

Privacy Violation in *i3* overlay networks: problems and suggestions

Elisha J. Rosensweig
School of Computer Science
UMass Amherst
elisha@cs.umass.edu

Abstract—In this paper, it is shown that the addressing system of the *i3* architecture is vulnerable to anti-privacy attacks. Specifically, a large addressing space for trigger IDs can be scanned in logarithmic time to locate a random ID, that might be private. This technique can be used also to quickly and systematically scan the entire ID space and locate all private *ids*. Several policy suggestions are made, both for the network and the single *i3* user, that can reduce the practicality of this attack.

I. INTRODUCTION

It has been said that many networking problems can be solved by introducing an additional level of indirection. A classic and powerful example of the usefulness of such a concept can be found in the *i3* architecture [1]. Putting indirection to use in an elegant way, this overlay network has been shown to support many desirable functions that are complex or unavailable currently at the IP level, such as multicast, anycast and service composition.

i3 derives its appeal partially from its relative simplicity and generic approach to communication types. The support of unicast, multicast and anycast is handled in basically the same way. A sender of data directs the data to an intermediate *i3* node, from where the information flows to all those who registered to receive the information. Registration of this kind is done by entering a *trigger* at the *i3* node. This trigger is made up of two sections: an *id*, that specifies the messages to be forwarded, and the address of the end-host, to which such messages should be forwarded. In accordance with this, the message sender appends an *id* to each message sent to the *i3* node, from where it is then forwarded as determined by the inserted triggers. As multiple triggers can be entered bearing the same *id*, but each with a different IP address, it is clear how multicast and unicast can be achieved. A more detailed description of the addressing system of *i3* is presented below (section II).

As many times happens, the introduction of additional functionality is also susceptible to misuse in *i3*. Indeed, *i3* is open to many security attacks. For example, there is nothing to stop a malicious user (Eve) from entering a trigger with some *id* being used for private conversations between Alice and Bob, resulting in Eve eavesdropping on their conversation. Another example, this time of a DoS attack, is that of Eve entering a large amount of triggers with unique *ids* but a single destination address - that of Alice. This will cause a flooding

of data towards Alice from numerous sources, with very little effort in the long run on Eve's part.

These problems affect both publicly known and private triggers, but the solution suggested by *i3* distinguishes between them. With regards to private triggers, *i3* relies on a large *id* space and keeping triggers alive for short periods of time to avoid detection and consequent attacks [1], with obvious emphasis on the first factor. The chances of locating a private *id* by chance or brute-force attacks indeed seems impractical with a sufficiently large *id* space. However, solving these problems with public triggers requires a different approach. In order to deal with the vulnerability of public triggers, the *i3* team have taken the *i3* architecture to the next level. In [2][3] they present a more resilient system, called *Secure-i3*. In [2] they set out to demonstrate how this new version of *i3* is immune to many such attacks. Specifically, they show that it is no less secure than simply using IP, while at the same time providing additional functionality as previously explained. One of the changes to the original architecture is in the format of the trigger *ids*, as explained in detail in section II-B.

The purpose of this short paper is to demonstrate that private *ids* are vulnerable in *i3*, in the sense that they can be discovered in a fast way by an attacker with some partial information regarding the *id*. This paper shows that this is a direct result of certain fundamental addressing properties in *i3*. Additionally, this attack is impossible in the IP network layer while still possible in the *Secure-i3* architecture, contrary to the mission statement of *Secure-i3*. In section IV I discuss how, with certain modifications, *Secure-i3* could significantly reduce the potential of the attack, but currently it is as vulnerable as the standard *i3*.

It is important to emphasize that the attack presented here is limited in its applicability, and lacks a degree of control found in most attacks. The main purpose of this paper is, therefore, to demonstrate that an attack is possible, and draw attention to the properties of *i3* that allow such an attack to take place.

The layout of this paper is as follows. In Section II I give a detailed description of *i3* addressing and the way unicast, multicast and anycast are supported, for both standard and secure versions of *i3*. Armed with this understanding, Section III presents the privacy-violating attack, followed by suggestions for dealing with this problem in Section IV. The paper comes to a close with some short concluding remarks (Section V).

II. *i3* OVERVIEW AND NAMESPACE MANAGEMENT

A. *Standard-i3*

An *i3* address, or *id*, is a 256-bit number, divided into two 128-bit sections which are treated in distinct manners. The first section, consisting of the most significant bits, is associated with an *i3* node, in the sense that all triggers with this prefix are to be found under this node. The second half (the least significant bits) can be chosen randomly by users, and the large *id* space can ensure that a randomly selected *id* will not collide with other *ids*. For the rest of the paper, the first section is denoted msb_{id} while the second is lsb_{id} .

When a packet is sent with a certain *id*, it is directed to the *i3* node which is associated with the msb_{id} of this *id*. Once it arrives, it is forwarded to all destinations that have registered triggers that match the message *id*. A *matching id* in *i3* is any *id* that (a) exactly matches the message *id* in its msb_{id} and (b) has the *longest matching prefix* compared to all other trigger *ids* at the node with regards to the remaining lsb_{id} . We say that two *ids* *completely match* when they are identical in every bit. In such a way, several services can be supported:

- **Unicast** - send message with *id* completely matching the *id* of the message destination.
- **Multicast** - same as Unicast, only with multiple triggers with the same *id* but different destinations.
- **Anycast** - given a common prefix of possible destinations, select a random suffix and send the message. This will reach one destination at least, with no control over which one.

From this design it is clear how simple it is to join a multicast group: just as for unicast, all an end-host needs to do is to enter a trigger with the correct *id* at the *i3* node. This simplicity can also be used to eavesdrop on private conversations, which is why some *i3* *ids* need to remain private and disclosed only to trusted parties that are welcome to the conversation.

B. *Secure-i3*

Secure-i3 was designed with the purpose of dealing with the numerous security risks of *Standard-i3*. In this section I present only the novel features of *Secure-i3* that are relevant to this paper.

In order to prevent eavesdropping, *Secure-i3* places restraints on the *ids* selected for each public trigger. By placing these constraints, it is impossible for an attacker to arbitrarily select a (*id*, address) pair to be used as a trigger. Three types of constraints are discussed in the paper, and these are presented here using different names in order to avoid unnecessary detail:

- ***id*-constraint**. The *id* is hashed and the value of this hash is stored as part of the trigger. Originally termed *r-constraint*.
- ***rnd*-constraint**. A random 128-bit number, selected by the trigger owner, is hashed into 128 bits of the *id*. Originally one of two versions of *l-constraints*.
- ***rnd**-constraint**. A concatenation of a random 128-bit number and the end-host address is hashed into 128 bits of the *id*. Originally one of two versions of *l-constraints*.

Of these three constraints, the one used to point to end-hosts (as opposed to other *i3* nodes) is the *rnd*-constraint. This requirement is enforced by the *i3* architecture, thus blocking a potential eavesdropper from placing a trigger with an *id* of its own choosing, since the random, user-selected number is needed in order to place such a trigger. However, these constraints are placed only on public triggers that are at higher risk. Private triggers are still assumed, in *Secure-i3*, to be protected through the sheer size of the *id* space.

III. LIGHTNING ATTACKS: FAST PRIVACY VIOLATION IN *i3*

A. *Three basic attacks*

The attack model discussed here assumes that the adversary Eve knows *all* the msb_{id} of a private *id* belonging to some trigger, with the purpose of the attack being to discover the lsb_{id} of this private *id*¹. A brute-force attack, which would insert a trigger for every possible *id*, would require trigger insertions exponential relative to the number of bits. For the remainder of this section, the term *id* refers only to the lsb_{id} section of the *id*, unless explicitly stated otherwise.

1) *Exposing a single id*: First, let us consider the case when Eve wishes to uncover a single private *id*, without regard to the destination address associated with it. That is, the attack will reveal an *id* for which a trigger has been inserted, but without discovering or considering the owner of the trigger. For the purpose of presenting the manner in which this attack can be accomplished, I demonstrate it in the unlikely scenario that there is only a single, private trigger *id* with the appropriate msb_{id} , that was placed there by the user Bob. Once the technique is made clear, more practical scenarios will be discussed.

The attack consists of alternating trigger updates and message sending. Eve begins by placing a trigger *t*, containing her address, in the same *i3* node that holds the known range of *ids* starting with msb_{id} , and Bobs trigger. Eve then sends a sequence of messages to this *i3* node. The content of these messages is unimportant, and can be selected freely by Eve. By observing which messages matched the *id* in *t* and arrived back to her and which did not, Eve deduces information about the private *id* (id^{prvt}).

Before describing the attack in a formal way, let us begin with a small observation regarding the implications of longest-prefix matching. If the *id* in two triggers are different but both have the longest matching prefix with a given message *id*, then both will receive the message. This is the basis for our attack, as demonstrated in the following example. Assume Bob has placed a private trigger with private *id* 1010. Eve begins by guessing that this private *id* is 0000. She inserts a trigger with the 0000 as its *id*, and sends a message with 1111. Since 1010 has the longer matching prefix with 1111, the message will be redirected only to Bob. Once Eve is sure her trigger will not be activated by this message, she deduces that the first

¹The manner in which this information is obtained by an attacker is not discussed here. As will be seen shortly, this work demonstrates the manner in which allowing for longest prefix matching can present a security problem, and this is relevant only for the lsb_{id} .

bit in Bob's id is the opposite of the first bit in the id used in her trigger, since Bob must have a longer matching prefix.

In the next iteration of the attack, Eve modifies her trigger to use 1000 as the id , effectively flipping the first bit to match the first bit of Bob's id . She then sends the same message, with 1111 as the id . This time, both Eve and Bob will get the message, since they both have a matching prefix of length 1 with this message id . Once Eve gets the message, she can deduce that she and Bob match on the first *two* bits. The first bit is identical, as known from the last round. The second must be identical, because Eve knows that her trigger id does not match the message on the second bit, but still it arrived - so Bob does not have a longer matching prefix than Eve in their respective triggers. For the next round, Eve's trigger remains the same but the message sent will be with 1011 as the id . As can be seen from this example, in each round a single bit in the id used by Bob is revealed, and eventually all of the id bits are exposed.

I now present this attack, and additional attacks, in a formal manner. At round i , Eve sends a message m_i to id_i^m , and then waits to see if she receives a copy of the message. Based on the result of this check, Eve either changes the id in t or the id to which to send the message in the next round. When the address space of lsb_{id} is n bits long, the process ends after n rounds with id^{prvt} completely discovered.

```

FindID()
  initialize  $id_1^t, id_1^m$ 
  for each  $i = 1, 2, \dots, n$ 
    send a message  $m_i$  to  $id_i^m$ 
    if  $m_i$  does not arrive at the attacker, flip the  $i$ -th bit in  $id_i^t$ 
    if  $m_i$  arrives at the attacker, flip the  $i$ -th bit in  $id_i^m$ 
     $id_{i+1}^t = id_i^t, id_{i+1}^m = id_i^m$ 
  return  $id_{n+1}^t$ 

```

Fig. 1. Pseudo-code of FindID

The pseudo-code is presented in Figure 1. We start off with selecting a random id for the trigger (id_1^t), and its 1's-complement for the first message id (id_1^m). These two ids , therefore, do not agree on any bit value.

The correctness of the attack is proven next. In the following discussion, id_i^t represents the value of the trigger id between the $i-1$ -th and i -th round; The same goes for id_i^m . The $n+1$ -th round is therefore used to reflect the state of each of these ids at the end of the procedure.

Lemma 1: For all $1 \leq i \leq n+1$, id_i^m and id_i^t agree exactly on the first $i-1$ bits (and no others).

Proof: At $i=1$ id_1^t is the 1's-complement for id_1^m , so there is no bitwise agreement between them. At each round in the protocol a single bit is flipped in the i -th index of either the message or the trigger id , which causes the two ids to match on that bit. Since no other changes are made, the lemma is proven. ■

Lemma 2: For all $1 \leq i \leq n+1$, id_i^t and id^{prvt} agree at least on the first $i-1$ bits.

Proof: The base case of $i=1$ is trivial, as any two ids agree on at least 0 bits. Assume now that the lemma holds for $i \geq 1$. From Lemma 1 we know that id_i^m and id_i^t agree on the first $i-1$ bits and disagree on all the rest. If id_i^t and id^{prvt} agree on the i -th bit as well, both will receive the message, as they both have the same length of matching prefix with id_i^m - a length of $i-1$. If they disagree on the i -th bit, id^{prvt} has a longer matching prefix than id_i^t , so only the private trigger will be triggered and Eve will not receive m_i . If this occurs, the attack protocol flips the i -th bit to match the value in id^{prvt} , and by the beginning of the next round id_{i+1}^t and id^{prvt} agree on the first i bits. ■

Based on Lemma 2, it is clear that after $id_{n+1}^t = id^{prvt}$, so by the end of the algorithm the private id has been fully discovered. This is done in linear time ($O(n)$) - a considerable improvement over the exponential complexity of brute-force attacks.

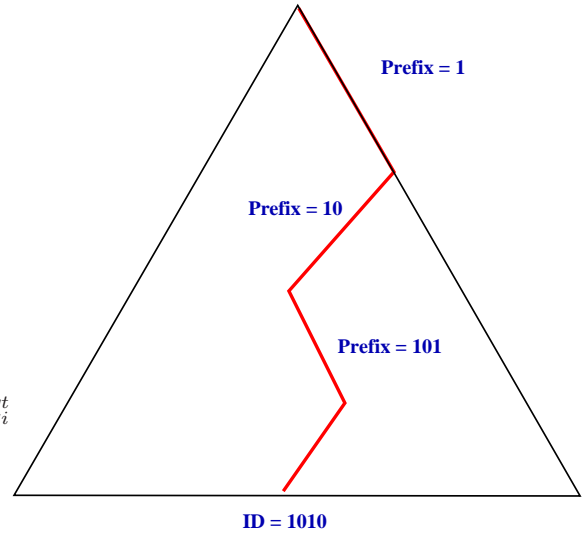


Fig. 2. Within a single "guess sequence", a private id is discovered.

In order to simplify the discussion of this technique, it is helpful to think of each id as a leaf node in a binary tree of height n . Each node at level k represents some prefix of length k bits, and all the ids with this prefix are descendants of this node. Each of the two edges exiting this node represent the $k+1$ bit, 0 or 1. The problem of discovering a private id is equivalent to discovering a leaf in the tree, by traversing discovered edges. For the rest of the paper, the above attack and its variations are referred to as "Lightning Attacks" both due to the lightning-like shape of a id traversal in a tree (Fig. 2) and the speed in which the attack is executed.

2) *Scanning (Discovering if any private id exists):* A useful tool for an attacker would be a fast technique to determine if private ids exist with a given prefix. This shall be referred to from here on as *scanning*. Assuming that no public id with prefix px exists ($|px| < n$), this can be achieved by the protocol in figure 3, with complexity of $O(1)$. In this pseudo-code, $a.b$ is a concatenation of strings, and \bar{a} is the 1's-compliment of bit string a .

```

IDExists(prefix  $px$ )
   $r = \text{rnd}(n - |px|)$  // random  $n - |px|$  bit sequence
  set  $id^t = px.r$  and send message  $m$  with  $id^m = px.\bar{r}$ 
  if  $m$  does not arrive at Eve - return true
  set  $id^t = px.\bar{r}$  and send message  $m$  with  $id^m = px.r$ 
  if  $m$  does not arrive at Eve - return true
  return false

```

Fig. 3. Pseudo-code of IDExists

Lemma 3: The algorithm returns *true* iff there is an *id* with the prefix px .

Proof: If there is a private *id* at the $i3$ node with the prefix px , it either agrees more with $px.r$ or with $px.\bar{r}$. Wlog assume it agrees more with $px.r$, so it agrees with $px.r$ for at least $|px| + 1$ bits. Since id^t agrees with id^m only on the first $|px|$ bits, in one of the two testing stages the message will not arrive at Eve.

On the other hand, if there is no private *id* in the $i3$ node, both test messages will arrive at Eve, since in both cases id^t will have the longest matching prefix (length $|px|$) with id^m . ■

3) *Listing all private ids:* A natural application of IDExists would be to scan an entire *id* space and locate all the private *ids* with a given prefix px . This is done using the recursive procedure described in Figure 4. Based on the correctness of IDExists, it is clear that this procedure will enumerate all the *ids* with prefix px . Since an *id* has n bits, and each node in the tree is traversed once only, the total running time of the algorithm is $O(n \cdot p)$, where p is the number of private triggers.

```

MapIDs (prefix  $px$ )
  If  $|px| = n$ 
    return  $px$  // ID found!
  if IDExists( $px.0$ )
     $ID = ID \cup \text{MapIDs}(px.0)$ 
  if IDExists( $px.1$ )
     $ID = ID \cup \text{MapIDs}(px.1)$ 
  return  $ID$ 

```

Fig. 4. Pseudo-code of MapIDs attack

B. Practicality, or: how to deal with public ids?

Returning to the basic attack, it is clear that FindID can discover an existing *id* but has no real control over which *id* is discovered. Applying it as-is in practical scenarios is unwise, therefore, since it might end up "discovering" a public *id*, information it is not interested in. In fact, a public *id* can be considered a "false lead", diverting attention from the private *ids*. As can be seen in Figure 5, a public *id* forces an attacker, that is interested to know if any private *ids* exist with the same prefix, to search in each sub-tree along the public *id* path. Without a public *id*, discovering if there is anything to look for at all would have cost $O(1)$ - a single call to IDExists - while with it $O(n)$ such calls will be required to cover the same

id space. The relative increase in complexity is less severe with regards to FindID, though as the number of public *ids* increases, so does the running time of these two functions. This section is devoted to assessing the complexity of the attack when public *ids* are present.

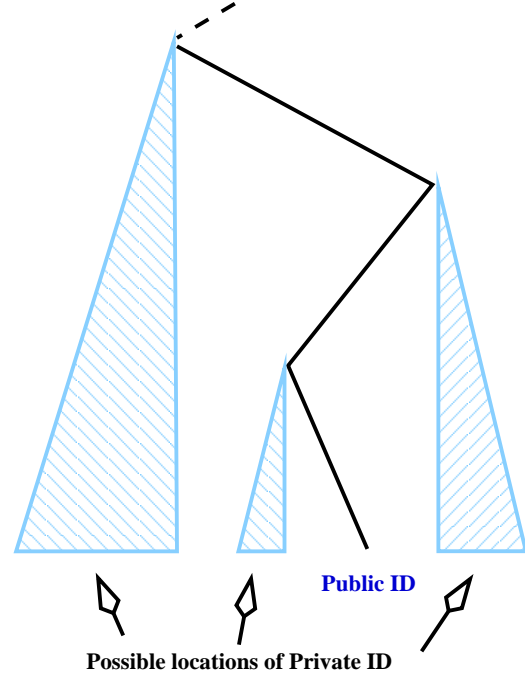


Fig. 5. Public *id* splits up a tree into several sub-trees, each of which needs to be checked separately.

The only step that can be taken by an attacker, in order to reduce complexity when facing public IDs, is to avoid placing queries for which the answer is known in advance. For example, a $\text{IDExists}(px)$ call should be placed only if there is no public *id* with a prefix px . This is because, as is plainly obvious, the function will return *true* because of the presence of the public *id*. In the following discussion, time-complexity is measured only as a function of required function calls (each of which uses a set number of operations).

Since when a node (prefix px) has two children there is no need to call IDExists on either of its children (since each will return *true*), we begin by assessing the number of such situations with a given set of public *ids*:

Lemma 4: When there are k public *ids*, there are exactly $k - 1$ nodes in the *id* tree with two children.

Proof: The tree of public *ids* has exactly k leaves and a single root, so $k - 1$ splits occur along the tree. ■

Since the number of nodes with two "public children nodes" is constant, the only factor controlling complexity is the number of nodes with a single public child, as each requires a call to IDExists to check a possible private *id* with a prefix differing only in the next bit.

Lemma 5: If there is a public *id* id^{pub} , with a prefix px that is unique among all public *ids*, discovering if there is a private *id* with the same prefix will require $n - |px|$ calls to IDExists.

Proof: For a given id^{pub} , denote each prefix of length k as px_k . For each prefix px_k , $|px| \leq k \leq n-1$, a single call to `IDExists` is required: if $px_{k+1} = px_k.b$, we need to call `IDExists(px_k.b)`, for which there is no known public id . ■

Lemma 6: Assume Eve wishes to discover if any private ids exist with a specific sequence of msb_{id} . If there are 2^k public ids , the worst case scenario for the attack sequence is $O(2^k \cdot (n-k))$ calls to `IDExists`.

Proof: Let us examine the following configuration of public ids : The k first bits of the i -th id will be the binary sequence with the value i , and the remaining $n-k$ bits will be some arbitrary sequence. In such a case, every call to `IDExists` with a prefix px shorter than $k+1$ bits will be redundant, as there is a known public id with this prefix. Therefore, Eve would be forced to examine each $k+1$ prefix separately, which alone would cost $O(2^k)$.

In addition to this increase, each of the public ids also has its last $n-k$ bits, that, according to Lemma 5, forces Eve to perform $n-k$ existence checks, so the total runtime is $O(2^k \cdot (n-k))$.

To conclude this proof, I show that this is the worst case scenario. If the configuration described here is not used for distributing the public ids , there exists a k -prefix px^- which does not belong to any public id . At the same time, due to the pigeonhole principle, there exist two public ids that share the same k -prefix px^+ , and denote by h the longest matching prefix these two share ($h \geq k$). Assume one of these were to change its prefix from px^+ to px^- , the effects on an attack procedure would be:

- Before the change, all ids with prefix px^- could be scanned with a single call to `IDExists`. After the change, at least $n-k$ calls are required in order to get the same information, according to Lemma 5.
- After the change, all ids with prefix px^+ can be scanned in $n-k$ calls. Before the change, there was an additional requirement for $n-h+1$ calls due to the second public id , according to Lemma 5.

Since $h \geq k$, it is clear that performing the change can only make things more difficult for Eve, proving that the described configuration is optimal. ■

Similar complexity bounds can be found for the other two attacks presented above. Specifically, it is clear that scanning the id space is the least expensive attack, as it does not require complete id resolving. With close analysis of the pseudo-code for each attack, it can be shown that the upper bound for the other attacks is no less than that proven in Lemma 6. More precise analysis of the upper bound on their complexity is outside the scope of this paper.

Secure- $i3$. Since Secure- $i3$ does not differ from standard- $i3$ in the handling of private ids , all of the above is relevant to Secure- $i3$ as well. In fact, the situation in Secure- $i3$ is somewhat worse than before. In Secure- $i3$ the system dedicates different $i3$ nodes for private and public triggers, so Eve will be able to perform all the attacks described here in best-case runtime. A possible solution to this, discussed in section IV, would be to apply the same restrictions used with

public triggers to private triggers. It is unclear from [1] and [2] if these limitations would cause problems, so additional solutions are presented in the next section as well.

IV. SUGGESTED COUNTERMEASURES

As is clear from our introduction, the weakness in $i3$ that allows Lightning attacks is inherent in the $i3$ design. Services such as anycast are available only due to the flexibility of longest-prefix-matching, which is the key to a Lightning attack. Thus, defenses against these attacks are likely to be found in intelligent policies adopted by the network and/or the end hosts. Several useful policies are presented here:

- 1) **Secure- $i3$.** Just as the restrictions placed on public triggers prevent eavesdropping to public communications, so can these restrictions solve the problem for private triggers. Specifically, as all end-host triggers are required to be *rnd*-constrained, placing a trigger that could redirect traffic to Eve would require discovering the random number selected by the user. Selecting an id and then discovering a number that would hash to it is, of course, unfeasible in general with one-way hash functions.

Though it is true that, as it is, the Lightning attacks for $i3$ all assumed that the msb_{id} section was already known, it still seems to be an additional leap to assume that this random number could be known by Eve. This is because (a) allowing such assumptions can lead to the complete irrelevance of cryptography, as we might as well assume that Eve knows the secret key of a specific cipher-text; and also because (b) ids in use might be stored in a known location in an $i3$ node, making it easier to obtain information regarding them, while the said random number is located in a single private trigger and at the end-host, both of which are probably much more difficult to locate.

In addition to all these problems, one might suggest that another problem is that the changes in the id required by the Lightning attacks could not be performed, due to the one-way hashing. This, however, is not the case, since the constrained section of the id anyhow requires complete matching, while the longest prefix matching takes place on a different, 64-bit section of the id . For more information on the trigger semantics, see [2].

- 2) **Public id distribution.** As is clear from section III-B, the presence of public ids can hinder the performance of lightning attacks and cause major increase in complexity. As the number of public ids grows, distributing them in an intelligent manner (Lemma 6) can have a significant impact on the speed in which Eve can determine anything regarding the existence of private triggers and their value.
- 3) **SPAM awareness.** While a Lightning attack is being mounted, messages that do not arrive at Eve are being received instead by a potential victim. As the attack stages progress, more and more messages like this will arrive at the target. A simple security measure would be

to change the private *id* when a certain number of such "spam" messages have arrived, since this may be a sign that the *id* is being targeted.

- 4) ***i3* reactive blocking.** The attack is based on Eve's ability to insert and modify triggers at will. However, a server may notice the attack pattern, remove problematic triggers and block the attacker's IP from future uses of the *i3* network. Attack patterns include a user sending itself messages (though this can be disguised by using two machines) and frequent changes to a trigger *id*.
- 5) **Limitations on *id* selection.** Similar to the ideas used in Secure-*i3*, several limitations on trigger insertion can be helpful in implementing the attack. For example, one might require that *id* selection in triggers be controlled by the *i3* network, and not by the user. One way in which this could be done is that a end-user would request from *i3* to connect to a certain public server, and *i3* in return assigns an *id* which is suitable. For different services different models would be required, but as long as the receiver does not have full control over the *id* to insert the attack is impossible.

V. CONCLUDING REMARKS

The contribution of this paper is in demonstrating the dangers in adopting novel addressing ideas: a simple and useful concept such as longest-prefix matching can be turned against the system and make it vulnerable to a variety of attacks. It is possible that the designers of *i3* had some inkling of this problem - after all, the attack is possible only under the important assumption that the msb_{id} portion of the address is known. This might put a serious damper on anyone trying to actually mount such an attack. However, as it is to be expected that *i3* will develop over time, and other overlay networks may utilize different concepts it incorporates, it is important to understand already now the implications of such an addressing approach, and this is what I set out to do here.

REFERENCES

- [1] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, *Internet Indirection Infrastructure*, Proceedings of ACM SIGCOMM, August, 2002
- [2] D. Adkins, K. Lakshminarayanan, Adrian Perrig, I. Stoica *Towards a More Functional and Secure Network Infrastructure*, Tech. Report, <http://i3.cs.berkeley.edu/publications/papers/csd-03-1242.pdf>
- [3] D. Adkins, K. Lakshminarayanan, Adrian Perrig, I. Stoica *Towards a More Functional and Secure Network Infrastructure (Brief Announcement)*, ACM PODC, St. John's, Newfoundland, Canada, 2004