networks where a single bad router can affect an entire network.*
Key words: *network, router, protocol,algorithm, security*

1. INTRODUCTION

Routing messages in a network is an essential component of Internet communication, as each packet in the Internet must be passed quickly through each network (or autonomous system) that it must traverse to go from its source to its destination. It should come as no surprise, then, that most methods currently deployed in the Internet for routing in a network are designed to forward packets along shortest paths. Indeed, current interior routing protocols, such as OSPF, RIP, and IEGP, are based on this premise, as are many exterior routing protocols, such as BGP and EGP.

The algorithms that form the basis of these protocols are not secure, however, and have even been compromised by routers that did not follow the respective protocols correctly. Fortunately, all network malfunctions resulting from faulty routers have to date been shown to be the result of meson figured routers, not malicious attacks. Nevertheless, these failures show the feasibility of malicious router attacks, for they demonstrate that compromising a single router can undermine the performance of an entire network.

In this paper we describe a new approach to securing the setup and flooding stages of routing algorithms. After a preliminary setup that involves distributing a set of secret keys equal that total no more than the number of routers, our method uses simple cryptographic hashing of messages (HMACs) to achieve security.

2. FLOODING

We begin by discussing the flooding protocol and a low-cost way of making it more secure. Our method involves the use message-authenticating scheme using cryptographic hashing.

2.1 The Network Framework and the Flooding Algorithm

Let $G = (V;E)$ be a network whose vertices in V are routers and whose edges in E are direct connections between routers. We assume that the routers have some convenient addressing mechanism that allows us without loss of generality to assume that the routers are numbered 1 to n. Furthermore, we assume that G is disconnected, that is, that it would take at least two routers to fail in order to disconnect the network.

The flooding algorithm is initiated by some router s creating a message M that it wishes to send to every other router in G. The typical way the flooding algorithm is implemented is that s incrementally assigns sequence numbers to the messages it sends. So that if the previous message that s sent had sequence number j, then the message M is sent with sequence number $j + 1$ and an identification of the message source, that is, as the message $(s; j +1;M)$. Likewise, every router x in G maintains a table S_x that stores the largest sequence number encountered so far from each possible source router in G. Thus, any time a router x receives a message $(s; j + 1; M)$ from an adjacent router y, the router x first checks if $S_x[s] < j +1$. If so, then x assigns $S_x[s] = j +1$ and x sends the message $(s; j +1; M)$ to all of its adjacent routers, except for y. If the test fails, however, then x assumes it has handled this message before and it discards the message.

If all routers perform their respective tasks correctly, then the flooding algorithm will send the message M to all the nodes in G. Indeed, if the communication steps are synchronized and done in parallel, then the message M propagates out from s is a breadth-first fashion.

If the security of one or more routers is compromised, however, then the flooding algorithm can be successfully attacked. For example, a router t could spoof the router s and send its own message $(s; j + 1;M')$. If this router reaches a router x before the correct message, then x will propagate this imposter message and throw away the correct one when it finally arrives. Likewise, a corrupted router can modify the message itself, the source identification, and/or the sequence number of the full message in transit. Each such modification has its own obvious bad effects on the network.

2.2 Securing the Flooding Algorithm on General Networks

On possible way of avoiding the possible failures that compromised or meson figured routers can inflict on the flooding algorithm is to take advantage of a public-key infrastructure defined for the routers. In this case, we would have s digitally sign every flooding message it transmits, and have every router authenticate a message before sending it on. Unfortunately, this approach is computationally expensive.

Our scheme is based on next strategy. The initial setup for our scheme involves the use of a public-key infrastructure, but the day-to-day operation of our strategy takes advantage of much faster cryptographic methodologies. Specifically, we define for each router x the set $N(x)$, which contains the vertices (routers) in G that are neighbours of x (which does not include the vertex x itself). That is, $N(x) = \{y : (x, y) \in E \text{ and } y \neq x\}$. The security of our scheme is derived from a secret key $k(x)$ that is shared by all the vertices in $N(x)$, but not by x itself. This key is created in a setup phase and distributed securely using the public-key infrastructure to all the members of $N(x)$. Note, in addition, that $y \in N(x)$ if and only if $y \in N(y)$.

Now, when s wishes to send the message M as a flooding message to a neigh boring router, x, it sends $(s, j+1, M, h(s \mid j+1 \mid M \mid k(x)), 0)$, where h is a cryptographic hash function that is collision resistant. Any router x adjacent to s in G can immediately verify the authenticity of this message (except for the value of this application of h), for this message is coming to x along the direct connection from s. But nodes at distances greater than 1 from s cannot authenticate this message so easily when it is coming from a router other than s. Fortunately, the propagation protocol will allow for all of these routers to authenticate the message from s, under the assumption that at most one router is compromised during the computation.

Let (*s; j* +1*; M; h*1*; h*2) be the message that is received by a router *x* on its link from a router *y*. If *y* = *s*, then *x* is directly connected to *s*, and *h*2 = 0. But in this case *x* can directly authenticate the message, since it came directly from *s*. In general, for a router *x* that just received this message from a neighbour *y* with $y \neq s$, we inductively assume that *h*2 is the hash value $h(s \mid j+1 \mid M \mid k(y))$. Since *x* is in *N*(*y*), it shares the key *k*(*y*) with *y's* other neighbors; hence, *x* can authenticate the message from *y* by using *h*2. This authentication is sufficient to guarantee correctness, assuming no more than one router is corrupted at present, even though *x* has no way of verifying the value of *h*1.

So to continue the propagation assuming that flooding should continue from *x*, the router *x* sends out to each *w* that is its neighbor the message $(s, j+1, M, h(s \mid j+1 \mid M \mid k(w)), h_1)$. Note that this message is in the correct format for each such *w*, for *h*1 should be the hash value $h(s \mid j+1 \mid M \mid k(x))$, which *w* can immediately verify, since it knows *k*(*x*). Note further that, just as in the insecure version of the flooding algorithm, the first time a router *w* receives this message, it can process it, updating the sequence number for *s* and so on.

2.3 Trading Message Size for Hashing Computations

In some contexts it might be too expensive for a router to perform as many hash computations as it has neighbors. Thus, we might wonder whether it is possible to reduce the number of hashes that an intermediate router needs to do to one. In this subsection we describe how to achieve such a result, albeit at the expense of increasing the size of the message that is sent to propagate the flooding message.

In this case, we change the preprocessing step to that of computing a small-sized coloring of the vertices in *G* so that no two nodes are assigned the same color. Algorithms for computing or approximating such colorings are known for a wide variety of graphs. For example, a tree can be colored with two colors. Such colorings might prove useful in applying our scheme to multicasting algorithms, since most multicasting communications actually take place in a tree. A planar graph can be colored with four colors, albeit with some difficulty, and coloring a planar graph with five colors is easy. Finally, it is easy to color a graph that has maximum degree *d* using at most *d*+1 colors by a straightforward greedy algorithm. This last class of graphs is perhaps the most important for general networking applications, as most communications networks bound their degree by a constant.

Let the set of colors used to color *G* be simply numbered from 1 to *c* and let us denote with V_i the set of vertices in *G* that are given color *i*, for *i* = 1*; 2; : : : ; c*, with $c \geq 2$. As a preprocessing step, we create a secret key k_i for the color *i*. We do not share this color with the members of V_i, however. Instead, we share k_i with all the vertices that are *not* assigned color *i*.

When a router *s* wishes to flood a message *M* with a new sequence number *j* + 1, in this new secure scheme, it creates a full message as (*s; j* + 1*;M; h*1*; h*2*; : : : ; hc*), where each $h_i = h(s \mid j+1 \mid M \mid k_i)$. (As a side note, we observe that the prefix of the bit string being hashed repeatedly by *s* is the same for all hashes, and its hash value in an iterative hashing function need only be computed once.) There is one problem for *s* to build this message, however. It does not know the value of k_i, where *i* is the color for *s*. So, it will set that hash value to 0. Then, *s* sends this message to each of its neighbors.

Suppose now that a router *x* receives a message (*s; j* + 1*;M; h*1*; h*2*; : : : ; hc*) from its neighbor $y \neq s$. In this case *x* can verify the authenticity of the message immediately, since it is coming along the direct link from *s*. Thus, in this case, *x* does not need to perform any hash computations to validate the message. Still, there is one hash entry that is missing in this message (and is currently set to zero): namely, $h_i = 0$, where *i* is the color of *s*. In this case, the router *x* computes $h_{j_i} = h(s \mid j+1 \mid M \mid k_j)$, since it must necessarily share the value of k_j, by the definition of a vertex coloring. The router *x* then sends out the (revised) message (*s; j* + 1*;M; h*1*; h*2*; : : : ; hc*).

Suppose then that a router *x* receives a message (*s; j* + 1*;M; h*1*; h*2*; : : : ; hc*) from its neighbor $y \neq s$. In this case we can inductively assume that each of the h_i values is defined. Moreover, *x* can verify this message by testing if $h_i = h(s \mid j+1 \mid M \mid k_i)$, where *i* is the color for *y*. If this test succeeds, then *x* accepts the message as valid and sends it on to all of its neighbors except *y*. In this case, the message is authenticated, since *y* could not manufacture the value of *hi*.

If the graph *G* is biconnected, then even if one router fails to send a message to its neighbors, the flood will still be completed. Even without biconnectivity, if a router modifies the contents of *M*, the identity of *s*, or the value of *j*+1, this alteration will be discovered in one hop. Nevertheless, we cannot immediately implicate a router *x* if its neighbor *y* discovers an invalid h_i value, where *i* is the color of *x*. The reason is that another router, *w*, earlier in the flooding could have simply modified this h_i value, without changing *s*, *j* + 1, or *M*. Such a modification will of course be discovered by *y*, but *y* cannot know which previous router performed such a modification. Thus, we can detect modifications to content in one hop, but we cannot necessarily detect modifications to h_i values in one hop. Even so, if there is at most one corrupted router in *G*, then we will discover a message modification if it occurs.

3. SETUP FOR DISTANCE-VECTOR ROUTING

Another important routing setup algorithm is the distance-vector algorithm, which is the basis of the well-known RIP protocol. As with the link-state algorithm, the setup for distance-vector algorithm creates for each router *x* in *G* a vector, D_x, of distances from *x* to all other routers, and a vector C_x, which indicates which link to follow from *x* to traverse a shortest path to a given router. Rather than compute these tables all at once, however, the distance vector algorithm produces them in a series of rounds.

3.1 Reviewing the Distance-Vector Algorithm

Initially, each router sets $D_x[y]$ equal to the weight, *w*(*x; y*), of the link from *x* to *y*, if there is such a link. If there is no such link, then *x* sets $D_x[y] = +\infty$. In each round each router *x* sends its distance vector to each of its neighbors. Then each router *x* updates its tables by performing the following computation:

for each router *y* adjacent to *x* **do**
    for each other router *w* **do**
        if $D_x[w] > w(x; y) + D_y[w]$ **then**
        {It is faster to first go to *y* on the way to *w}*
        Set $D_x[w] = w(x; y) + D_y[w]$
        Set $C_x[w] = y$
        endif
    endfor
endfor

If we examine closely the computation that is performed at a router *x*, it can be modeled as that of computing the minimum of a collection of values that are sent to *x* from adjacent routers (that is, the $w(x; y)+D_y[w]$ values), plus some comparisons, arithmetic, and assignments. Thus, to secure the distance-vector algorithm, the essential computation is that of verifying that the router *x* has correctly computed this minimum value.

3.2 Securing the Setup for the Distance-Vector Algorithm

Since the main algorithmic portion in testing the correctness of a round of the distance-vector algorithm involves validating the computation of a minimum of a collection of values, let us focus 7 more specifically on this problem. Suppose, then, that we have a node *x* that is adjacent to a collection of nodes $y_0, y_1, : : :, y_{d-1}$, and each node y_i sends to *x* a value a_i. The task *x* is to perform is to compute

$$m = \min_{i=0,1,...,d-1} \{a_i\}$$

in a way that all the y_i's are assured that the computation was done correctly. As in the previous sections, we will assume that at most one router will be corrupted during the computation (but we have to prevent and/or detect any fallout from this corruption). In this case, the router that we consider as possibly corrupted is *x* itself. The neighbors of *x* must be able therefore to verify every computation that *x* is to perform. To aid in this verification, we assume a preprocessing step has shared a key $k(x)$ with all *d* of the neighbors of *x*, that is, the members of $N(x)$, but is not known by *x*.

The algorithm that *x* will use to compute *m* is the trivial minimum-finding algorithm, where *x* iteratively computes all the pre_x minimum values

$$m_j = \min_{i=0,1,..., j} \{a_i\}$$

for $j = 0; : : : ; d-1$. Thus, the output from this algorithm is simply $m = m_{d-1}$. The secure version of this algorithm proceeds in four communication rounds:

1. Each router y_i sends its value a_i to *x*, as $A_i = (a_i, h(a_i \mid k(x)))$, for $i = 0; 1; : : : ; d - 1$.
2. The router *x* computes the m_i values and sends the message $\left(m_{i-1}, m_i, A_{i-1 \bmod d}, A_{i+1 \bmod d}\right)$ to each y_i. The validity of $A_{i-1 \bmod d}$ *and* $A_{i+1 \bmod d}$ is checked by each such y_i using the secret key $k(x)$. Likewise, each y_i checks that $m_i = \min\{m_{i-1}, a_i\}$.
3. If the check succeeds, each router y_i sends its verification of this computation to *x* as $B_i = (\text{"}yes\text{"}, i, m_i, h(\text{"}yes\text{"} \mid m_i \mid i \mid k(x)))$. (For added security y_i can seed this otherwise short message with a random number.)
4. The router *x* sends the message $\left(B_{i-1 \bmod d}, B_{i+1 \bmod d}\right)$ to each y_i. Each such y_i checks the validity of these messages and that they all indicated "yes" as their answer to the check on *x*'s computation. This completes the computation.

In essence, the above algorithm is checking each step of *x*'s iterative computation of the m_i's. But rather than do this checking sequentially, which would take $O(d)$ rounds, we do this check in parallel, in $O(1)$ rounds.

REFERENCES

[1] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *IEEE Symposium on Security and Privacy*, pages 115{124, 1998

[2] S. Cheung. An e_cient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, pages 90{98, 1997

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990

[4] R. Hauser, T. Przygienda, and G. Tsudik. Reducing the cost of security in link-state routing. *Computer Networks and ISDN Systems*, 1999

[5] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice-Hall, Englewood Cli_s, NJ, 1995

[6] S. Murphy, M. Badger, and B. Wellington. RFC 2154: OSPF with digital signatures, June 1997. Status: EXPERIMENTAL.