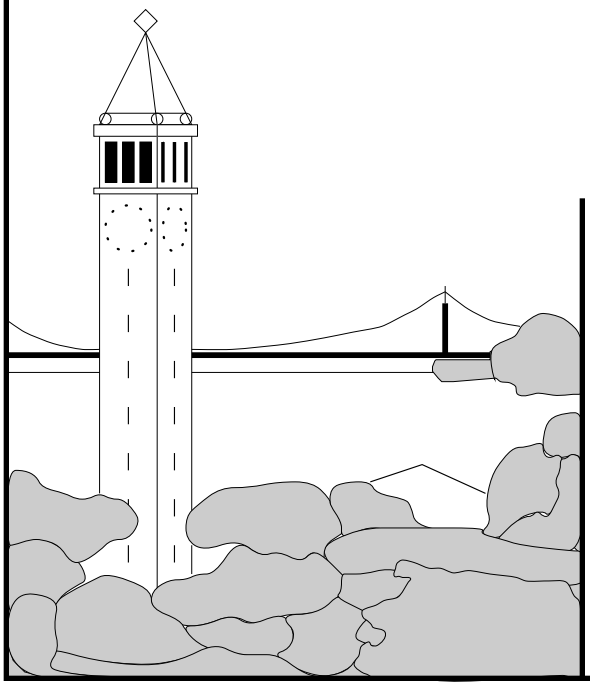
Towards a More Functional and Secure Network Infrastructure

Daniel Adkins *Karthik Lakshminarayanan* *Adrian Perrig* *Ion Stoica*
UC Berkeley *UC Berkeley* *CMU* *UC Berkeley*



Report No. UCB/CSD-03-1242

Computer Science Division (EECS)
University of California
Berkeley, California 94720

This research was supported by the NSF under Cooperative Agreement No. ANI-0225660, ITR Grant No. ANI-0085879, and Career Award No. ANI-0133811.

Towards a More Functional and Secure Network Infrastructure

Daniel Adkins Karthik Lakshminarayanan Adrian Perrig Ion Stoica
UC Berkeley UC Berkeley CMU UC Berkeley

Abstract

We propose an overlay network infrastructure that provides better protection against DoS attacks as well as more functionality than today's Internet. Our solution is based on three simple principles: (i) enabling end-hosts to communicate without revealing their IP address, (ii) giving end-hosts control to defend against Denial-of-Service (DoS) attacks at the overlay level, and (iii) making sure that the added functionality does not introduce vulnerabilities not present in the Internet. Our design is based on the Internet Indirection Infrastructure ($i3$).

1 Introduction

As the Internet evolves into a global communication infrastructure that encompasses our society, economy, and government, two of its limitations become increasingly apparent: the difficulty in deploying new services such as multicast and anycast, and the lack of security. While a plethora of solutions have been proposed to address the former limitation, the latter has received far less attention. For example, in today's Internet, an end-host can do little to defend against a flooding attack. As described in [15], a packet flood can easily isolate a server from the network for several hours. Worse yet, many of the solutions that aim to add functionality introduce security problems of their own, thus increasing the vulnerability of the end-hosts as well as that of the Internet infrastructure itself. For instance, supporting multicast can give an attacker more efficient ways to mount a DoS attack.

In general, there is a trade-off between adding more functionality[1] and achieving better security. Consider a set of primitives required to implement some functionality. As the functionality of a system increases, we usually need more primitives to implement them. The probability that none of the primitives has a security flaw decreases (security is often an all-or-nothing game, as a single flaw can render the entire system vulnerable). Thus, simplicity and less functionality (fewer primitives) usually contribute to increased security [29]. Contrary to this popular belief, in this paper, we show that one can build a communication infrastructure that provides both more functionality and improved security — namely protection against DoS attacks — than the Internet.

Traditionally, the main research thrust in preventing distributed DoS attacks has been on IP-level packet filtering, IP traceback and pushback techniques (as we review in Section 9). However, these approaches do not address the fundamental inability of an end-host[2] in today's Internet to stop a traffic flow directed towards it. Typical end-hosts in the Internet would benefit from having more flexible and fine-grained control over the traffic they receive.

The idea of using overlay networks to provide better defense against DoS attacks was first introduced in SOS [20], and later explored in Mayday [1] in more detail. However, these solutions assume that the set of clients accessing each of these servers is known in advance, and that client authorization is done out-of-band. They also assume that the IP infrastructure provides a basic filtering service near the target hosts. In this paper, we provide a solution that not only removes these limitations, but also provides more functionality than the Internet.

Our solution is based on an overlay network infrastructure built on top of IP. To achieve better protection against DoS attacks than the Internet, we follow three simple design principles. First, the overlay infrastructure should enable end-hosts to communicate without revealing their IP addresses. This would leave an attacker with no direct way of attacking a host (that communicates exclusively through the overlay) via IP. Second, the infrastructure should give end-hosts the ability to defend against attacks, possibly by stopping the attack in the infrastructure. This would make the infrastructure strictly better than the Internet, where an end-host can do little against a flooding attack directed at its IP address. Third, one should make sure that the added functionality doesn't open security holes not present in the Internet, such as allowing the attacker to use the infrastructure to mount new attacks against end-hosts.

To demonstrate our point, we re-design the Internet Indirection Infrastructure ($i3$) [31], an indirection-based overlay network infrastructure, with the above principles as our basis. There are two reasons for choosing $i3$. First, $i3$ provides more functionality than IP. In addition to unicast, $i3$ provides direct support for anycast, multicast, mobility and service composition. Second, as we shall see in Section 5, the flexibility of $i3$ helps in realizing the first two of the principles we have mentioned above. Though $i3$ provides all these advantages, $i3$ introduces some vulnerabilities not present in the Internet. Following our third design principle, we re-design

[1] A system that provides more functionality is a system that allows users to execute a larger variety of tasks.

[2] In this paper, we loosely use the term *end-host* to refer to any site connected to the Internet.

$i3$ (we call the new system *Secure-$i3$*) to eliminate these vulnerabilities without sacrificing functionality.

The paper is organized as follows. Section 2 enumerates our design principles for building a secure overlay. Section 3 gives a brief overview of $i3$, and Section 4 presents our assumptions. We describe the details of our solution in Section 5, and consider its impact on the original $i3$ functionality and performance in Section 6. Section 7 presents a evaluation of the overhead our solutions incur. Finally, we present related work and conclusions.

2 Design Principles for a Secure Overlay

In this section, we present three simple design principles to build an overlay communication infrastructure that provides better protection against flooding attacks than the Internet. While we realize that there could be other possible ways of designing secure systems on top of an IP network, we believe that these design principles are general enough to be applicable in designing other secure overlay systems.

Hide IP address. The overlay infrastructure should enable end-hosts to communicate without revealing their IP addresses. This would leave an attacker with no direct way of learning the IP address of an end-host that communicates exclusively through the overlay. If this principle is violated, the overlay network cannot provide better protection than the underlying IP network, as the attacker could simply use the IP address of the victim to attack it.

Give end-hosts control to defend against attacks. Hiding end-host IP addresses would amount to nothing if the infrastructure itself does not provide better protection against flooding attacks than the underlying IP network. We believe that the key to achieving this is to give end-hosts the *ability* to stop an attack directed at them or route around parts of the infrastructure that are under attack. Since end-hosts are, in general, in the best position to detect if they are under attack, they would greatly benefit from the ability to stop the attack *before* the traffic arrives at their ingress link, i.e. the link that connects them to the Internet. Indeed, if the packets of the attack do arrive at the ingress link, there is nothing that the victim can do against the attack.

Note that this principle argues that the popular belief that *more functionality hurts security* is not necessarily the case. In particular, careful choice of network primitives which enable end-hosts to have more control on data forwarding and routing can improve security. Of course, we need to make sure that solving a security problem will not create new ones, and this is precisely what the last principle addresses.

Avoid new vulnerabilities. The added functionality should not introduce new security vulnerabilities not present in today's Internet. A network infrastructure that provides better protection against flooding attacks but creates new security problems would not be the most desirable outcome of our research. While this principle might seem obvious, we feel that it is important to emphasize it, as too often security solutions fail to study the potential side effects carefully.

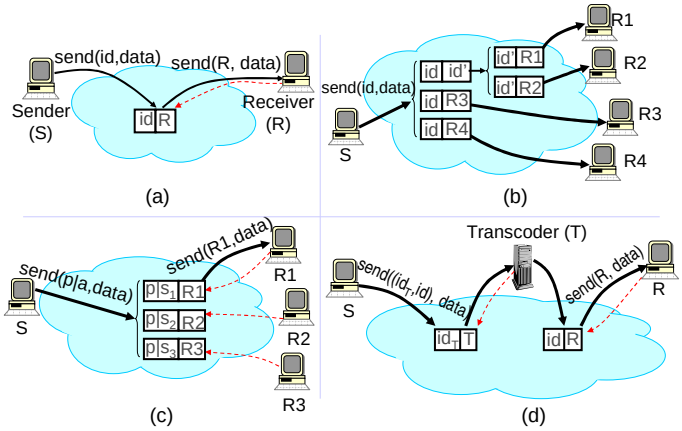


Figure 1: Basic communication primitives in $i3$: (a) unicast, (b) multicast, (c) anycast, and (d) service composition.

We do not expect one to formally *prove* that the proposed system does not introduce new security problems. This would require a formal description of the behavior and the properties of the system, which is very hard to do in practice (e.g., how would one formally describe the behavior of the Internet?). Instead, we expect one to study this issue in reasonable depth and provide qualitative arguments supporting the fact that new security vulnerabilities are not introduced. For example, Mayday [1] is one of the recent works which provides a discussion of attacks and defenses on the proposed architecture in reasonable depth.

3 Overview: $i3$

To develop a solution that can provide more functionality and better protection against flooding attacks than today's Internet, we start with the Internet Indirection Infrastructure ($i3$). As described in [31], $i3$ already provides more functionality than the Internet. In addition, as we shall see in Section 5, the flexibility of $i3$ provides a good starting point for realizing the first two principles discussed in Section 2.

At its roots, $i3$ provides indirection, that is, it decouples the act of sending a packet from the act of receiving it [31]. There are two basic operations in $i3$: sources send packets to a logical *identifier* and receivers express interest in packets by inserting a *trigger* into the network (Figure 1(a)). Packets are of the form $(id, data)$ and triggers are of the form $(id, addr)$, where $addr$ is either an identifier or an IP address. Given a packet $(id, data)$, $i3$ will search for a trigger $(id, addr)$ and forward $data$ to $addr$. Receivers refresh the triggers that they insert as long as they desire to receive packets sent to the identifier that the trigger corresponds to (soft-state approach). In addition, $i3$ supports an operation to remove triggers.

Identifiers in packets are matched with those in triggers using longest prefix matching. To reduce the probability of accidental collision, two IDs match only if they share a prefix with a length of at least 128 bits. $i3$ is implemented as an overlay network of nodes that store triggers and forward packets. Identifiers are mapped to $i3$ nodes using a distributed lookup service such as Chord [32]. A trigger is stored at the node that is responsible for its identifier in accordance with the Chord lookup protocol. Similarly, packets are routed to the appropriate node by Chord. The mapping procedure ensures that all IDs which share the same 128-bit prefix are mapped on the same node; thus, the longest prefix matching operation is performed locally.

$i3$ provides direct support for a variety of communication abstractions, including mobility, multicast, *anycast* and service composition. A *mobile* host that changes its address from R to R' can preserve the end-to-end connectivity by updating its trigger from (id, R) to (id, R').

Creating a *multicast* group is equivalent to having all members of the group register triggers with the same identifier. There is no difference between unicast and multicast in $i3$, and an application can switch between the two on the fly. Figure 1(b) shows a two level multicast hierarchy. Conceptually, triggers can be thought of as *pointers* that point either to receivers or to other triggers.

All hosts in an anycast group maintain triggers that have identical 128-bit prefixes (Figure 1(c)). Packets are delivered to the group member that has the trigger with the longest matching identifier. This scheme can be used to implement applications such as server selection.

Finally, $i3$ can provide *service composition*, that is, allow either the sender or the receiver to forward packets through intermediate points in the network. One way to achieve this is to replace the packet ID with a stack of IDs. Forwarding such a packet is similar to source routing in IP. Figure 1(d) shows how a sender S can use a stack of IDs, $[id_T, id]$, to forward the packet through a transcoder T. A receiver can control packet forwarding by replacing the second field of its trigger with a stack that describes the forwarding path [31].

4 Assumptions

In this section, we present our assumptions about the underlying system, how the end-hosts are supposed to operate and the threat model of attackers.

Underlying Infrastructure:

1. $i3$ depends on a peer-to-peer lookup service such as Chord. We assume that the lookup service is itself robust and reliable, and that attackers cannot compromise the lookup service. For related work on security of peer-to-peer lookup services, see [6, 30].
2. There is no additional functionality (such as packet filtering) that the IP network provides.
3. $i3$ nodes are infrastructure resources, and hence have significantly more bandwidth than "typical" end-hosts in the Internet.
4. End-hosts have a secure connection providing confidentiality and authenticity to the first $i3$ node, e.g., a connection secured through IPSec [18, 19].

Operation of end-hosts:

1. End-hosts (servers) that expect connections from arbitrary end-hosts (clients) must have triggers whose identifiers are well known. These triggers are called *public* triggers. Once a client contacts a server through its public trigger, they can exchange a pair of identifiers which they use for the remainder of the communication. Triggers corresponding to these identifiers are referred to as *private* triggers [3] Note that the use of public triggers as initial rendezvous points gives end-hosts complete freedom in picking the IDs of their private triggers.
2. End-hosts do not reveal their IP addresses, and for all practical proposes, the address of an end-host is difficult to guess. Note that even if the attacker learns the IP address of a target end-host by other means, as long as the target can obtain a different IP address (using some mechanism offered by the ISP, such as DHCP), it can defend itself against a potential DoS attack. $i3$ allows the target end-host to maintain the ongoing connections as long as it updates its triggers to point to its new address.
3. End-hosts do not reveal their private IDs, and it is infeasible for an attacker to guess a private ID, since IDs are 256 bits long.

Attacker threat model:

1. Since we assume a secure connection from a client to the first $i3$ node, an attacker cannot eavesdrop on any traffic between end-hosts and $i3$ nodes. Also, since $i3$ nodes are part of an infrastructure, we assume that eavesdropping on an $i3$ node at the IP level is as hard as eavesdropping on an IP router. Note that this assumption does not preclude eavesdropping at the $i3$ level.
2. An attacker cannot compromise $i3$ nodes. The security of a particular $i3$ node is a software problem which we do not address in this paper.

5 Solution

The goal of this paper is to demonstrate that it is possible to build an overlay infrastructure that provides both more functionality and enhanced robustness against DoS attacks than the Internet.

To achieve this goal, we re-design $i3$ which already provides significantly more functionality and flexibility than the Internet (see Section 3). Next, we discuss how $i3$ can be used to

[3] End-hosts that do not need to be contacted by arbitrary end-hosts don't need to maintain public trigger.

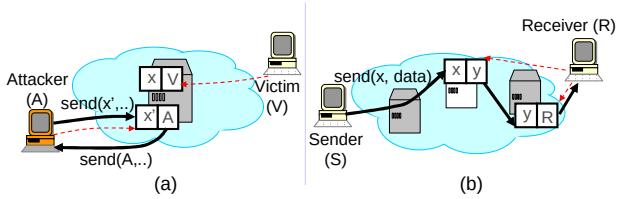


Figure 2: (a) An attacker learning the IP address of the $i3$ node that stores the victim's public trigger (x, V) (x and x' have the same prefix). (b) Possible solution: every $i3$ node responsible for public IDs won't allow triggers pointing to end-hosts and won't allow end-hosts to cache its IP address.

realize the first two design principles presented in Section 2, and then present a set of techniques to realize the last principle. For the sake of clarity, we refer to the modified, secure form of $i3$ as *Secure-*$i3$.

5.1 Hide IP Addresses

If an overlay communication infrastructure is to provide better protection against flooding attacks than the underlying Internet, the infrastructure should enable end-hosts to communicate without revealing their IP addresses. Otherwise, an attacker can learn a victim's IP address and use it to mount a flooding attack via IP (recall that we assume that there is no filtering done at the IP level). Decoupling the address of an end-host E from its identity by a level of indirection would then enable other end-hosts to contact E. This is exactly what $i3$ provides, that is, it allows end-hosts to communicate exclusively through IDs rather than IP addresses (see Figure 1). Obviously, we assume that the node does not reveal its IP address, for instance using DNS.

However, while there is no direct way for an attacker to learn the IP address of a victim in *Secure-*$i3$, the attacker can easily learn the IP address of the node that stores the victim's public trigger. As illustrated in Figure 2(a), an attacker A can insert a trigger (x', A) pointing to itself and send a packet to that trigger. As a result, the packet will be sent back to the attacker via IP thus revealing the IP address of the node storing (x', A). By choosing x' to have the same prefix as the victim's public [4] ID x, the attacker can easily learn the IP address of that node. Then, the attacker can use IP flooding to attack the $i3$ node storing the victim's ID. This can render the victim unreachable for new clients, as the new clients would not be able to contact the victim via its public ID.

Our solution to this problem is to make sure that nodes storing public triggers will never communicate with end-hosts via IP. Such a node will store only triggers of the form (x, y) where both x and y are IDs. Thus, an end-host R that advertises a public ID x needs to insert a pair of triggers (x, y) and (y, R), where y is a private ID (see Figure 2(b)). This way the end-host will receive the packets sent to x from the node storing ID y instead of the node storing x. To implement this solution we can dedicate a region of the ID space to public IDs, and then make sure that nodes that store IDs in this region will not allow triggers pointing to end-hosts.

With the aforementioned scheme, the attacker can still learn the IP address of an $i3$ node storing private triggers. However, this is not a serious concern since private triggers are secret, and therefore there is no direct way for the attacker to learn where the victim's private triggers are located. To summarize, the attacker has no effective way of mounting an IP-level flooding attack against a victim: it cannot learn the IP address of the victim or of the node storing the victim's public trigger, and it has no effective means of identifying the node storing the victim's private trigger. However, attacking at the $i3$ level is still possible, which is what we address next.

5.2 Give End-hosts Control Against Attacks

In the previous section, we have illustrated mechanisms which preclude an attacker from flooding a victim at the IP level. Of course, all of these would amount to nothing if the overlay infrastructure cannot provide better mechanisms to defend against the attack.

We believe that the key to providing a better defense for the end-hosts against flooding attacks is by giving end-hosts the *ability to defend themselves against the attack*. This can be achieved by empowering end-hosts with more control so that they can stop any flow addressed to them. Since end-hosts are usually in a better position to detect flooding attacks than the infrastructure is, we believe that they should also have the capability to defend themselves. This is because end-hosts have more information (than the infrastructure does) about the content and semantics of the traffic they expect to receive. For example, if CNN receives hundred times the usual traffic in the absence of any important event or news, then CNN might classify it as an attack.

In the remainder of this section, we illustrate how end-hosts can defend against flooding attacks using the flexibility of $i3$. In particular, we show how end-hosts can (i) stop attacks on private triggers, (ii) dilute or slow down attacks on public triggers, (iii) evade attacks on $i3$ servers, and (iv) provide multicast access control.

5.2.1 Stop the Attack

An end-host under attack can stop the traffic received through one of its triggers by simply removing that trigger. Since the trigger is stored at an $i3$ node in the infrastructure, removing the trigger will completely eliminate the traffic received by the end-host through that trigger.

In contrast, in today's Internet, a victim can do nothing against a flooding attack directed at its IP address. [5] While

[4] In $i3$, triggers with the same prefix are stored at the same $i3$ node [31].

[5] In some cases, customers can call up their ISP, but this is usu-

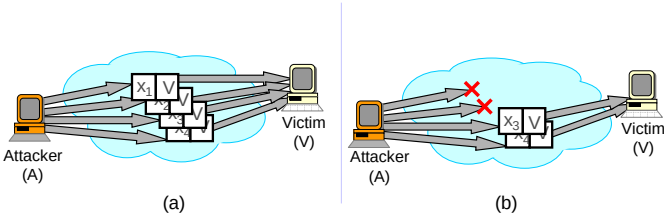


Figure 3: (a) Flooding attack via victim's public triggers. (b) Dilute the attack by dropping two of the four public triggers.

it is theoretically possible to engineer the network to detect and defend against such an attack (e.g., by implementing IP traceback [4, 11, 28] and *pushback* [16, 21]), there are a few problems with this approach. First, more intelligence in the network increases network complexity and makes it harder to manage. Second, when a new type of attack emerges, it will take some time to engineer the network to detect it, and during this time end-hosts will remain vulnerable to the attack.

We note that removing a trigger is particularly useful in the case of private triggers, as this would stop the traffic from the attacker without affecting legitimate traffic. Recall that in $i3$, each receiver communicates with different senders using different private triggers that are kept secret by the senders. Hence, the attacker will not be aware of the private triggers of other legitimate senders communicating with the receiver.

However, this technique would not work well for public triggers. Recall that servers that want to be contacted by arbitrary clients need to maintain a public trigger in $i3$. In this case, an attacker can use the server's public trigger to mount an attack. Of course, the server can stop the attack by dropping the public trigger, but this would prevent new clients from connecting to the server. However, note that even if the server becomes unreachable for new clients, the current clients can still communicate with the server via their private triggers. In contrast, in the Internet a flooding attack on the server will disrupt ongoing connections as well.

In another example, removing the public trigger of a compromised server can avoid *collateral damages*. Consider a web-server S run by Bank of America sharing an access link with a machine M that is connected to its ATM network. M is involved only in private communication with the ATMs, whereas S expects clients to connect to it, and hence has a public trigger. In the event of an attack on S, removing public triggers of S would allow one to access the ATM as before, at the cost of completely shutting down a less critical service.

5.2.2 *Dilute the Attack*

As discussed in the previous section, it might be impossible for a server to stop the attack totally. As long as the server has a public trigger, the attacker can abuse it.

ally a very time-consuming process.

We propose a simple solution to alleviate this problem. The main idea is to give a server the ability to drop a fraction f of the total traffic destined for its triggers. The hope is that by doing so, a server under attack will also drop a fraction f of the offending traffic. Of course, random dropping on traffic destined for public triggers would also hurt legitimate clients, since clients' requests to connect through these triggers will fail. However, because a client chooses a new random trigger every time, it will succeed after $1/(1-f)$ tries on an average. Thus, this approach allows a server to degrade its service gracefully by trading between the extent of offending traffic dropped and the time a legitimate client takes to connect.

Implementing this technique is easy. A server maintains n public triggers, and each client is expected to randomly choose one of these triggers when it attempts to connect. In the face of a flooding attack, the server gradually drops the public triggers that receive the most traffic. The reason behind this strategy is that legitimate clients are expected to generate uniform load (as they select public triggers randomly), and thus the triggers receiving a disproportionate amount of traffic are those that are most likely to carry the attacker's traffic. By removing the highest loaded $m(< n)$ public triggers, the server will shed at least a fraction $f = m/n$ of the attacker's traffic. It is worth noting that removing public triggers will *not* affect clients that are already communicating with the server through their private triggers.

We make two other points. First, to implement a similar service at the IP layer, one would need to protect the traffic of ongoing connections from other traffic (this includes traffic to establish new connections, e.g., SYN packets, as well as other traffic arriving at the end-host). This would require the router (of the service provider) before the ingress link to implement packet classification and bandwidth management, and to allow end-hosts to push filters on a per source basis. Rate-limiting SYNs as it is done today by some ISPs is not sufficient as an attacker can still affect the ongoing connections by sending non-SYN packets.

Second, an intelligent attacker can learn which public triggers are alive—by listening for replies corresponding to each trigger it contacts—and then can redirect all its traffic to those triggers. To get around this problem, the server can rapidly change the subset of m active public triggers (out of the original set of n triggers), using a memoryless process. If the average lifetime of a trigger is on the same order of magnitude as the time it takes the attacker/client to receive a reply from server, the attacker won't be able to infer from the reply whether the corresponding trigger is still alive.[6] To reduce the frequency with which the server would need to

[6]Let T be the time after which a client receives a reply. If the mean trigger lifetime is $T \ln 2$, the attacker cannot infer whether the trigger is still alive or not upon receiving the reply.

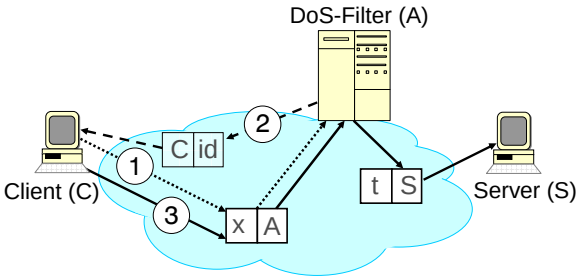


Figure 4: Slowing down a DoS attack on a public trigger.

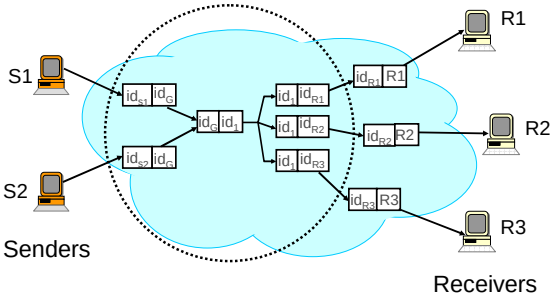


Figure 5: Multicast Access Control

change its public triggers, one possibility would be to delay each reply (e.g., by a few seconds). Finally, note that since a legitimate client selects a public trigger randomly, this scheme will not affect the number of tries a client has to perform to connect to the server.

5.2.3 *Slow Down the Attack*

A solution to slow down the attack on a public trigger is to use a powerful third-party server A (called DoS-filter server) that shields the server S from the attack. In this solution, we assume that server A is much more powerful than S and it can sustain a DoS attack that will otherwise cripple S.

The main idea behind this scheme is to give a *cryptographic puzzle* to the client that has to be solved by the client in order to contact the server S. This scheme is illustrated in Figure 4. Server S stores a private trigger (t, S) where t is known only by the DoS-filter server A. In turn, A inserts a public trigger (x, A) and advertises x as being the public ID of server S. Related work about the use of crypto puzzles is mentioned in Section 9.

When a client C wants to contact S, C sends a message to ID x. This message is in turn delivered to A (step (1) in Figure 4). Upon receiving this message, A sends a cryptographic puzzle back to client C via the private trigger (id, C) that was inserted by the client C (step (2) in Figure 4). Client C then solves the puzzle and sends the answer back to ID x. Upon receiving this message, A verifies that C has solved the puzzle and forwards the packet to ID t (step (2) in Figure 4). Finally, the packet is delivered to server S which allocates a private trigger for client C as before.

To avoid replay attacks, A will respond to each message with a unique puzzle. Once it sends the puzzle, A stores it and waits for a reply. On receiving the reply (i.e., the solution to the puzzle), A removes the puzzle. Of course, if A does not receive a reply within a specific period of time, A will remove the puzzle from its list.

We also note that these schemes would be adopted by servers *only when under attack*. Hence, under normal operation, clients will not have the burden of either solving crypto puzzles or trying multiple times to reach a server.

5.2.4 *Evade the attack*

If the attack is directed towards a particular $i3$ node, then end-hosts can route "around" the attacked $i3$ node by choosing a different trigger. For comparison, if a router is attacked in the Internet, routing around the attack is not under the control of end-hosts.

5.2.5 *Multicast Access Control*

One of the main security concerns of IP multicast is access control. This is because there is only one IP multicast address which is used both by senders to send traffic to, and by receivers to subscribe to. This implies that any receiver can potentially send multicast traffic to the entire group, which is clearly not desirable.

Secure-i3 can avoid this problem by having different IDs for senders and receivers of the multicast group. In practice, the management of the multicast group, i.e. the job of assigning IDs for subscribing to the group and sending to the group, can be done by a private entity or a third party.

For example, in Figure 5, there are two senders and three receivers. The multicast service provider would construct the tree of triggers which lie inside the circle shown in the figure. It would then provide id_{S1} and id_{S2} respectively to senders S_1 and S_2, and id_{R1}, id_{R2} and id_{R3} to receivers R_1, R_2 and R_3. Receiver R_i would be responsible for refreshing the trigger (id_{Ri}, R_i). The receivers are not aware of the tree topology, but they would still receive the contents from S_1 and S_2. Furthermore, only sender S_i (and of course, the multicast service provider) is aware of the private ID id_{Si} — thus, id_{Si} is the secret key for access control. This decoupling allows one to perform fine-grained access control by not publishing the sender's triggers. Clearly, sender S_i can violate the mechanism by sending id_{Si} to a larger set of senders. But, this is similar to someone letting out their password in public, and is outside the scope of our solution. Moreover, violation of the sending rate to any of the IDs id_{Si} of the multicast group is similar to a flooding attack on private triggers and can be dealt with by removing the trigger (id_{Si}, G). The multicast service provider, P, can easily detect flooding by inserting the triggers (id_{Si}, P) for each sender S_i, and monitoring the sending rates of each of the senders.

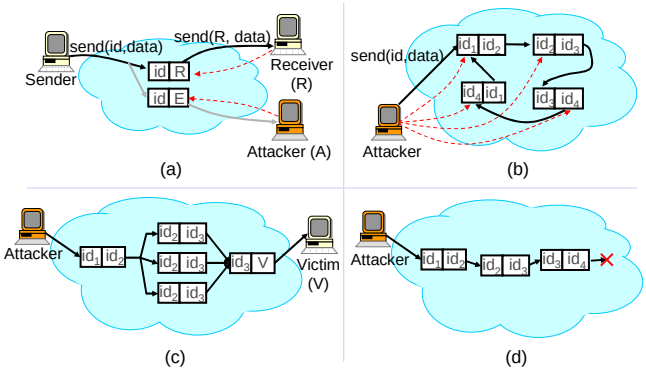


Figure 6: Security problems in $i3$: (a) eavesdropping; (b) loop, (c) confluence, and (d) dead-end.

Note that this is a very efficient non-cryptographic solution to the problem of access control for multicast groups. So far, the majority of research has considered cryptographic solutions [33, 34] to address this problem. Since we assume that eavesdropping within the *Secure-$i3$* infrastructure is hard, our non-cryptographic solution to access control also allows simple member addition and deletion, and achieves forward and backward secrecy (i.e., a joining member cannot receive content sent before it joined, a leaving member cannot receive content sent after it was removed).

5.3 Avoid New Vulnerabilities

Thus far, we have shown that, with minimal changes, $i3$ can realize the first two principles presented in Section 2. We now systematically analyze the potential ways by which an attacker can abuse the flexibility of $i3$ to mount an attack on the end-hosts or on the infrastructure itself. We do this by exhaustively enumerating all the possible ways of invoking the "primitives" offered by $i3$. In the process, as suggested by the last principle, we provide qualitative arguments supporting the fact we introduce no new vulnerabilities.

At the $i3$ level, end-hosts can perform only one of the two operations: (i) insert/remove a trigger (ii) send a data packet to an ID. Apart from removal of valid triggers, malicious operations on the control path can be performed only using trigger insertions, and this can cause new attacks. Sending a data packet to an ID can be used only for packet floods, possibly on topologies constructed by using the flexibility of $i3$.

We shall first see what an attacker can do by inserting arbitrary triggers. Triggers in $i3$ can point either to end-hosts, or to other IDs.

5.3.1 *Attacks using triggers pointing to end-hosts*

Assume a legitimate end-host R maintains trigger (id, R) in $i3$. The only way an attacker can directly attack end-host R is by abusing either id or R.

1. *Abusing id:* The only possibility for an attacker A to abuse R's id is to insert (id, X), where X is an address different from R. In such a scenario, every packet sent to R (through id) is transparently forwarded to X as well.

Eavesdropping. If the attacker sets X to its own address A, the attacker will eavesdrop on all the traffic to R (see Figure 6(a)). Thus, eavesdropping is much easier in $i3$ than in the Internet, where the eavesdropper has to be on the same LAN or on the path to the victim.

Impersonation. A minor variant of the above attack involves an attacker A making end-host R drop its public trigger (x, R) by flooding it (as in Section 5.2.2). Then, if A inserts (x, A), A can not only eavesdrop on R's traffic but also actively respond to it, thus impersonating R.

The attacker's ability to eavesdrop is inherent in $i3$'s architecture. Packets are multicast when several triggers have the same ID, and furthermore this is transparent to both the senders and the receivers. Before moving ahead, we ask a basic question: Would disallowing multiple triggers with the same ID solve our problem? The answer is "no". Though this solves the problem of eavesdropping, it does not solve the problem of impersonation. Actually, it may worsen the situation as R will not be able to insert its own trigger.

2. *Abusing R:* The attacker can abuse R by inserting a trigger (id', R), and sign-up the victim R for high-bandwidth streams. Note that this attack is not possible under our assumptions, since this would require the attacker to know the IP address of R. However, we note that the technique presented Section 5.3.6, covers even the case of an attacker knowing the IP address of R.

5.3.2 *Attacks using triggers pointing to IDs*

With triggers of the form (id, id'), an attacker can do two things: (i) construct arbitrary topologies, and (ii) arbitrarily interconnect existing topologies.

1. Construct arbitrary topologies. The attacker may construct topologies to multiply the attack traffic and direct it at one end-host using a trigger of the form (id, R). Below we give three examples of such undesirable topologies.

Loops. An attacker may form a loop by inserting triggers $(id_1, id_2), (id_2, id_3), \ldots, (id_{(n-1)}, id_n), (id_n, id_1)$ (see Figure 6(b)). Packets sent to any of the IDs of the loop would indefinitely cycle around and consume infrastructure resources.

Confluence. An attacker can construct a confluence as shown in Figure 6(c)). In the event of a confluence formation, packets are first replicated as they would be in a multicast tree. Then, instead of being delivered to separate end-hosts, all the replicated packets converge to eventually overwhelm an end-host via its public trigger.

Dead-ends. An attacker can construct a chain of triggers or a portion of a tree which does not ultimately point to a valid end-host (see Figure 6(d)). A data packet sent on such a topology would be routed through the chain of triggers only to be dropped when it reaches the dead-end. Such a packet will consume infrastructure resources without doing any useful work.

2. Interconnecting existing topologies. By inserting arbitrary triggers, an attacker can channel the traffic between different applications (**reflection**). For instance, an attacker can sign-up a victim to a high bandwidth traffic stream by inserting a trigger (id', id), where id is the public ID of the victim. In another example, the attacker can cause BBC to receive all the connection requests destined for CNN by simply inserting a trigger (id_{CNN}, id_{BBC}).

5.3.3 Attacks by sending data to arbitrary topologies

Node confluence. The flexibility that $i3$ offers in placing a trigger (by choosing a prefix for the ID) can be used by an attacker to store all the leaf triggers of a multicast tree at the target $i3$ node. As a result, for every packet sent by the attacker, the target $i3$ node will be bombarded with n duplicates, where n represents the number of leaf triggers (i.e., receivers) of the multicast tree.

Of course, one of the simplest things an attacker can do to attack the infrastructure is to build a larger random multicast tree and blast packets into the infrastructure.

We now propose three techniques to address the above problems: trigger constraints, pushback, and trigger challenges.

5.3.4 Technique 1: Constrained Triggers

One observation while studying the attacks that can be mounted on $i3$ is that most of the attacks are due to the fact that end-hosts can insert triggers with arbitrary IDs. Proactively eliminating the possibility these attacks by imposing some constraints on the triggers is desirable. The challenge then, is to define such constraints without sacrificing $i3$ functionality.

We address this challenge by enforcing a constraint on the structure of triggers, i.e., for a trigger (x, y), the choice of x constrains the choice of y or vice-versa. We divide the 256-bit ID into three fields: a 64-bit prefix, a 128-bit key, and a 64-bit suffix (Figure 7). This preserves user's flexibility to choose the prefix (choice of trigger placement), and the suffix (to enable anycast) while restricting the key using some constraints. A *Secure-$i3$* trigger with identifier x matches a packet with identifier y iff: (a) the prefix and key of x exactly match the prefix and key of y, and (b) there is no trigger that has a longer prefix match with y than x.

We now explain the constraints on trigger structure and rules on trigger insertion.

1. *Trigger Constraints:* Only triggers of the form (x, y), where either $y.key = h_r(x)$ or $x.key = h_l(y)$ [7], are allowed in *Secure-$i3$*. The reason we choose different hash functions h_l and h_r is to avoid cycles (see Lemma 1). h_l and h_r are one-way cryptographic hash functions mapping 256-bit strings to 128-bit strings. h_l and h_r are publicly known functions, and hence any $i3$ node or host can check and enforce the constraints. If $y.key = h_r(x)$, we say that trigger (x, y) is $right$-constrained (or r-constrained); otherwise, we say that it is $left$-constrained (or l-constrained). Intuitively, we can see that these constraints will disallow construction of arbitrary topologies in *Secure-$i3$*.
2. *End-host constraints:* If a trigger (x, y) points to an end-host we use the fields *y.prefix* and *y.suffix* to encode the end-host address. Furthermore, if an l-constrained trigger (x, y) points to an end-host, we use only *y.key* to constrain *x.key*, i.e., instead of computing *x.key* as $h_l(y)$ we compute it as $h_l(y.key)$. As we shall see, ignoring *y.prefix* and *y.suffix* when computing $h_l(y)$ allows us to preserve support for anycast and mobility.
3. *Constraints at the $i3$ nodes:* Public $i3$ nodes allow only l-constrained triggers. As we shall see, this constraint precludes eavesdropping and impersonation of end-hosts. It also does not allow attackers to attack public $i3$ nodes by leveraging multicast functionality. In addition, as discussed in Section 5.1, public $i3$ nodes do not allow triggers pointing to end-hosts.

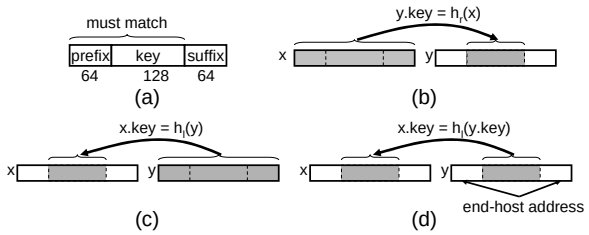


Figure 7: (a) *Secure-$i3$* identifier. (b) r-constrained trigger (x, y). (c) l-constrained trigger when y is an ID. (d) l-constrained trigger when y contains an end-host address.

We use the following observations to argue that constraining triggers eliminate many of the problems we discussed before. Secure one-way hash functions, such as MD5 [27], provide strong collision resistance, which means that it is computationally infeasible to find two distinct values x and x', such that $h(x) = h(x')$. Based on the birthday paradox, finding a collision requires approximately $\sqrt{2n}$ operations. Since $n = 2^{128}$ in our case, finding a collision would require on the order of $2^{64.5}$ operations. More generally, assuming that our cryptographic one-way hash function behaves as a random mapping, finding a cycle requires on the order of $\sqrt{\pi n/2}$ operations [14]. Since $n = 2^{128}$ in our case, find-

[7] If it is clear from the context, we use notations $y = h_r(x)$ and $x = h_l(y)$ instead of $y.key = h_r(x)$ and $x.key = h_l(y)$.

ing a cycle would require on the order of $2^{64.3}$ operations. While computing 2^{64} operations is not out of the reach with current technology, this still requires a substantial effort — consider that breaking a DES key only requires 2^{56} operations, and breaking a 512 bit RSA key requires on the order of 2^{50} operations [9].

Next, we show how constrained triggers can be used to solve three of the problems discussed in this section: eavesdropping, impersonation, and forming undesirable topologies.

To avoid *eavesdropping* and *impersonation*, an end-host R inserts only public triggers that are l-constrained. For trigger (x, y), the end-host uses the fields $y.prefix$ and $y.suffix$ to store its address, and uses $y.key$ to store a secret value that is known only to R.

To eavesdrop the traffic destined to ID x, an attacker would need to insert a trigger (x, y') that points to itself. (i) If (x, y') is an r-constrained trigger, the insertion will fail because the public *Secure-i3* node storing (x, y) would not accept r-constrained triggers (Refer the last constraint). (ii) On the other hand, if the attacker inserts an l-constrained trigger, the attacker would need to find $y.key$. This is infeasible as it reduces to inverting h_l, which is a one-way function.

Lemma 1. *With constrained triggers, it is computationally infeasible to construct any topology other than trees.*

Proof. Let T be a topology formed using constrained triggers. Define G_u to be the undirected graph such that (i) every ID in a trigger corresponds to a vertex in G_u (ii) $\exists$ edge e between two IDs x and y iff (x, y) or (y, x) is a trigger in T. Next, define G_d as the directed tree formed by assigning directions to the edges of G_u in the direction of the constraint that the trigger corresponding to that edge satisfies. For example. a trigger $(x, y = h_r(x))$ in T would have an arrow from x to y in G_d. Note that G_u is the underlying graph of the directed graph G_d.

The proof is by contradiction. Let T be a topology of triggers that is not a tree. From above definitions, G_u has a cycle, i.e. the underlying graph of G_d has a cycle. Two cases arise.

Case (i) G_d has at least one vertex with in-degree 2. This implies that $\exists x, y$, such that $h_i(x) = h_j(y)$, for $h_i, h_j \in h_l, h_r$, such that $x \neq y$ or $h_i \neq h_j$. For this to be infeasible, we require $h_i \neq h_j$, otherwise for $x = y$, this would yield a loop. If $h_i \neq h_j$, then getting x, y that satisfies this constraint is infeasible as this requires $\sqrt{2n}$ operations.

Case (ii) All vertices of G_d are of in-degree at most one. We know that underlying graph of G_d has a cycle, say C. Consider the sub-graph of G_d induced on the vertices of C, call it C'. We know that $\forall v \in C', in\_degree(v) \leq 1$. But C' is a cycle. Hence $\forall v \in C', in\_degree(v) = 1$. Thus, we have $x = \{h_l, h_r\}^*(x)$. This requires $O(\sqrt{\pi n/2})$ operations [14] and is hence infeasible.

□

5.3.5 Technique 2: Pushback

To remove *dead-ends* in a topology of triggers, we propose a simple pushback mechanism. When a data packet reaches a dead-end, the $i3$ node will not be able to find a trigger corresponding to the ID in the packet. In such a scenario, this $i3$ node would send a message to the $i3$ node where the packet was last matched asking it to remove the previous trigger in the chain. This would result in a cascading removal of triggers until all the "useless" triggers are removed. We discuss how we deal with pushbacks initiated by malicious end-hosts and with the case of invalid IP addresses as leaves in a topology of triggers in the next section.

Consider a dead-end trigger (x, y), i.e., y is not an end-host address and there is no other trigger with an ID that matches y. Let A and B denote the $i3$ nodes responsible for x and y respectively. Consider a packet $(x, data)$ that is sent into $i3$. This packet is first forwarded to node A (which stores the trigger (x, y)), which then forwards this packet as $(y, data)$ to node B. Upon receipt of packet $(y, data)$, node B detects that there is no matching trigger for y, and hence sends a pushback message for ID y back to node A. Here, we take advantage of the fact that an $i3$ packet carries the address of the node were the packet was last matched, which is node A in this case (used for caching in $i3$ [31]). Upon receiving the pushback message, node A needs to remove all triggers that point to y. One way to implement this operation is to maintain an inverted table for all triggers stored at an $i3$ node. For each trigger $t = (x, y)$, we maintain an entry $(y, ptr(t))$, where $ptr(t)$ represents the pointer to trigger t.

5.3.6 Technique 3: Trigger Challenges

To solve the problem of *reflection* attacks, that of *dead-ends* to non-existent hosts, and that of malicious trigger removal, we extend the solution already proposed in $i3$ [31]. *Secure-i3* nodes challenge the insertion of every trigger that points to an end-host. Upon receiving a trigger insertion message, the $i3$ node computes a challenge and sends it back to R. In turn, R re-sends the trigger insertion message together with the challenge received from $i3$. Finally, upon receiving this message (with the correct challenge), the $i3$ node inserts trigger (x, R). In order to avoid maintaining state in $i3$ nodes for every attempted trigger insertion, the challenge is computed as a secret one-way hash function on the values x and R. Since every attempted insertion of a trigger (x, R) yields the same hash result, $i3$ nodes do not need to maintain any state about the challenge.

To increase the robustness of this mechanism, an $i3$ node can periodically change the challenge of a trigger (by changing the hash function). Since every trigger refresh message is acknowledged in $i3$, the new challenge can be piggybacked with the acknowledgment. Note that this simple scheme not only ensures that the trigger points to an existing end-host but also ensures that the end-host is the one (i) which in-

serted and (ii) which is maintaining the trigger. We note that, an end-host can still remove triggers when under attack since (i) out-going traffic from the end-hosts is under its control and (ii) the end-host already is aware of the challenge. In fact, in the worst case, the trigger would be removed after a timeout period (of 30 seconds [31]) as the client will not refresh it.

A similar spoofing attack can arise when an attacker injects malicious pushbacks, thereby removing valid triggers from $i3$. We adopt a similar technique in this case: upon receiving a pushback message, an $i3$ node sends a challenge to the sender of the pushback. Only when the challenge is acknowledged is the trigger removed. Furthermore, since we assume that attacks cannot snoop on communication between $i3$ nodes (Refer Section 4), the attacker would not be able to respond to the challenge.

5.3.7 Defense against Multicast-based attacks

We first explain why *node confluence* is not as serious as it might appear. First, the fact that an $i3$ public node does not store r-constrained triggers makes it impossible for an attacker to use this attack on public $i3$ nodes (which store public triggers). Second, if the attacker attacks an $i3$ node storing private triggers, the worst that can happen is that the $i3$ node will become overloaded. Arguably, the simplest way for an overloaded $i3$ node to shed load is to drop some of its triggers (maybe starting with the triggers that carry the most traffic). Of course, this would cause legitimate clients to lose triggers as well, but they can easily recover by changing their private triggers and storing them at other nodes. If the attacker also decides to move its triggers, this case would degenerate to building a random multicast tree. On the other hand, if the attacker insists on attacking the same $i3$ node, this will gradually cause all legitimate clients to move their private triggers to other *Secure-*$i3$ nodes, which would again render the attack futile.

An attacker can attack the infrastructure by building a larger random multicast tree and blasting packets into the infrastructure. However, the extent of damage that the attacker can cause on the infrastructure would be limited by the resources at his disposal, which is a fundamental limitation for any communication infrastructure. To see why, consider the following three constraints. First, we have shown that all leaf triggers of the multicast tree have to point to valid receivers. Otherwise, the pushback mechanism will rapidly prune the dead-end branches of the tree thus stopping the attack. Second, a leaf trigger that points to an end-host R can be inserted and maintained only by R itself. Third, R's ingress link cannot be overloaded. Otherwise, the challenge updates[8] sent by $i3$ for R's trigger might be be lost, which will lead to R's trigger being removed from $i3$. As a result, an attacker can build a multicast tree using only end-hosts it controls, and it cannot send more traffic than its receivers can handle.

[8]Here we assume that challenges are changed with a high frequency.

5.4 Summary

We summarize the advantages of *Secure-*$i3$ over the Internet. First, *Secure-*$i3$ provides more functionality than the Internet by means of a flexible indirection primitive. Second, the flexibility of *Secure-*$i3$ plays a major role in enhancing the end-hosts' ability to defend against attacks. Finally, using simple techniques, we ensure that we do not create any more security vulnerabilities.

5.4.1 End-host Protection Against DoS Attacks

We summarize our arguments of why *Secure-*$i3$ provides better protection to end-hosts against DoS attacks.

1. *Prevent IP level flooding attacks on end-hosts.* By hiding the IP address of the end-hosts, we prevent packet floods at the IP level.
2. *Inability to attack private communication.* End-hosts communicate by means of private IDs after establishing a connection which the attackers have no knowledge of. Hence, they cannot identify the $i3$ nodes which store the private triggers of the end-hosts.
3. *Alleviate flooding at the* $i3$ *level.* By techniques such as trigger dropping, and slowing down the attacker using crypto puzzles, end-hosts (servers) can provide a degraded service depending on the intensity of the attack.
4. *Minimize collateral damage.* Different services co-located behind a common access link can isolate the attacks on one another by removing the triggers corresponding to the service under attack.
5. *Access control for multicast channels.* Unlike IP multicast, providing access control in $i3$ is straightforward, and this would prevent flooding attacks directed towards multicast addresses.

5.4.2 End-host Response to Attacks on Infrastructure

IP level attack. By not revealing the IP address of the public *Secure-*$i3$ nodes that store public triggers, we prevent IP level attacks on public *Secure-*$i3$ nodes. Such attacks are possible on private *Secure-*$i3$ nodes, but we do not foresee this to be a problem as private triggers are supposed to be kept secret. Thus, as far as attacking private *Secure-*$i3$ nodes is concerned, the best an attacker can do is to attack a random private *Secure-*$i3$ node.

Secure-$i3$ *level attack. Secure-*$i3$ allows end-hosts to build only tree communication topologies which carry packets to valid destinations: constrained triggers ensure that end-hosts can build only tree topologies, pushback ensures that the packets are ultimately delivered to an IP address, and trigger challenges ensure that each IP address belongs to a valid end-host. Thus, attackers can mount an attack only to the extent of resources they have. Based on the study in [24], we

Attacks \ Defenses	Trigger Constraints	Public *i3*Node Constraint	Pushback	Challenges
Eavesdropping Impersonation	✓	✓		
Loops Confluences	✓			
Dead−ends			✓	
Public *i3*Node Confluence		✓		
Reflection				✓
Malicious Trigger−Remove				✓

Figure 8: Attacks and Defenses

expect *Secure-*$i3$ nodes to have more bandwidth than most of the attackers.

The ability of end-hosts to re-locate their triggers is also crucial to evading attacks on infrastructure nodes. In contrast, in the Internet, an end-hosts can do little if a router that carries its traffic is under attack.

By analyzing various attack possibilities methodically, we identified the possible new attacks on $i3$. Figure 8 summarizes the various possible attacks and the techniques that prevent these attacks in *Secure-*$i3$.

6 Discussion: *Secure-*$i3$ vs. $i3$

In this section, we discuss how *Secure-*$i3$ differs from $i3$, and the impact of *Secure-*$i3$'s extensions on the performance and functionality of the original $i3$.

6.1 Proposed Changes

We shall now summarize the main aspects of our re-design which makes *Secure-*$i3$ more secure.

ID structure. The 256-bit identifier in *Secure-*$i3$ is divided into three fields: a 64-bit prefix, a 128-bit key, and a 64-bit suffix. In contrast, identifiers in $i3$ are divided into two fields, a 128-bit prefix and a 128-bit suffix. In both *Secure-*$i3$ and $i3$, end-hosts have full control on choosing the prefix and the suffix of an ID: the prefix determines the location of a trigger, while the suffix is used to implement anycast. IDs x and y are matched based on the longest prefix matching rule, given the constraint that both their keys and prefixes match. Finally, unlike $i3$, *Secure-*$i3$ defines separate ID spaces for public and private IDs.

Constraints on triggers. Secure-$i3$ only allows triggers of the form (x, y) where $x.key = h_l(y)$ or $y.key = h_r(x)$.

Constraints at public Secure-$i3$ *nodes.* Public *Secure-*$i3$ nodes (i) store only l-constrained triggers with public IDs, and (ii) maintain no triggers pointing to end-hosts.

Pushback. If a "dead-end" is detected, trigger removals are cascaded up the chain of triggers.

Challenges. To prevent malicious insertion of triggers on behalf of a victim, triggers pointing to end-hosts are challenged. Pushback messages are also challenged.

Stack of identifiers. A trigger with a stack of identifiers is defined to satisfy the trigger constraints iff the *top* of the stack satisfies the constraints. Intuitively, this definition follows from the fact that only the top of the stack is involved in routing the packet inside the $i3$ infrastructure. The rest of the stack is used by the end-hosts after the packet leaves the $i3$ infrastructure, and hence is not involved in the constraints. Hence, all discussions involving a trigger with one identifier would also apply to a stack of identifiers.

6.2 Impact on Performance

In this section, we discuss the impact of the changes made on both data and control paths.

Data path. All the components of the re-design of $i3$ (trigger constraints, pushback and challenges) to make it secure are restricted to the control path. Since *Secure-*$i3$ does not perform any additional operations on the data path, packet forwarding should be as efficient as in the original $i3$.

Trigger Challenges. Public triggers and private triggers that do not point to end-hosts are not challenged and hence do not have this overhead. The insertion of a trigger that points to end-hosts requires an additional RTT. However, end-hosts that want to eliminate this extra RTT can maintain a private trigger in $i3$ even when it is not using the trigger. This would eliminate the trigger challenge step during connection setup.

Checking Constraints. Upon receiving a request for trigger insertion, a *Secure-*$i3$ node needs to check whether the trigger is pointing to an end-point, and whether it is r or l-constrained. Also, when an *Secure-*$i3$ node actually inserts the trigger, it needs to insert an entry in an inverted table, which is required to implement the pushback operation. One important observation is that all these operations are performed only at trigger insertion. Upon trigger refreshing, *Secure-*$i3$ does not need to do any more operations than $i3$. We evaluate the overhead of these operations in Section 7.

Overhead with Public Triggers. Since public triggers cannot point to end-hosts (servers) in *Secure-*$i3$, end-hosts need to add another level of indirection (see Section 5.1). While this can increase the delay in contacting an end-host through its public trigger, this penalty is incurred only by the first packet. This is because, when the client contacts the server, they exchange a set of private triggers.

Control on Trigger Placement. End-hosts still have the freedom of picking the prefix and suffix of a trigger ID. This means that an end-host can still control the location of a trigger by choosing an appropriate prefix, the only difference being the size of the prefix and suffix (64-bits as opposed to 128-bits).

6.3 Impact on Functionality

While constrained triggers help in eliminating undesirable topologies, whether this would limit the functionality and flexibility of $i3$ remains to be seen. We now answer this by showing that constrained triggers have only a limited impact on *Secure-*$i3$'s ability to support the basic communication primitives: anycast, multicast, and service composition.

Mobility. Constrained triggers do not have any impact on mobility, as the constraints are not computed over the IP addresses of the end-hosts.

Multicast. Applications can still build legitimate multicast trees as in $i3$ by using r-constrained triggers. The triggers that are used to build multicast trees are private triggers and hence having r-constrained triggers would not expose the multicast group to eavesdropping attacks.

Anycast. Anycast functionality is not affected by trigger constraints. Anycast requires insertion of triggers (i) which point to different end-hosts, and (ii) the IDs of which share the same *prefix* and *key*. In addition, the *suffix* of each trigger ID encodes application semantics such as end-host location. Let (x, y) be an anycast trigger inserted by host R, where R's address is encoded in the fields $y.prefix$ and $y.suffix$. If the trigger is r-constrained then $y.key = h_r(x)$, and the end-host has the entire freedom in choosing trigger's ID x. If the trigger is l-constrained, we have $x.key = h_l(y.key)$ (see Figure 7(d)). Since $x.key$ should be the same for every anycast trigger, all triggers need to share $y.key$. Thus, in an anycast group with l-constrained triggers, end-hosts need to exchange the secret key $y.key$ out-of-band.

Service composition. Disallowing insertion of arbitrary triggers still allows sender-driven service composition, but weakens the flexibility of receiver-driven service composition. In particular, it will not be possible for a receiver to redirect packets with a *given* ID x to an intermediate node with a *given* ID y, since this would require the receiver to insert a trigger of the form (x, y), where x and y are fixed in advance. However, this situation can be dealt with at the application level by negotiating a private trigger that will satisfy either an r or an l-constraint with the sender or the intermediate node. In particular, if y is given, the receiver can ask the sender to send packets with ID $h_l(y)$ instead of x. We expect this constraint to be acceptable for the vast majority of applications.

7 Evaluation

We implemented *Secure-*$i3$ and deployed it on a cluster of Linux machines. For efficiency, our one-way hash function is based on the advanced encryption standard (AES) (aka. Rijndael block cipher) [10] in the Matyas, Meyer, and Oseas construction [22]. Note that any other keyed one-way hash function (i.e., message authentication code) would also suffice, for example HMAC-MD5 [3]. The 128-bit key is encrypted by the AES cipher and then the output is XORed with the input. The second step is necessary to make the function irreversible. We get two different one-way functions, h_l and h_r, by keying the cipher with two different publicly known keys (different from the keys we hash).

Scenarios in Trigger Insertion Operation	Overhead (in μs)
No constraints checking ($i3$)	16.9
Challenge fails (*Secure-*$i3$)	5.3
Challenge matches, constraints fail	7.9
Challenge, constraints match	21.7
Challenge, constraints match (inverted table)	39.5

Figure 9: Overhead for processing trigger insertion requests

7.1 Computational Overhead

As we mentioned in Section 6.2, checking trigger constraints and challenges involves computation of a one-way hash function. To measure the additional overhead, we ran tests on a *Secure-*$i3$ node running on a 866 MHz Pentium III machine running Linux 2.4.8. The results are averaged over half a million trigger insertions. The trigger insertion time is computed as the time to add the entry to the trigger list from the time the packet was completely received. Clearly, if the challenge checking or trigger constraint checking fails, the trigger is not inserted.

We compare the time it takes to process the trigger insertion in all the possible cases, i.e.: (i) without any checking (i.e., as in $i3$), (ii) challenge fails (in *Secure-*$i3$), (iii) challenge succeeds but the constraints do not match (in *Secure-*$i3$), and (iv) trigger is successfully inserted (in *Secure-*$i3$).

Table 9 indicates that checking constraints at the $i3$ nodes increases the overhead of trigger insertion only by about 28%. Cases (ii) and (iii) take lower time than case (i) as the trigger is not inserted in these cases. We also note that the running time for a hash-computation is less than $3\mu s$ (not shown in the table). We finally note that the cost of a trigger insertion including the maintenance of an inverted trigger table (for pushback) takes $39.5\mu s$. Though this is about 2.3 times the cost of insertion in original $i3$, this is incurred only upon a successful trigger insertion — trigger refreshes do not involve this extra cost.

7.2 Reaction to DoS Attack

To illustrate how *Secure-*$i3$'s flexibility can help end-hosts to defend against DoS attacks, we illustrate the technique described in Section 5.2.2.

We use a server S running behind a cable modem with the total inbound bandwidth of approximately 1.7 Mbps (measured at the time of our experiments). S maintains 10 public triggers, and one private trigger, through which it receives a traffic of about 400kbps. We emulate an attacker by blasting 4 Mbps traffic uniformly distributed among S's public

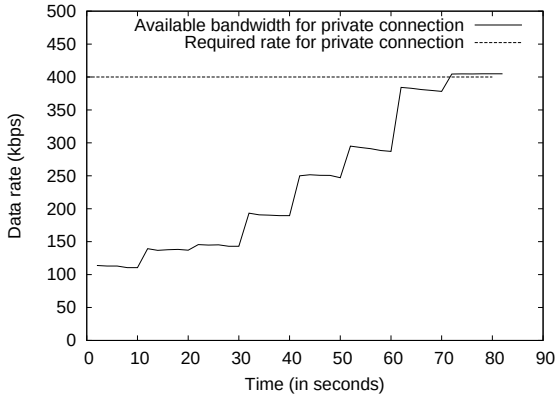


Figure 10: Reaction to DoS attack

triggers. In response, S starts dropping public triggers every $10\ s$ until the traffic on the private connection suffers no more losses. Figure 7.2 shows how much bandwidth is available to the private connection as a function of time. As expected, when seven triggers are dropped, the attack traffic rate drops to around $1.2\text{Mbps} = (4\text{Mbps} \times (1 - 7/10))$. Therefore, S will receive the entire traffic through the private trigger, since the private trigger is unaffected. The contention based nature of cable connection, along with the packet-shaper implemented at the cable modem are possible reasons why this is not a perfect step-function.

8 Practical Considerations

Deployment. In this paper, we assume that *Secure-*$i3$ is deployed by a single provider such as Akamai. This allows us to ignore trust and security issues between *Secure-*$i3$ nodes. However, a possible scenario is the presence of multiple providers much like in the Internet, and this raises questions about trust, security and policies between providers. Answering these questions is a topic of future work.

Public ID resolution. To connect to a server, a client needs to find a public ID of that server. One possibility would be to use DNS in order to obtain public IDs. Another possibility would be to use a lookup protocol such as Chord to implement name resolution [8]. Evaluating these alternatives from the viewpoint of security and availability, and developing new solutions are important aspects of our future research.

Secure-$i3$ **state.** *Secure-*$i3$ nodes need to maintain state in the form of public and private triggers. An attacker can then mount a DoS attack by compromising a very large number of end-hosts, and making each of them insert a large number of triggers. This may overload some *Secure-*$i3$ servers which will prevent legitimate triggers from being inserted. While this is a topic of future research, we make two observations. First, if the attacker does not attack the entire infrastructure, clients might be able to route around the attack by re-locating their triggers to other *Secure-*$i3$ nodes. Second, under heavy load, *Secure-*$i3$ nodes can challenge every trigger insertion by sending a puzzle, thus slowing down the attack.

9 Related Work

Due to the potentially adverse economic impact of DoS attacks and their prevalence in today's Internet [24], providing protection against these attacks has received considerable attention recently. The proposed solutions can be roughly divided into two classes: IP-level and overlay-based solutions.

IP level solutions are based on two key techniques: IP traceback and pushback. IP techniques aim to detect the sources of DoS attacks even when the attacker spoofs the source addresses. With *pushback* technique [21], routers try to identify the offending traffic and then push filters to the upstream routers to stop the attack as close to the originators as possible. Both of these techniques require router support: IP traceback requires routers to mark packets [28, 11] or send special ICMP messages [4], while pushback requires routers to detect the offending traffic and perform packet classification. In contrast, our solution requires no router support. Furthermore, the detection of the attack is left to the end-hosts which in general have more information than the network about the traffic they receive. This makes our techniques more fine-grained and flexible than the aforementioned ones.

Secure Overlay Services [20] was one of the first solutions to explore the idea of using overlay networks for pro-actively defending against DoS attacks. SOS protects end-hosts from flooding attacks by (i) installing filters at the ISP providing connectivity to the end-host and (ii) using an overlay network to authenticate the users. Mayday [1] generalizes this architecture and analyzes the implications of choosing different filtering techniques and overlay routing mechanisms.

However, these solutions assume that the set of authorized users is known in advance, and that the set changes infrequently so that updating the authentication rules in the overlay nodes occurs rarely. In addition, these proposals assume support from the IP infrastructure for providing basic filtering near the target nodes, which might not be possible in the general case. In contrast, our solution provides protection to all end-hosts without imposing any restriction on the set of senders, and doesn't require support from the IP infrastructure. Another point worth noting is that an infrastructure such as $i3$ can easily support the authentication functionality by composing a service which provides the authentication.

Our principle of hiding the IP address is similar in spirit to Crowds [26], Onion-routing [25], Chaum-Mixes [7] and Web-Mix [5]). However, the goal of these systems is to provide anonymity rather than to enhance security.

The first use of cryptographic puzzles is due to Merkle [23], who used puzzles for the first instantiation of a public-key protocol. Dwork *et al.* propose puzzles to discourage spam-

mers from sending junk email [13]. Juels *et al.* used puzzles to prevent TCP SYN flooding [17]. Aura *et al.* [2] and Dean *et al.* [12] propose puzzles to defend against DoS on the initial authentication.

Finally, we note that while in this paper we assume a secure overlay routing infrastructure, related pieces of work in this area ([6], [30]) have studied various attacks in which the infrastructure's nodes are compromised.

10 Conclusions

In this paper, we present an overlay communication infrastructure that provides more functionality and at the same time better protection against DoS attacks than the Internet. Our solution, *Secure-*$i3$ (a re-design of $i3$), is built around three design principles: enabling end-hosts to communicate without revealing their IP addresses; empowering end-hosts to stop or slow-down a DoS attack directed to them, and to route around DoS attacks directed at the infrastructure; and making sure that we do not introduce new vulnerabilities.

Secure-$i3$ uses three techniques to eliminate the vulnerabilities of $i3$. (i) trigger constraints to restrict the communication topology of an application to a tree, (ii) pushback to eliminate chains of triggers pointing to dead-ends, and (iii) trigger challenges to avoid reflection. We argue that these changes have virtually no impact on $i3$'s functionality. In addition, using experimental results we argue that the overhead introduced by *Secure-*$i3$ on the control path is acceptable.

While in this paper, we have tried to argue that our system does not introduce new vulnerabilities, more remains to be done. Section 8 mentions several open questions. In addition, we plan to deploy *Secure-*$i3$ in a wide-area testbed (e.g. on PlanetLab). Ultimately, only real applications and users would help to answer these questions.

References

[1] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *USITS*, Seattle, WA, 2003.

[2] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant Authentication with Client Puzzles. In *Security Protocols—8th International Workshop*, Lecture Notes in Computer Science, Cambridge, United Kingdom, Apr. 2000. Springer-Verlag.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In N. Koblitz, editor, *Advances in Cryptology - Crypto '96*, pages 1–15. Springer-Verlag, 1996. Lecture Notes in Computer Science Vol. 1109.

[4] S. Bellovin. ICMP Traceback Messages, Internet draft, draft-bellovin-itrace-00.txt, work in progress, march 2000, 2000.

[5] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In H. Federrath, editor, *International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 115–129, Berkeley, CA, USA, July 2000. Springer-Verlag, Berlin Germany.

[6] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-peer Overlay Networks. In *OSDI*, Boston, MA, Dec. 2002.

[7] D. L. Chaum. Untraceable Electronic mail, Return addresses, and Digital pseudonyms. *Communications of the ACM*, 24(2):84–88, Feb. 1981.

[8] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using Chord. In *IPTPS 2002*, Cambridge, MA, March 2002.

[9] Security of e-commerce Threatened by 512-bit number factorization. `http://www.cwi.nl/~kik/persb-UK.html`, Aug. 1999. CWI press release.

[10] J. Daemen and V. Rijmen. AES proposal: Rijndael, Mar. 1999.

[11] D. Dean, M. Franklin, and A. Stubblefield. An Algebraic Approach to IP Traceback. *Information and System Security*, 5(2):119–137, 2002.

[12] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proc. of the 10th USENIX Security Symposium*, Washington, D.C., Aug. 2001. USENIX.

[13] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In E. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. International Association for Cryptologic Research, Springer-Verlag, 1993.

[14] P. Flajolet and A. Odlyzko. Random Mapping Statistics. In *Advances in Cryptology — EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*. International Association for Cryptologic Research, Springer-Verlag, 1990.

[15] S. Gibson. The strange tale of the denial of service attacks. `http://grc.com/dos/grcdos.htm`, Mar. 2002.

[16] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *NDSS 2002*, San Diego, CA, Feb. 2002. Internet Society.

[17] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proc. of the Symposium on NDSS*, pages 151–165, 1999.

[18] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). Internet RFC 2406, IETF, Nov. 1998.

[19] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Internet RFC 2401, IETF, Nov. 1998.

[20] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proc. of ACM SIGCOMM 2002*, pages 20–30, Pittsburgh, PA, Aug. 2002.

[21] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *CCR*, 32(3):62–73, July 2002.

[22] S. Matyas, C. Meyer, and J. Oseas. Generating Strong One-way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[23] R. Merkle. Secure Communication Over Insecure Channels. *Commun. ACM*, 21(4):294–299, Apr. 1978.

[24] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. In *Proc. of USENIX Security*, pages 9–22, 2001.

[25] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE JSAC*, 16(4), May 1998. Spl. Issue on Copyright and Privacy Protection.

[26] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[27] R. L. Rivest. The MD5 message-digest algorithm. RFC 1321, Apr. 1992.

[28] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. of ACM SIGCOMM 2000*, pages 295–306, Stockholm, Sept. 2000.

[29] B. Schneier. Why cryptography is harder than it looks. `http://www.counterpane.com/whycrypto.html`, 1997.

[30] E. Sit and R. Morris. Security Considerations for Peer-to-peer Distributed Hash Tables. In *Proc. of IPTPS, 2002*, Cambridge, MA, Mar. 2002.

[31] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proc. of ACM SIGCOMM 2002*, pages 10–20, Pittsburgh, PA, Aug. 2002.

[32] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. In *Proc. of ACM SIGCOMM 2001*, pages 149–160, San Diego, CA, Aug. 2001.

[33] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures. Internet RFC 2627, IETF, June 1999.

[34] C. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs. In *Proc. of the ACM SIGCOMM '98*, pages 68–79, 1998.