

MULTIPLE IDENTITY ATTACKS ON DISTRIBUTED
SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF THE HEBREW UNIVERSITY OF JERUSALEM
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

Elliot Jaffe
February 2005

Preface

This thesis explores a form of attack on a distributed system in which one or more nodes in the system attempts to maintain multiple identities through self-replication. Requirements for defenses of this attack are explored. A review of the state of the art is provided. A new defense is proposed and explored in both analysis and simulation.

Acknowledgements

I would like to thank my family for their support during the past two years. I have thoroughly enjoyed learning, teaching and researching through the course of my masters degree.

Thank you to Professor Dahlia Malkhi, Professor Danny Dolev, and Professor Scott KirkPatrick for their support. Danny Dolev was particularly supportive of my intention to return to school after a fifteen year hiatus and provided much needed confidence when I faltered.

I have enjoyed working with Dahlia Malkhi on this thesis as well as on teaching duties during the past year. It is always a pleasure to work with a well organized professional.

Special thanks to Danny Bickson for his support and effort and coffee. Despite being significantly older than many of the masters and doctorate candidates, I felt welcomed and included as part of the Danss group.

Contents

Preface	ii
Acknowledgements	iii
1 Introduction	1
2 Vulnerable Systems	4
2.1 Distributed Storage Systems	4
2.2 Overlay Networks	5
3 System Model	7
3.1 Types of Nodes	8
4 Provably Indefensible Systems	10
4.1 Unbounded Identities by Accretion	10
4.2 Constant Identities with Constrained Resources	11
4.3 Unbounded Identities by Collusion	12
5 Defensible Systems	15
5.1 Cost of Entry	16
5.2 Cost of Usage	17
5.3 Cost of Recognized Abuse	17
5.4 Revocation	19
6 Defenses	21
6.1 Symmetric Defense	22

6.2	Probabilistic Defense	25
6.2.1	Analysis	25
7	Sisyphus Defense	27
7.1	The Sisyphus Protocol	27
7.2	Discussion	29
7.2.1	Assumptions	29
7.2.2	Analysis	30
7.2.3	Routing	34
7.2.4	Scalability	37
7.2.5	Denial of Service attacks	37
8	Practical Results	39
8.1	Simulation	39
8.1.1	PhysicalNode	40
8.1.2	VirtualNode	41
8.1.3	SisyphusNode	42
8.1.4	Voucher	42
8.1.5	RemoteNode	42
8.2	Events	43
8.2.1	clockTickEvent	43
8.2.2	connectionEvent	43
8.2.3	reConnectionEvent	43
8.3	Operation	44
8.4	Attacks	44
8.4.1	SimpleAttack	44
8.4.2	CollectAttack	44
8.5	Churn	45
9	Related Work	49
9.1	General System Models	49
9.2	Attacks on distributed systems	50
9.3	Centralized Solutions	52

9.4	Trust Based Systems	52
9.5	Self-replication defenses	55
10	Conclusions	57
	Bibliography	58

List of Figures

3.1	Network Model	8
5.1	Verification vs. Failure Costs	18
6.1	Symmetric Defence	22
6.2	Symmetric Bounds	23
8.1	Time Units	39
8.2	Simulation Classes	40
8.3	Average false negatives with 1000 nodes and 1000 connections per time unit and 1%, 2%, 5% and 10% churn	45
8.4	Average Voucher Set size with 1000 nodes and 1000 connections per time unit and 1%, 2%, 5% and 10% churn	46

Chapter 1

Introduction

One way to view a Distributed System is as a cooperative collection of software programs or nodes communicating over some type of network. In addition to the normal challenge of building robust functional software, Distributed Systems introduce issues of reliability, conformance and trust. Reliability may be expressed as dealing with failures; programs or nodes that stop running or networks that fail to deliver messages. Conformance is related to the correctness of each program. Software that occasionally sends incorrect messages can wreak havoc by introducing confusion and uncertainty into the system.

Designers must decide how much they trust the individual components of the system. It may be possible for an external agent to introduce nodes that behave differently from the production software. This type of behavior ranges from software that attempts to collect or discover private information to malicious software that attempts to destroy or incapacitate the system itself.

In the past, most Distributed Systems research focused on passive approaches to these issues. In these systems, one argued that even if some percentage of the nodes behaved incorrectly, the rest of the system would still reach the desired result or provide the desired service. Examples of these approaches are Byzantine agreement algorithms [16] or transaction protocols [29]. Recently, there has been interest in proactive approaches to distributed system problems, particularly in the area of trust or security.

This thesis focuses on one particular type of distributed systems attack that seems

particularly hard to handle using passive defenses. The attack is described in general terms and it is shown that certain system models are indefensible. In the chapter on vulnerable systems, a number of distributed system designs are shown to be particularly susceptible to this type of attack. This is followed by an exploration of a number of possible solutions, describing their correctness and cost to the overall system. The related works chapter presents a survey of the state of the art in distributed system attacks and defenses.

Throughout this thesis, an entity is said to maintain two simultaneous identities if no communicating node can identify these two identities as belonging to the same entity. In this case that the entity is said to *present* two identities to the system.

An underlying requirement in many survivable systems is an upper bound on the fraction of corrupted participants. For example, in many distributed algorithms, a threshold of a third is a known resilience upper bound [9]. In other settings, such as a routing mesh like a hypercube, if as few as a tenth of the nodes in a sufficiently large system are corrupted, then those nodes control most of the routing paths (whose length is logarithmic). The systems resilience threshold to routing attacks is therefore significantly lowered.

If an attacker is able to physically infiltrate a distributed system with many corrupt participants, more than the resilience threshold with which the system was designed, it can compromise any protection built in the system. It is therefore assumed (also in this work) that this is not the case, i.e., that the fraction of faulty participants in the systems conforms with some assumed upper bound.

Assuming that no significant fraction of existing nodes are corrupted, a distributed system may still be vulnerable to other specific attacks. In an attack recently named as a Sybil [10] or Self-Replication attack, an attacker assumes multiple identities and although physically it does not pass the assumed resilience bounds, in practice it plays the roles of more participants than the assumed threshold. This may again lead to a complete compromise of system safety.

In this work, we focus solely on the problem of enforcing a bound on the number of identities assumed by an attacker in a distributed system, under the assumption that its physical strength is appropriately limited.

It has often been suggested to use computational power in order to limit masquerading over the Internet. For example, Microsoft’s anti-spam project [1] revolves around an old idea by Dwork and Naor [11], offering to use computation as ‘stamps’ for Email. Computational challenge was suggested by Franklin and Malkhi in [12] to perform web page usage metering. This seems like a good idea, but in a peer environment with no centralized control, it is not obvious how to use it. Some of the issues that need to be addressed are:

- Who would present and verify the computational challenges to participants? We should be careful not to let the attacker “vouch for itself”, in a similar way that certain web pages point to each other in order to boost their “Google ranking”.
- How will the verifier certify valid participants? In a peer network, it is unreasonable to assume a central certification authority. This seems to completely rule out the possibility that a peer will possess a *certificate* of honesty that it can directly present to other participants.
- How to prevent a slow infiltration of an attacker that gradually assumes more and more identities? There needs to be a fine tuning that allows legitimate users to do useful work, while preventing the attacker from spending all of its time just assuming identities.

As will be shown later, Computational Challenges alone do not provide a workable solution. This work extends these concept of Computational Challenges with a witness-set mechanism similar to that presented by Stuart Haber in his work on Digital Timestamps [14] and by Dahlia Malkhi, et.al. in their work on Reliable Multicast protocols [17]. This work extends the witness-set model with a practical method for selecting the witness sets and handling the dynamics of node entry and exit during the system lifetime.

This work presents a complete peer-based architecture for controlling the number of false identities assumed by its participants, using reasonable assumptions about an attacker’s power. The resulting architecture is presented analytically and through a simulation developed for this purpose.

Chapter 2

Vulnerable Systems

In this chapter, we explore a number of system designs which are susceptible to self-replication attacks. We describe the overall design and point out where and how the system is vulnerable.

2.1 Distributed Storage Systems

A number of researchers [3],[24] have proposed storage systems where each file is distributed across a large number of nodes. Typically, each file is broken into small segments. These segments are then distributed and replicated across a large number of nodes. Replication protects the system against the failure of any single node. By selecting the number of replicas for each piece of data, the system can be shown to have an arbitrarily small chance of actually losing data.

One measure of confidentiality in these systems is the privacy of each data file. The intent is that no unauthorized node can gather enough information to re-create the original file. If file segments are distributed across multiple machines then in order for a file to be reconstructed, the challenger must collect at least one copy of each segment of the file.

In a self-replication attack on a distributed storage system, a single node can create and maintain sufficient identities that the file system may inadvertently allocate at least one copy of every segment to that single physical node, thus violating the system's confidentiality guarantees.

When discussing Distributed Storage Systems, it was assumed that data was generated by a single node and then distributed to other nodes for storage. The attacker attempted to have those data segments stored on a single physical self-replicated node. The following example demonstrates that this attack also applies to systems where the data itself is generated in the remote nodes.

Recently, Michael Rabin has proposed a Virtual Satellite model [23], now being implemented at Harvard, which extends the bounded storage model of Maurer [18] for unbreakable encryption [8, 22] to practical peer-to-peer systems. In this model, each peer locally generates random pages which are replaced very frequently. Two nodes use their shared key to select from which peers in the network to take pages. The selected pages are then replaced, guaranteeing that at most two nodes reviewed the page contents. An adversary cannot know which peers will be accessed, and it cannot listen to all network traffic, hence the adversary cannot access those pages and the one-time pad is unbreakable.

If an adversary can use a self-replicating attack to maintain a sufficient number of multiple identities then it is possible for that attacker to be those node(s) which provide the random pages, thus allowing the attacker access to the random data and hence significantly weakening the resulting one-time pad.

2.2 Overlay Networks

Overlay networks define an addressing and routing mechanism that is orthogonal to the underlying networking system. The benefits of such networks include the ability to route data between nodes using non-compatible networking system, content addressable networks and anonymity.

Recent works [4] [28] have explored the robustness of overlay network systems to failures of internal nodes. The problem is whether such systems can continue to route messages through alternate paths. At issue is whether an attacker can control the routing of messages through the capture of a small number of routing nodes.

Three approaches have been explored to provide guarantees of message routing in overlay networks. First, if the system can assume secure node identifiers, then routing can be easily verified. Unfortunately, there are no known non-centralized mechanisms

that guarantee secure node identifiers.

Secondly, in the Secure Routing Table management approach, additional routing tables are maintained, providing multiple routing paths through the network. An attacker must then capture nodes on each routing table in order to guarantee control over message traffic.

Finally, in the Secure Message forwarding approach probabilistic cryptographic approaches are taken to ensure that, with high probability, at least one copy of the message reaches each of the destination nodes.

An attacker can sustain a self-replication attack to masquerade as a sufficient number of routing nodes then such that all paths to the destination node pass through one or more of its false identities. The previous approaches are effective if a centralized authority exists for either node identities or PKI. In the absence of such a centralized authority, an attacker that can garner an arbitrarily large number of identities can break any of the known routing algorithms.

Chapter 3

System Model

Our first step is to clearly define the system model under consideration. This model is extremely general. We believe that successful defenses will be the result of narrowing the model in one form or another.

We will first describe the formal model, and then define a number of properties that may be observed in that model.

The system consists of as a set of physical nodes N each of which has a two types of resources; a bounded resource S and a renewable resource P . Bounded resources represent fixed resources that once consumed are held for the duration of the process. An example of a bounded resource is disk space which can be used to store private data.

Renewable resources P are bounded during a time span T . Each T time, the resource is refreshed and may once again be allocated. Typical renewable resources include processing time or network messages.

System nodes are connected through a network cloud which provides direct end-to-end connectivity between any two nodes in N . Communications between nodes is reliable, instantaneous and independent of the size of the packets. That is, all packets which are sent by the initiating node are received by target node. There is no delay in the transmission of any packet. The size of the packet has no impact on the transmission time of the packet. These are simplifying assumptions for the purpose of discussion.

Each node has its own set of capacity limited resources S and P which it may use

for any purpose. We assume that there is by design a minimum capacity requirement for any node which wishes to join the system. Nodes with less than this capacity are unable to perform the minimum amount of work necessary to participate in the system. For example, a node must have minimally sufficient capacity to perform the certification procedures for generating their own identity. Denote this minimum capacity as S_{min} and P_{min} respectively.

Physical nodes are represented in the system using abstract identities. Each physical node n may attempt to represent itself as one or more abstract identities $n_i, i \in \mathbb{N}$.

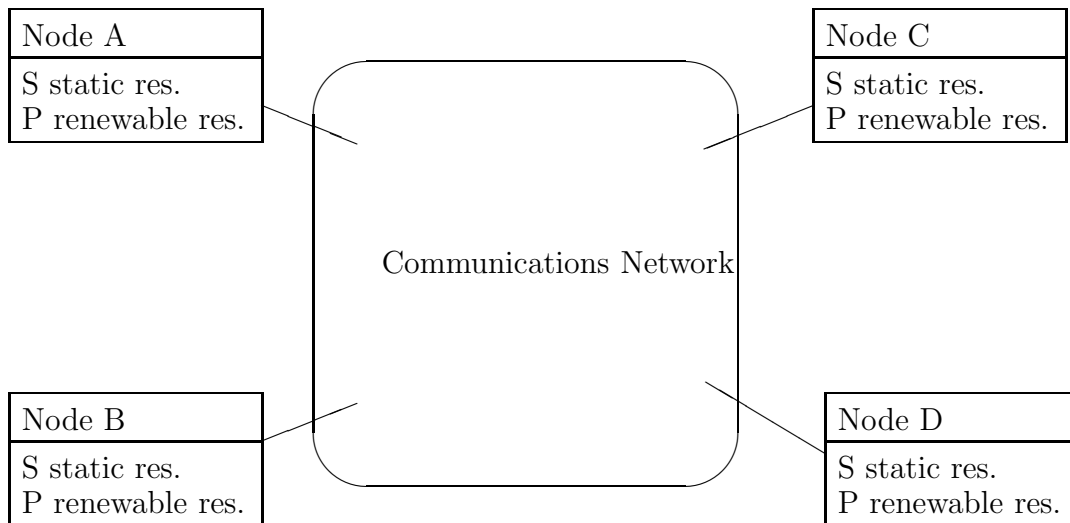


Figure 3.1: Network Model

3.1 Types of Nodes

There are many possible classifications of system nodes. In this section we define a taxonomy of nodes based on their perceived behaviors within the system.

Our broadest taxonomy divides between *legal* nodes and *malicious* nodes. A legal node faithfully performs all external and internal system protocols and procedures. It never presents false information nor does it communicate out-of-band with other system nodes.

A malicious node is the antithesis of the legal node. It is not required to obey

any of the system protocols or procedures. A malicious node may actively attempt to damage other system nodes by sending false data, incorrect messages, performing denial of service attacks or otherwise interrupting the normal system operations.

A *passive malicious* node is weaker than a full malicious node in that it is perceived as faithfully performing all system network protocols. Such a node may present false or misleading information when queried. It may also perform additional protocols and communications unspecified by the system, including communication and colluding with other nodes both inside and outside of the system.

A passive malicious node is differentiated from an active malicious node in that a passive node does not initiate communications or otherwise attempt to disrupt or damage the system.

Chapter 4

Provably Indefensible Systems

This section discusses attacks that can result in the system accepting multiple identities associated with a single physical node. It will be shown that under some circumstances, there is no upper bound to the number of identities that such a node can present.

This chapter is based on the original Sybil paper [10]. That paper presented four lemmas concerning bounds on Sybil attacks. Those lemmas are reviewed in the form of observations qualified by our system model as presented in the previous chapter.

4.1 Unbounded Identities by Accretion

A simple approach to assigning node identities is to have some form of a cost function associated with each new node. It will be observed that the number of presentable identities is bounded only by the attackers ability to generate this cost.

Observation 1 *Assuming only an initial cost of entry, a single node may present an unbounded number of identities.*

PROOF. Assume that entry to the system has a cost C . A malicious node can pay the cost and generate an initial identity. That same node can immediately begin the entry process again. This process can be repeated indefinitely, assigning to this node an unbounded number of identities. \square

Consider a system that uses a one-time cryptographic challenge as the initial cost of entry. If that challenge is purely computational then an attacker can generate over time an unbounded number of identities. The problem is that purely computational resources are time bound and do not consume non-renewable resources.

4.2 Constant Identities with Constrained Resources

The previous section assumed that resources allocated to pay the cost of entry may be re-used at a later time. This section considers systems where resources may not be re-used, or will be required to maintain the nodes identities over time.

Observation 2 *If identities are based on a bounded resources, then any node with resource S can present at least $\lfloor \frac{S}{S_{min}} \rfloor$ identities to the system.*

PROOF. The minimally capable node must have at least S_{min} resources in order to join the system. If $S < S_{min}$ then the node in question cannot by design join the system. Thus $S \geq S_{min}$ and hence there is some value g such that $g = \lfloor \frac{S}{S_{min}} \rfloor$. The node can use these resources to present and maintain at least $g \geq 1$ identities to the system. \square

An example of non-renewable resources is a system that implements a Storage Enforcing Commitment (SEC) [13]. Such systems require that a node maintain a non-trivial amount of static data. At any time, other nodes can challenge the original node to prove that it still retains the static data and thus to verify that the node does indeed represent this identity. Since a node has a bounded amount of storage, this approach limits the number of identities that the node may present if and only if checks are made on a regular basis to insure that the data is actually present.

Observation 3 *If identities are based on a renewable nodes resources, then any node with P resources can present at least $\lfloor \frac{P}{P_{min}} \rfloor$ identities to the system.*

PROOF. The minimally capable node must have at least P_{min} renewable resources per time slice in order to join the system. If $P < P_{min}$ then by the design assumption, this node cannot join the system. Thus $P \geq P_{min}$ and hence there is some value

$\lfloor \frac{P}{P_{min}} \rfloor \geq 1$. During any given time slice, a node can use these extra resources to present and maintain at least that many identities within the system. \square

An example of a system that relies on renewable resources is a system that implements a Cryptographic Challenge to prove identity. Since the resource is renewed each time slice, in order to remain close to the lower bound q , the system must challenge and utilize these resources each time slice.

Recall that the previous two observations depend on the definition of the minimally capable node. The system may be designed such that the minimally capable node must utilize all of its resources in order to maintain its identity within the system. If all nodes in the system are minimally capable then the system is uninteresting since no other work can be performed no matter how many nodes are involved.

There are two remaining design options with respect to node capacity. Either there is additional capacity built into the minimally capable node that can be used for useful work, or a large majority of nodes are more powerful than the minimally capable node. In either case, a malicious node can use this extra capacity to present additional identities instead of contributing to the system designed workload.

4.3 Unbounded Identities by Collusion

Some researchers have proposed systems where membership and identity are based on a quorum of system nodes. Such systems are sometimes called Webs of Trust because membership is based on a chain of trust relationships that form a logical web of connections.

These systems limit the number of identities that node may present by requiring that a quorum of other system nodes vouch for that identity. For example, Datta, Aberer and Hauswith [7] present a distributed peer-to-peer system that maintains identities based on quorums of system nodes. In their system, identities are based on a mapping of identifiers to dynamic IP addresses. Nodes vouch for the fact that they communicated with a node that presented that identifier and had the specified IP address.

There are two design issues that must be addressed for such systems.

1. What is the mechanism for selecting the vouching nodes.
2. What is the cost and hence quality of each vouching node's vote.

This section will show that under many circumstances such systems are vulnerable to attacks where a number of passive malicious nodes may cooperate in such a way as to increase their membership within the system.

In quorum based systems, if a node N_i accepts the identity of node N_j then “ N_i vouches for N_j ”. The intention is that vouching for an identity implies belief that the identity is valid. In the context of self-replicating attacks, a valid identity is one that is believed to be the sole identity of a given physical node.

Observation 4 *If membership requires any Q vouchers, then Q passive malicious nodes can vouch for an unlimited number of new identities.*

PROOF. Assume that the system is designed such that any quorum of Q nodes suffices to vouch for an identity. Assume that there are Q passive malicious nodes that wish to vouch for given identity regardless of its actual status. Since there is no selection criteria for vouchers, the system will accept their votes and thus accept this identity.

Once again, since there is no selection criteria for vouchers, these same nodes can vouch for any number of identities. \square

Observation 5 *If one node can present Q identities, then that node can vouch for an unlimited number of new identities.*

PROOF. As has been previously shown, a more than minimally capable node can present multiple identities. If a node can present Q identities, then by the previous observation, this single node can collude with itself to introduce an unlimited number of identities at very limited additional cost. \square

These observations assume that the cost of vouching is negligible. That is, no significant resources are required to be utilized when vouching for a node. If the vouchers needed to expend resources in order to vouch for a node, then by observation 3 and 2, the total number of vouched for nodes would also be bounded during any given time slice.

Further, these observations assume that once vouched for by a quorum, that identity is accepted for all time. That is, no further or subsequent verification of a nodes is performed. Enforcing this assumption might require for example that a voucher utilize resources in the vouching process, and that the identity be re-verified every time slice. Re-checking identities provides an additional bound on the number of false identities in the system.

Chapter 5

Defensible Systems

In order to protect a system against self-replication attacks, the system must either guarantee that each and every node cannot self-replicate or it must have a way to identify or recognize those nodes that are actively self-replicated. Systems which cannot perform either of these approaches are clearly susceptible to self-replication attacks.

Recognition of self-replicating nodes is not sufficient in and of itself. The problem with many algorithms is that they provide mechanisms for some constituent system node or small group of nodes to categorize an identity as invalid. Yet there remains the issue of how to communicate this information to the rest of the system.

This chapter explores some of the possible approaches, identifying critical issues that any Sybil defense must provide in order to properly protect the underlying system.

The root cause of any self-replicating attack is a negligible cost to the attacker for each new identity. Existing social systems have applied many different types of cost to this problem, ranging from cash payments to peer pressure to prison. In this section we will informally explore this space.

There are at least three points where costs may be applied to identity generation. A charge can be applied for: entry to the system, ongoing usage of a system, or on determination of abuse of the system.

5.1 Cost of Entry

The most obvious point to apply a cost for identities is on a node's initial entry to the system. A candidate wishing to join an exclusive club must pay an up-front initiation or membership fee for that privilege. Such fees are common when joining any group, including academic, professional or legal organizations. The intention is to set the fee such that it self-selects candidates who are capable of meeting this minimum requirement.

A related approach is to perform a credit or background check on entry to the system. It is common when signing up for a new service to be asked to provide a credit card. The service uses this credit card to check the credit rating of the candidate. A low credit rating may indicate a problem such as a history of missed payments. Alternately, it may simply indicate that the candidate has no credit history. In either case, the service or organization uses this information as an indication of the world's trust in this candidate.

The best possible real world check is to physically meet the candidate. Our senses of vision, smell and sound provide a currently undeniable verification of the candidate's identity. Genetic testing and biometrics are extensions of this approach.

In computer systems, the cost of entry for a new node is negligible. Software may be copied from machine to machine. Hardware systems are commodities. Assembly line production techniques have reduced the difference between any two devices to something approaching zero. There are effectively no identifying characteristics that can uniquely identify a node within the system.

In human systems, currency is frequently used as a surrogate identification method. Yet there is no currency for computer systems. How can one define costs for entry when each node is equivalent? As will be shown in subsequent chapters, there are resources associated with each system, and it is possible to monetize these resources.

At this point, our concern has been to limit entry into a system by applying some non-zero cost. Assuming that there is a mechanism to pay for entry, there remains the issue of verifying that a given node did at one point in time legally join the system. Purely digital objects can be trivially copied. Hence some cryptographic system such as digital certificates or private key encryption must be employed.

The original Sybil paper [10] proposed one such system where the cost of entry can be made sufficiently high and where there is a trusted reliable digital signature authority to verify that the entry cost was indeed paid.

5.2 Cost of Usage

One issue with entry costs is that there must be an additional mechanism to securely record that the fee was paid. Another approach is to use a pay-as-you-go approach. A node pays a fee directly to its partner for each and every use. This requires only that the partner be able to validate the immediate payment with no requirement for historical data.

While this approach does not require a centralized authority for payment or verification, it imposes severe limitations on the communications within the system. In order to be effective, the cost of each communication must be non-negligible. But at the same time, since each node has limited resources, its potential interactions with other nodes is therefore also limited.

5.3 Cost of Recognized Abuse

Both of the previous approaches attempted to deterministically verify the identity of each node. If successful, no interactions with invalid nodes would occur within such a system.

An alternative approach is to apply a probabilistic method in order to limit but not deny access to self-replicating nodes. Such an approach is similar to that employed by law enforcement officials or tax collectors. It is assumed that the majority of participants are obeying the standards and laws of the community. Enforcement officials do not attempt to challenge and check each participant. Instead, they rely on two factors, occasional verification of a participant's activities and the cost of failure.

The goal in such a system is to balance the cost of verification to the system with the cost of failure to the participant. Totalitarian systems apply the extremes to both factors, assuming that each and every participant is attempting to subvert the system

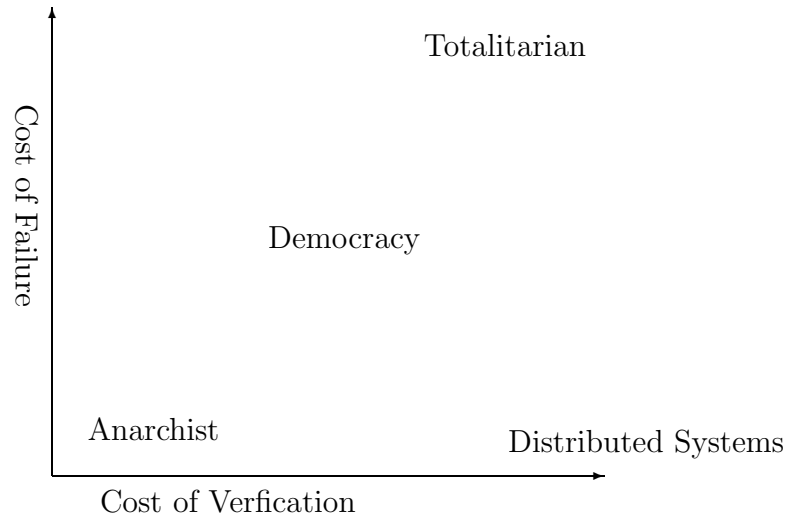


Figure 5.1: Verification vs. Failure Costs

and at the same time, imposing draconian penalties to each and every infraction. Democratic systems usually apply a balanced approach, checking participants based on profiling and imposing a range of penalties based on the frequency and severity of the abuse. Anarchic systems assume that all participants are valid, and that the cost of abuse is minimal.

Interestingly, each of these systems has different expectations as to the number and frequency of abusers. Totalitarian systems aim for zero tolerance. Democratic systems assume some amount of accepted abuse. Anarchic systems allow all participants to abuse the system at will, but hope that the participants will self-limit their activities such that the system continues to function.

As it applies to self-replication attacks, one assumes that any node can gain an identity, but that there is some mechanism whereby the duplicate identity will be recognized as such. It is possible to challenge a software based system through the utilization of a node's limited resources such as processing power, storage space or network bandwidth.

For a system susceptible to self-replication attacks, the balance between verification and cost of failure must be such that the number of duplicate identities be bounded. Thus if a node is found to have duplicate identities, the system will wish

to impose a cost on this node for its failure to observe the rules.

There are very few mechanisms for imposing post-facto costs on computer programs or systems. Those options that do exist are frequently socially unacceptable because they may punish innocent bystanders. For example, in spam attacks, it is possible to perform a distributed denial of service attack against the spammer. In addition to denying the spammer operations, a DDOS attack also impacts the communication paths upstream of the attacker, disrupting perfectly legal network communications.

Reactive attacks on system nodes perforce rely on denying network services to the target node. The system itself is unassailable. Any opportunity that would allow an external service to forcibly extract a cost could also be used by a malicious node to attack healthy legal nodes.

5.4 Revocation

Assume that there is a mechanism to identify a node identity as invalid. Must every node apply this mechanism each and every time a node is accessed, or is there a way to safely distribute the status of this identity? Ideally, one would wish for a revocation system which could be queried on demand to securely report on the status of a node identity.

A revocation system must minimize the chance of a false revocation. At some point, the revocation system must be told to revoke an entity's privileges. The process of revoking a participant's rights must be sufficiently secure that a malicious node cannot attack the system simply by revoking all legal nodes.

To be effective, the revocation mechanism must be available. It must be possible to check the status of a given identity at anytime. If it is not available then the system must either grind to a halt or allow all interactions, potentially providing an attacker unrestricted access to the system.

An ideal revocation system would be one that is decentralized, highly available and local. A decentralized system is not a single point of failure. It is less susceptible to denial of service attacks. A highly available system enables the system to maintain continuous operations. A localized revocation system is one in which an attack on a

component of the revocation system denies access to only a local bounded number of nodes.

The Sisyphus protocol provides exactly these guarantees as will be shown in chapter 7.

Chapter 6

Defenses

This chapter reviews a number of possible defenses against self-replication attacks. The defenses explore different basic approaches based on the discussion in the previous chapter.

The symmetric defense attempts to verify each interaction between participants. It imposes a high cost on interaction in return for needing no revocation mechanism. As will be shown, this defense is applicable only in restricted environments where all communication is directly from node to node with no overlay routing.

The probabilistic defense describes a simple method to probabilistically identify self-replicated identities. It has two drawbacks. The first is that there is some risk that legal nodes will be incorrectly identified as self-replicating. The second problem is that there is no known non-centralized revocation mechanism that will not allow an attacker to disrupt the system.

All of the proposed defenses rely on Computational Challenges to identify replicated identities on the attacking node. As will be shown, the simple straightforward approaches fail and cause circularity in cases where a vouching node may also be the attacker. These deficiencies are addressed in the subsequent chapter on the Sisyphus Defense.

6.1 Symmetric Defense

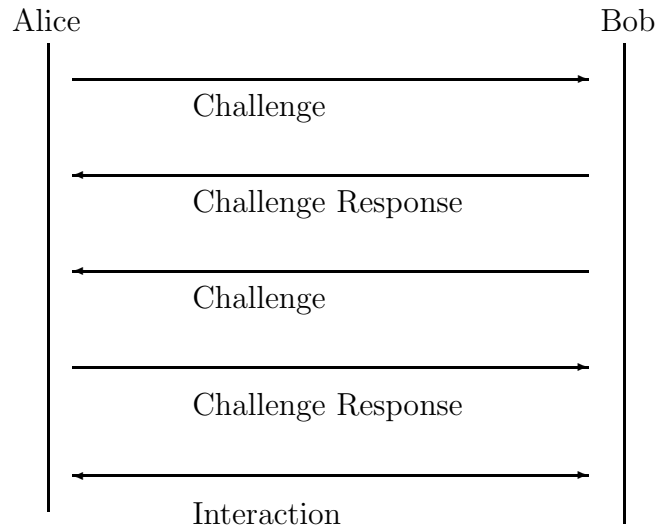


Figure 6.1: Symmetric Defence

Two nodes wish to communicate. In order to interact with each other, each node will initially challenge the other. The challenge must be renewed at a regular interval I .

The cost of generating each challenge is assumed to be negligible. But as resources are by definition limited, a node can respond to only a bounded number of challenges during each unit of time. Thus the number of identities that either node may maintain is strictly limited. Furthermore, each additional communications partner adds a fixed computational overhead. The limit is reached once the node no longer has resources with which to respond to new challenges.

A self-replicating node is therefore bounded in both the number of identities and the number of nodes with which it could interact.

The symmetric protocol would be well suited to scenarios where nodes need only communicate directly with a small number of neighboring nodes. It would be unreasonable for systems where the in- or out-degree of any given node is significant. For example, in Client/Server systems the in-degree of a server is potentially the number of clients in the network. In Distributed Storage systems, the opposite situation occurs. A client node wants to store data on as many remote nodes as possible in

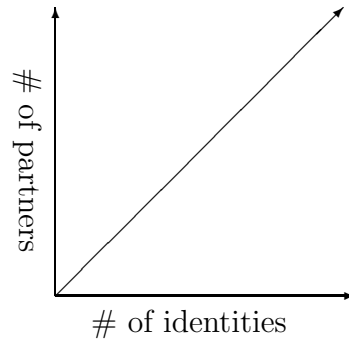


Figure 6.2: Symmetric Bounds

order to increase the chance of eventually retrieving the data. In such a case, the out-degree of the client node may be a hundred or more nodes.

Unfortunately the symmetric defense may fail even in systems that consist of a large number of nodes and where any given node has a small in/out-degree. Consider Distributed Hash tables, where there has been significant focus over the past few years on bounding the degree of the network. In such systems, each node sends direct messages only to its immediate neighbors. Those messages intended for other nodes in the system are routed through a chain of contiguous nodes to their destination.

Assume that the destination address is actually a valid single node and not a node with multiple identities. Further assume that all the routing nodes are non-malicious. The destination node can indeed maintain a small number of identities and have each one routed through a different portion of the network. But this does not change the basic fact that the number of identities is bounded when using the Symmetric defense.

Assume instead that one of the routing nodes is malicious. That node can correctly represent its identity to its neighbors, performing all required challenges. If the network routes messages through its component nodes then a malicious node can generate multiple identities and claim that those identities are its neighbors. It can then route message to and from these false identities with no further validation. In this way, a self-replicating node can create an unbounded number of network identities.

Assume that a node must be validated by all nodes less than a fixed number H of hops away. If the number of challenges answerable by the attacking node is greater than $H * (H - 1) * (H - 2) * \dots * 1 = H!$, then the attacking node can create a chain

of length H and present an unlimited number of uncheckable identities behind that chain.

If we require all nodes to validate all other nodes then we are once again forced to recognize that the Symmetric defense limits ALL nodes to a small number of validated nodes and hence we have significantly restricted the size of the network.

In summary, the Symmetric defense is usable only where the total number of nodes in the network is small enough that all nodes can directly validate all other nodes.

6.2 Probabilistic Defense

Assume that at every time period T , each node in the system randomly chooses another node to challenge. Each node is challenged on average once each time period. A self-replicating node maintaining K identities would thus be challenged on average K times each time period. Since each node has a limited number of resources, a self-replicating node would be unable to maintain an infinite number of identities.

There are two major problem with the probabilistic defense. One relates to how to publish the results of each challenge. The other concerns the ability of a malicious node to abuse the challenge mechanism.

Assume that an identity failed a challenge and is thus believed to be false. The node(s) that assigned the challenge believes that the identity is false, but has no safe way of distributing that information to other nodes. Unfortunately, all known mechanisms can also be used by malicious nodes to arbitrarily eject good nodes from the system.

Alternatively, malicious nodes can use the challenge system to selectively kick out legal nodes. Since malicious nodes are not restricted to choosing challenges randomly, they may collude to challenge any given node or set of nodes. These target nodes can then be trivially overloaded regardless of their resources.

The following exposition details the probabilistic aspects of this protocol, but is essentially anecdotal because of these two flaws. It is unclear why one would deploy such a system as a self-replication defense when the possibility of abuse is even worse than that self-replication attack itself.

6.2.1 Analysis

It is clear from the basic algorithm that a self-replicating node would be able to maintain only a limited number of identities based on its available resources. One limitation of this algorithm is that because of its probabilistic basis, it is possible that legal nodes will be believed to be malicious.

If all nodes in the system are close to the P_{min} , the minimum capability necessary to join the system, then as we will show, a number of good nodes will be ejected during each round.

Theorem 1 *In a system with N nodes, with high probability, at least one good node will be thrown out each challenge period if the resources of an average node is less than $P_{min} \frac{\log n}{\log \log n}$. Furthermore, no good node with more than $P_{min} \log n$ resources will ever be thrown out.*

PROOF. Assume that each node chooses its challenge node uniformly from the set of nodes. This is equivalent to randomly assigning n balls into a n bins, where each bin represents one of the nodes in the system and a ball represents a challenge.

It is well known that with high probability, at least one node will have at least $L = \frac{\log n}{\log \log n}$ balls. That is, at least one node will be challenged more than L times. The minimum resources necessary to answer a single challenge is P_{min} . Thus if the average node has less than $P_{min} L$ resources then, with high probability, some node will be ejected at least every other round.

The other well known bound on this analysis is that with very low probability, no bin will have more than $\log n$ balls. Thus if any single node has $P_{min} \log n$ resources than it can respond to $\log n$ challenges and it will never fail a challenge round.

□

This analysis suggests a policy for setting the minimum cost of each challenge. The log of the number of nodes in the system grows reasonably slowly. A designer can thus require that the average node have P resources such that $\frac{P}{P_{min}} > \log n$. These additional resources may be used as computational resources in the system during those times that the node is not being challenged.

This bound also suggests that a malicious node with $P = P_{min} \log n$ resources will be able to maintain $\log \log n$ identities in this system.

Chapter 7

Sisyphus Defense

This section describes a new defense against self-replication attacks. The defense is called Sisyphus after the character in Greek legend who was doomed to ceaselessly roll a rock to the top of a mountain only to have the rock roll back down of its own weight. Similarly, in this protocol each node may interact with the system only as long as it is performing challenges. If a node stops responding to challenges, it loses its identity.

As was shown in the previous chapter, Computation Challenges alone provide only part of a full solution. The Sisyphus protocol extends these challenges with the notion of a randomly selected vouching-set for each node. As will be shown, this construct provides the missing pieces necessary for a complete solution.

This chapter first describes the protocol itself, followed by a discussion of its correctness and formal bounds. All discussion in this section are based on the system model in chapter 3.

7.1 The Sisyphus Protocol

Assume that when a node enters the system, some portion of its address is defined by node's position in the network. This assumption is consistent with an ad-hoc mobile network and with the Internet. Although some portion of the network address is defined by the physical network, a node is free to select for itself all other portions of the address. This is equivalent to a node being part of a class C IP network and

being able to select the lower 8 bits of its address. When a node enters the system, it generates a random identity string based within its fixed address space. If it attempts to choose any other address, it will be inaccessible to the rest of the network.

Once a node has decided on a network address, it repeatedly executes a well-known hash algorithm to generate a bounded list of k random network addresses. This set of network addresses may be mapped to a group of nodes V that is called the Vouching-Node set. Given a network address, any node within the system can re-generate this exact Vouching-Node set. There is no expectation or guarantee that these addresses actually exist within the network. A mapping mechanism will be required to associate the pure hash generated values with actual system nodes. Subsection 7.2.3 describes a number of possible mapping algorithms.

Upon entry to the system, a target node may sequentially contact each member of its own vouching node set and request a cryptographic challenge. Alternately, a node may initiate the challenge mechanism on a target node by requesting the challenge state from that target nodes vouchers.

The target node performs the challenge for each vouching node. As is the case with cryptographic challenges, a vouching node can verify the result within linear time constraints. The successful challenge is then accepted as an identity certificate for the challenging vouching node. The union of the vouchers challenge results forms the basis for trusting a node's identity with a simple majority of the results forming a consensus.

Each identity certificate has an associated limited time-to-live (TTL) lifespan that is recorded by the vouching node. Within a period of time before the expiration of the certificate, the vouching node must generate and deliver a new challenge to this node. Vouching nodes maintain no additional state other than the certificates for those nodes that they directly verify through challenges.

Any two nodes Alice and Bob that wish to exchange data must first verify each others identities. The protocol is symmetric, and thus with no loss of generality, it will be presented from Alice's viewpoint.

Alice requests a report on the veracity of Bob's identity from each member of Bob's Vouching Node set. Each of Bob's vouching nodes that is queried will check for the existence of certificate for Bob. If the certificate exists and is still valid (within

the TTL), that node will certify Bob's identity and return the TTL of the existing certificate. If there is no certificate then the voucher will immediately challenge Bob and generate a new certificate. If there is no challenge, the node returns a negative result.

Alice counts the results and accepts Bob's identity if and only if a majority of Bob's Vouching Nodes returned positively. Communications with Bob is valid only during this period. Alice may at any time request a re-confirmation of Bob's identity. Alice should re-check Bob's identity once the TTL of the certificate expires.

7.2 Discussion

Each entity attempts to present an identity to other entities in the system. If an entity e successfully presents an identity i to another entity l , we say that l accepts i .

Let the identity of a node X be represented as ID_X . Multiple identities maintained by a node X will be represented as $ID_{X-1}, ID_{X-2}, \dots, ID_{X-n}$. The Vouching Node set for a node X is represented as V_X .

As discussed in Section 3.1, the node population may be divided into three groups. Legal nodes, Malicious nodes and Passive Malicious nodes. Self-replicating identities are assumed to minimally belong to the set of passive malicious nodes.

7.2.1 Assumptions

Assumption 1 *The physical network location of a node defines at least N bits of the address of each node.*

This follows from the specification of the system model. It applies to the Internet where addresses are allocated by network providers as subsets of the IP address space. It applies trivially to hypercube architectures where all addresses are fully defined by the network. It applies to ad-hoc mobile networks because of the need for physical routing across long distances.

Assumption 2 *The attacker has limited physical resources and limited access to, at most, P percent of the physical network. The attacker can perform at most C challenges in a given time period T .*

Assume that the attacker has limited physical resources and network access. If this was not so then the attacker could control an arbitrarily large percentage of the network by installing new unique malicious nodes. Unlimited access to the network gives the attacker unlimited control of the routing and address allocation within the network.

Assumption 3 *The network consists of at most B percent malicious nodes.*

Assume that the number of malicious nodes is bounded and is strictly less than the number of good nodes. If this were not the case, then malicious nodes could control and corrupt any distributed protocols or algorithm that the good nodes might attempt to perform.

7.2.2 Analysis

The goal of this algorithm is to bound the number of identities that a given node can present to other nodes within the system. This section shows that the algorithm correctly enforces this behavior and that the number of presentable identities is bounded.

Consider the problem of proving that each Vouching Node set is trustworthy. The proof argues that a majority of those vouching nodes must be trustworthy because they are selected at random from the network population.

Lemma 1 *If k bits of the address are fixed by the underlying network, then the attacker's ability to masquerade as more than half of the vouchers in its vouching set is probabilistically limited and conditioned on the size of the vouching set, the total number of bits in the address space and the number of networks that the attacker has corrupted.*

PROOF. Let Q be the total number of bits in the address. Let k out of Q preassigned bits be fixed for any given attacker. Let v be the number of nodes in the voucher set.

Let x_i be an random indicator variable such that

$$x_i = \begin{cases} 1 & \text{if attacker can masquerade as voucher } i, \\ 0 & \text{otherwise.} \end{cases}$$

The hash function takes this address and produces a set of addresses uniformly distributed over the address space. Assume than an attacker has compromised a total of n networks in an attempt to improve his chances. The probability of any given address in the vouching set having exactly these values of the k bits within any of the attacker's address spaces is $P[x_i = 1] = n2^{-k}$.

Let V_b be the number of vouching nodes that the attacker can control. That is, the number of nodes in the vouching set that have exactly these k bits the same as the attacker's k fixed bits.

$$V_b = \sum_{i=1}^v x_i \text{ and } E[V_b] = vn2^{-k}$$

Set ϵ such that:

$$(1 + \epsilon)E(V_b) = (1 + \epsilon)vn2^{-k} = \frac{v}{2}$$

and thus

$$\epsilon = \frac{2^{k-1}}{n} - 1$$

Applying The Chernoff Inequality, we see that

$$\begin{aligned} P(V_b \geq 1 + \epsilon E(V_b)) &< e^{-\frac{\epsilon^2 E(V_b)}{3}} \\ &= e^{-\left(\frac{2^{k-1}}{n} - 1\right)^2 \frac{vn2^{-k}}{3}} \\ &= e^{-\left(\frac{2^{2k-2}}{n^2} - \frac{2^k}{n} + 1\right) \frac{vn2^{-k}}{3}} \\ &< e^{-\frac{v}{3} \left(\frac{2^{k-2}}{n} - 1\right)} \end{aligned}$$

From this inequality, it is clear that 2^{k-2} must be sufficiently larger than n . That is, the number of networks potentially represented by the fixed bits in the address space must be significantly larger than the actual number of networks controlled by the attacker.

Furthermore, one can see that by setting v sufficiently high, it is possible to decrease the probability of success as low as required.

This probability applies for any single address that the attacker creates. Since only k bits of the address are fixed, the attacker has the ability to select new addresses by permuting the other available bits and running the result through the hash function. The attacker succeeds if any one of his available addresses generates at least $v/2$ vouchers within any of the networks that the attacker controls.

Note also that the hash function must be available to all participants so that they can compute a node's vouchers without generating load on any other system component. The hash function may not be computationally expensive, hence this attack can be performed offline. \square

This proof shows that it is possible to arbitrarily reduce the probability of an attacker owning its own vouching set. In order to implement this attack, a node would select a possible address, and perform the v hash evaluations necessary to identify its vouchers. The node would then check to see if these vouchers were in its own addressable space.

Although computational bounds are not involved in this proof, it can be shown based on the Chernoff bounds in this proof that the attacker will need to process on average $e^{\frac{v}{3}(\frac{2^{k-2}}{n}-1)}$ addresses before finding an address that allows the attacker to control more than half of his vouchers. Typical Internet addresses are allocated from class C domains and thus the number of fixed bits is $k = 16$. If the size of the voucher set is $v = 5$ then the attacker will need to try approximately e^{65} addresses in order find one of the small set of addresses that will allow the node to present its own vouching set.

It should also be noted that the number of possible addresses available to the attacker is also limited to $n2^{Q-k}$. The voucher may run out of possible addresses before a solution is found, particularly if:

$$n2^{Q-k} < e^{\frac{v}{3}(\frac{2^k-2}{n}-1)}$$

This proof applies only when nodes select vouchers based on direct physical addressing. In real networks, a very large portion of the address space is unused or unreachable. Some sort of routing mechanism must be created to automatically route addresses to the nearest extant node. This routing mechanism must operate in such a way that it does not allow an attacker to subvert the previous proof and to become its own vouchers.

This lemma shows that for appropriately sized voucher sets, an attacker is highly unlikely to control its own vouchers. Since the majority of the attacker's vouching nodes are legal, other nodes can rely upon them to truthfully report the status of the attacker's challenges. The attacker's identity is thus valid as long as it has performed the appropriate challenges within the maximum certificate TTL.

A legal node also benefits from this proof in that it has been shown that the vouching set is indeed randomly distributed across the system, thus minimizing the chance that any given adversary can control more than half of the vouchers for any given node.

It is now possible to prove that the number of identities that an attacker can maintain is bounded.

Theorem 2 *An attacker can maintain a bounded number of identities dependent only on the availability of the attacker's local resources.*

PROOF. By assumption 2, an attacker has finite resources with which to perform these challenges. These resources can be represented as the ability to respond to X challenges.

Under the Sisyphus algorithm, each set of vouching nodes for each and every identity will issue exactly K challenges every time period. In order for a node to maintain even one identity, we have that

$$X \geq K$$

There exists some number of identities $n \geq 1$ such that

$$X \leq nK$$

And thus an attacker can not maintain more identities than

$$n \leq \frac{X}{K}$$

□

Implementations of the Sisyphus algorithm should consider the trade-offs between the maximum number of sustainable identities and the strength of the minimally capable node.

It is important for vouching nodes to coordinate their challenges such that the chance of two or more challenges occurring simultaneously is minimized. In such a case, the challenged node may be unable to maintain even one identity as it may fail one or more challenges due to resource limitations.

Note also that as the number of identities maintained by Bob increases, the amount of work that Bob can perform outside of challenges decreases. Hence there is a natural balance on the side of a self-replicating node in terms of the number of identities that can be maintained while simultaneously performing local tasks.

In summary, it has been shown in Th 2, there is a fixed upper bound on the number of identities that an attacker may maintain. Lemma 1 proved that no node or groups of nodes can collude to offer an unlimited number of identities to any other node unless the network is itself compromised.

The maximum number of supportable identities is bound only by the resources of the attacking node. The size of the network is unrelated to this bound. Assumption 2, was specified in terms of the percentage of nodes in the network that are malicious. As the network grows, the attacker must work much harder to maintain the same percentage of malicious nodes.

7.2.3 Routing

The Sisyphus defense employs a hash function to generate the addresses of the Vouching Nodes. At issue is the relationship between these generated addresses and the addresses of actual live system nodes that can act as vouchers. If the space of hash function is 32bits, then there are some four billion possible values. Even if all the

nodes in the largest network today (The Internet) were accessible, only 10% of the nodes would actually exist.

One might consider mapping a vouching set to actual nodes by using a Distributed Hash Table (DHT) such as Chord [26]. These systems are based on dynamic routing of addresses to the nearest actual live node, where *nearest* is a metric defined by the system itself.

Solutions of this type based on existing DHTs would be seriously flawed. By design, DHT nodes have very low in- and out-degrees. A self-replicating node within the network would have control over a significant portion of the network through its ability to route messages to known malicious nodes or to handle those messages itself. The low node degree serves to minimize the message path between any two nodes, but it also allows an attacker to corrupt a significant portion of the system with very little effort.

One approach is to utilize a DHT style routing mechanism that is based on a high node degree. Such a system would dynamically identify the active node closest to the voucher node's address, but be less susceptible to routing attacks. The design trade-off in such systems would be an extremely high message rate because each message must be routed independently through simultaneous paths. If messages were not routed in this way then it would be possible to capture an intermediate routing node and subvert the voucher mapping system.

An alternative is to assume the existence of at least a small number of trusted nodes. Each node as it enters the system would perform a Sisyphus join with the trusted nodes. Since these nodes are trusted, the join need not be symmetric, that is, only the node wishing to join the system would need to be verified, and not the trusted nodes themselves. The trusted nodes would maintain a list of all nodes in the system which can then be selected using a common metric from the hash generated vouching nodes addresses.

The trusted core is minimally required to accept node registrations and to perform the mapping function. A malicious node must be unable to corrupt a trusted node such that it controls the mapping function.

There must be no way for the attacker to corrupt the mapping function through external stimuli. To perform the mapping function, a trusted node must be willing

to accept registrations mapping an identity to a routing address. These identities are the only available targets for the mapping function.

The Sisyphus algorithm assumes a relationship between identity and routing address. This relationship may be trivially checked by the trusted node, rejecting all invalid identities. It was shown in Lemma 1 that the attacker cannot subvert the system using valid addresses, hence an attacker can control and insert as many valid mappings as it wants without subverting the system.

If an attacker inserts valid identities which it does not control, then at worst it will introduce dead nodes into the system. Such nodes will occur naturally as nodes enter and leave the system. The trusted core may employ a polling mechanism to check the liveness of system nodes. Whenever a mapping is requested, the trusted core can first check if the resulting node is live. Only actual live nodes would be returned as vouching nodes.

The system must have a mechanism to handle node churn. A node's vouching set will change as members of that set naturally enter and leave the system. From the perspective of the target node, over time, communication partners will not receive identical vouching sets for this node. The new additional nodes will increase the number of challenges from the original set K to some larger set. The target node should be able to handle some additional challenges as long as there is some leeway in its available resources.

An extension to the algorithm would allow the target node to request that its challengers recompute the vouching set. Changing the vouching set does not weaken the algorithm as the new vouching set would still be randomly selected and hence trusted. Furthermore, the target node will not be recognized by other system nodes until it responds to challenges from the its new vouching set.

One may ask "If there is already a trusted core, why not simply perform all the computational challenges in the trusted core itself"? There are at least three issues with such an approach.

As the size of the trust core increases, so does the complexity and administrative challenge of maintaining its trustworthy status. Maintaining 99.999% reliability is much harder than only 99% reliability. The additional challenges impose a natural limit to the size of the trusted core. In the Sisyphus algorithm, the trusted core

is involved only when a node initially joins the system or initiates communications with a new node. It is reasonable to assume that such events represent a very small percentage of system activities. If the trusted core were required to perform node challenges, then it would be much more involved, challenging nodes as certificates expire and voting for node identities. The added burden would be significant and would impact the scalability of the system.

Secondly, as currently specified, the trusted core need not be highly available. Since it is involved only on node entry or new communications, the unavailability of the core would not impact ongoing system operations. Requiring the trusted core to directly challenge each node would mean that if the trusted core was unavailable then the system would halt operations as soon as the certificates expire.

Finally, the trusted core could be made unavailable by way of a directed denial of service attack. Each vouching set is also susceptible to this attack, yet if a given vouching set is unavailable then only that target node is affected. Shutting down the trusted core would impact the status of all nodes.

7.2.4 Scalability

The proposed algorithm has no upper bound on the number of nodes that one node can interact with. The limiting factor is the ability of that node's vouching node set to query the vouching nodes of the participants. No limit has been set on the network capacity, hence, at least theoretically, there is no limit on the nodes that one can interact with.

The network has no limit on the number of nodes. Each node's vouching node set is chosen uniformly and randomly from the network. The burden of maintaining the system is spread uniformly across the network.

7.2.5 Denial of Service attacks

There is no DOS attack available to either the initiating node, the target node or a given vouching node. An initiating node may request an unlimited number of challenges concerning a target node. The target node's vouching node set will cache the TTL of their latest result and return immediately to the initiating node, thus

generating no additional system load on the target node. The load on the vouching is limited to a table lookup and hence is minimal.

A vouching node cannot overload a target node because the target node can legally ignore any one challenge. This follows because only a majority of vouching nodes need to have a valid certificate in order to vouch for a target node's identity.

Finally, the target node cannot overload its own vouching node set because the protocol is specified in the reverse. They challenge the target node, and not visa-versa.

Chapter 8

Practical Results

As part of this thesis, a working simulation was created of the Sisyphus system and a number of possible attacks. This chapter describes that simulation.

8.1 Simulation

A simulator was written in C++ using a standard event-driven model. Each event is an object with a requested execution time. Events are stored on a priority queue and executed in ascending order through the life of the simulation.

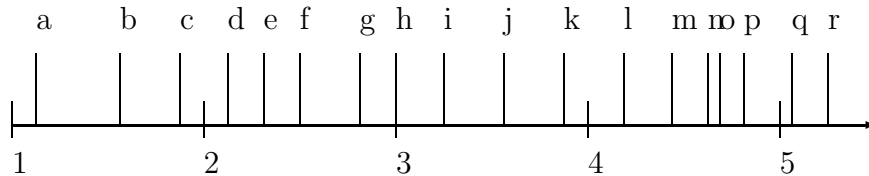


Figure 8.1: Time Units

The system uses a double word value to store the execution time and assigned special significance to whole numbers. Each unit number begins a new virtual time slice. The current time value is accessible via the SystemConstants package. Events may be scheduled for execution at any point during this or any later time unit. Figure 8.1 shows the time units *one* through *five* and the events *a* through *r*.

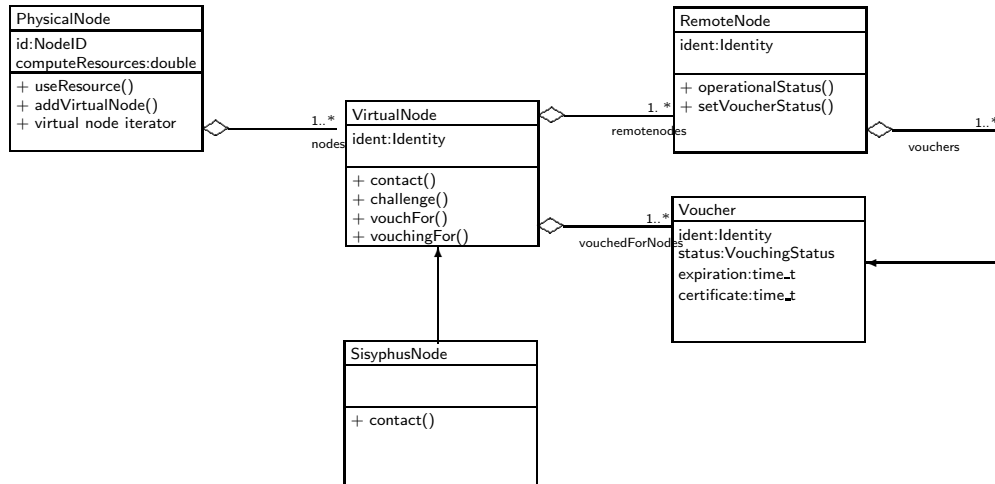


Figure 8.2: Simulation Classes

Figure 8.2 shows the relationship between each of the system classes. Each of the classes will be described below along with their relationships.

8.1.1 PhysicalNode

The PhysicalNode class represents a basic network device. The device has two properties; an id which uniquely specifies this object and a resource value which represents the resources available to this node at any given time.

Resources are assumed to be renewable. There are actually three values associated with each resources; the maximum resource value, the current resource value and the last time that the resource was used. A resource may be used through the useResource() method. If the last time that resource was used is more than one full time unit, then the resource is first reset to its maximum value and only then are the requested resources consumed.

Each PhysicalNode maintains a list of VirtualNodes that reside in this node. This represents a system node and corresponds to the possibility that more than one system node resides on a single physical network device sharing the same underlying resources. A potential self-replication node would have more than one VirtualNode per PhysicalNode.

8.1.2 VirtualNode

Each VirtualNode object represents a system node. A VirtualNode resides on exactly one PhysicalNode and utilizes the PhysicalNode's resources to satisfy system challenges.

VirtualNodes support four basic operations; *contact*, *challenge*, *vouchFor* and *vouchingFor*.

The *contact* operation is the basic method for initiating a communication link with another system node (as this is a simulation, no actual communications occur). The purpose of this call is to provide a general mechanism for executing challenge protocols when two nodes begin mutual communications. The default implementation performs no operations and returns success.

The *challenge* operation represents the action of performing a cryptographic challenge by this VirtualNode. The challenge consists of attempting to consume a standard number of resources from the underlying PhysicalNode. The challenge operation returns a boolean success value. This operation may not be over-ridden by derivative classes.

The *vouchFor* operation tells this VirtualNode to act as a vouching node for a specified identity. The operation allocates and returns a Voucher object for the target identity.

The *vouchingFor* operation is used to check the vouching status of the specified identity. The default implementation performs the Sisyphus voucher algorithm. It challenges the target node if and only if the node was previously in the STARTUP state or if the node was in the GOOD state but the certificate has expired. The operation updates the status, certificate and expiration values on the returned Voucher object.

As each node in the system can perform two basic functions, so too does the VirtualNode maintain two sets of state values. For the standard system interaction of *contact*, each VirtualNode maintains a list of RemoteNode objects. Each RemoteNode object represents one unique identity that this VirtualNode has attempted to contact.

As a Vouching node, each VirtualNode maintains a list of Voucher objects, one for each of the nodes that it Vouches for.

8.1.3 SisyphusNode

The SisyphusNode object is a subclass of the VirtualNode object. A SisyphusNode replaces the *contact* method with the Sisyphus protocol. The algorithm calls the *vouchingFor* method on each of the Vouching nodes in this RemoteNode object. Finally, it checks the operational status of the RemoteNode in order to decide if the contact was successful.

8.1.4 Voucher

An object of the Voucher class is used to represent two different states within the system. One state is that of that vouching status within a voucher node. In this case, the values represent the current belief of this voucher node for the identity in question.

The second state is that of a VirtualNode that once communicated with this identity. The Voucher object represents that last communications with this voucher node. Somewhat confusingly, the identity of the target node in this case is maintained on the RemoteNode object. The identity in the VirtualNode object is that of the vouching node itself.

There are no significant methods on the Voucher object. Its purpose is simply to record status information.

8.1.5 RemoteNode

A RemoteNode object represents an attempt to contact a given identity. The RemoteNode encapsulates the status of the last contact event. It maintains a list of vouching nodes for this identity and a summary status.

The RemoteNode object does not actually perform the vouching operation. It provides a recording mechanism through the *setVoucherStatus* method. This method records the given Voucher's status about the identity in this RemoteNode.

The *operationalStatus* method reviews the vouching status of each of the known vouching nodes and summarizes the result as one of STARTUP, GOOD or BAD.

8.2 Events

The simulation implements the following events. Each event schedules its next execution time with the simulation.

8.2.1 `clockTickEvent`

This event is initially scheduled for time 1 and reschedules itself for one time unit in the future. Currently, this event is used to maintain the global variable representing the current system time.

8.2.2 `connectionEvent`

This event randomly selects two `VirtualNode` objects within the system and has each node contact the other. It records the result of each contact. If the simulation has been defined to maintain ongoing communications between nodes and this contact was successful then a `reConnectionEvent` is scheduled for one time unit in the future. A unique `reConnectionEvent` is scheduled for each contact.

The `connectionEvent` reschedules itself such that the average number of `connectionEvents` per time unit matches that defined by the operator for this simulation run.

8.2.3 `reConnectionEvent`

The `reConnectionEvent` implements a simple contact from one `VirtualNode` to another. As opposed to the `connectionEvent`, the `reConnectionEvent` is not symmetric. Contact is only one way between the two nodes.

The event records the result of this contact. A time-to-live value is maintained for each connection. If the value is still non-negative then this `reConnectionEvent` is rescheduled for one time unit in the future.

8.3 Operation

The operation of simulation consists of setting the parameters for the number of connection operations per time unit and the length of each communication in terms of time units. The length of the simulation is specified as a whole number. An initial `clockEvent` and `connectionEvent` are scheduled and the simulation is started. Events are executed, rescheduled, and scheduled until the target end-of-simulation time is reached.

8.4 Attacks

Two simple attacks have been implemented on this system. As expected, each attack is immediately identified and the corresponding identities marked as false.

8.4.1 SimpleAttack

The `SimpleAttack` method creates a single `PhysicalNode` with a large number of `VirtualNodes`. At each `SimpleAttack` event, one of the self-replicated identities is selected along with a random `VirtualNode`. Both nodes attempt to *contact* each other. The `VirtualNode` which is attacking does not bother to check the status of its target. The target node challenges the attacker.

The `SimpleAttack` event reschedules itself such that the average number of attacks per time unit matches that defined by the operator for this simulation run.

As expected, the self-replicating node can maintain only as many identities as it has the resources to respond to challenges.

8.4.2 CollectAttack

The `CollectAttack` attempts to perform exactly one contact event per time unit.

The `CollectAttack` method creates a single `PhysicalNode` with a large number of `VirtualNodes`. At each time unit, a different self-replicated identity is selected [time modulo the number of replicated identities] along with a random `VirtualNode`. Both

nodes attempt to *contact* each other. The VirtualNode which is attacking does not bother to check the status of its target. The target node challenges the attacker.

This attack shows that the system must perform vouching challenges during each time unit. When not implemented, the attacker can maintain each of the replicated identities since only one is challenged during any given time unit. When regular challenges by the vouching nodes are implemented, the attacker is unable to maintain more identities than those supported by its local resources.

8.5 Churn

In actual systems, the set of system nodes is not static. Nodes frequently join and leave distributed systems. This changing set of nodes is called Churn. The simulation was extended to model exit and entry of nodes at each time step. Two events were added, `addEvent` and `removeEvent`. At the beginning of each time step, each event randomly adds or removes a fixed number of nodes as defined by the configuration file.

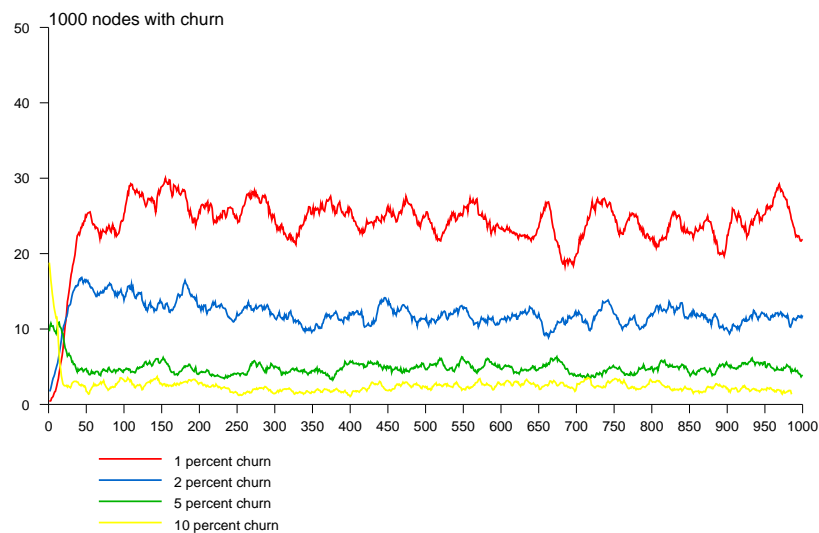


Figure 8.3: Average false negatives with 1000 nodes and 1000 connections per time unit and 1%, 2%, 5% and 10% churn

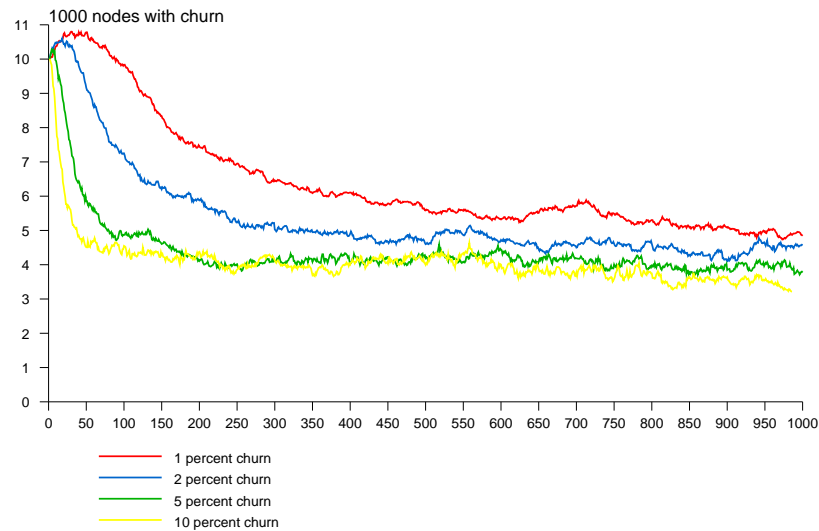


Figure 8.4: Average Voucher Set size with 1000 nodes and 1000 connections per time unit and 1%, 2%, 5% and 10% churn

The simulation was used to explore the behavior of the system under churn. Specifically, what would happen to legal nodes? Figure 8.3 shows a system with 1000 nodes, with 1000 random connections between nodes during each time slice. The three lines show increasing amounts of churn. At 1% churn, on average, a relatively large number of nodes are ejected from the system because they fail their challenges. Surprisingly, as the churn increases, the number of false negatives decreases. It should be noted that even under the worst case of 1% churn, the system ejects less than 0.3% of the nodes during any given cycle.

The Churn results reflect the mechanism used to manage and maintain the list of vouching nodes for any given initiating node. The process begins any time an initiating node attempts to contact a target node that it has not yet contacted. The initiating node contacts the mapping authority, identifying the vouchers for that target node. The initiating node records this list for use at any time in the future.

The list is not changed by the initiating node. Status data is maintained for each voucher on the list. A voucher's status may be marked as DEAD if the voucher leaves the system or stops responding to requests. The target node is marked as false at

any point that a quorum of positive voucher votes can no longer be generated.

In principle, there are two ways to generate a false negative. A target node may have too few vouchers, or too many vouchers.

If enough of a given target node's vouchers leave the system then the initiating nodes cannot form a quorum of responses. The target node is then assumed to be malicious.

On the other hand, the addition of a sufficient number of new nodes to the system can also generate false negatives. Whenever one of the new nodes attempts to initiate contact with the target node, that new node will contact the mapping authority for a set of vouchers. Yet, since there are so many new nodes, the mapping between the hashed values and the nearest actual nodes will no longer produce the same results as earlier mappings. This process increases the number of nodes that challenge and vouch for a target node resulting in an overload of challenges. The target node has limited resources and will thus fail some number of these challenges.

It may be thought that the average number of vouchers would be related to the incidences of false negatives. Figure 8.4 shows that although the average number of vouchers per node decreases over time, there is no correlation with the relatively constant number of false negatives per cycle.

Review of the simulation data suggests that the main reason for false negatives is that, over time, a nodes vouchers leave the system and hence there are insufficient vouchers to maintain a positive quorum. We have considered a number of possible solutions to this problem.

In the first solution, the initiating node manages the vouchers. If the initiating node detects too many dead vouchers, then it can notify all of the vouchers that they are no longer to vouch for this target node. The initiating node retrieves a new voucher list and requests those node to vouch for the target node.

A vouching node refrains from challenging the target node only as long as no initiating node has requested its status. In this way, initiating nodes can reasonably refresh their vouching lists without enabling an attacker to disrupt the system.

One problem with this approach is that according to the protocol, vouchers must challenge the target node each cycle regardless of whether an initiating node has requested its vouching status. Nodes that are not contacted may be able to generate

additional identities as the old vouchers leave the system. These duplicate identities will eventually be detected if and when the old identities are re-contacted.

The second solution transfers control to the vouchers. On a regular basis, a vouching node can check with the mapping authority to see if it is still a voucher. If it is no longer a voucher, then it stops performing challenges and sets its status to REPLACED. Initiating nodes that receive this status will query the mapping authority for a new set of up-to-date vouchers.

This option generates additional traffic at the mapping authority due to the regular checks by the vouchers themselves. In the first solution, vouchers continued to challenge a target node unless explicitly notified by an initiating node. In this second solution, vouchers can themselves cease challenging a target node based on the latest mapping results. This would allow a target node that is un-contacted to generate multiple identities as its resources for the old identity would no longer be consumed.

A final option is to have the vouching nodes regularly check with the mapping authority as in option two. When that voucher is no longer an appropriate voucher for the target node, it would stop performing its challenge and notify the new vouchers to begin challenging this target node. In this way, not only are old vouchers removed from the vouching set, but the new vouchers are automatically refreshed. A target node would have little opportunity to remain unchallenged.

A window of opportunity exists in both the second and final option based on the frequency of the voucher's communications with the mapping authority. Increased frequency would reduce the chance of a node having dead or duplicate voucher at the cost of increased load and traffic to the mapping authority. This approach also increases the system's dependency on mapping authority. What should a voucher do if the mapping authority fails to respond?

Alternatively, decreased frequency of updates allows a target node to generate and maintain multiple identities, even if these identities are only valid for a restricted period of time.

In summary, Churn remains a significant design challenge to implementations of the Sisyphus system. The low incidence of false negatives may not be an issue in systems with very low churn or very high churn.

Chapter 9

Related Work

This chapter presents a survey of works related to managing identities of nodes in a distributed system. Most published research first presents a system and threat model and then suggests one or more potential solutions. Our approach here is to describe the standard system and threat models in sections 9.1 and 9.2. The rest of this chapter will review published research and solutions as they relate to those models.

9.1 General System Models

Identity management in distributed systems is important only as it impacts the scalability, performance or service levels of the system. Some systems such as Freenet [5] are intentionally designed to maintain node anonymity and yet provide accepted service. The designers of Freenet intentionally choose a trade-off between anonymity and the impact of self-replication attacks.

Many systems are designed to provide availability guarantees. These systems are typified by Peer-to-Peer file sharing systems. Their goal is to provide access to files and services provided by each node. The internal contents of a given node are irrelevant as long as that node provides the required services.

Other systems are designed for some form of Secure Storage, such as Distributed Key Escrow or Distributed file storage. A defining characteristic of these systems is that data or services are provided through cooperative activity of the system nodes, but the final results are legally available to only a small subset of nodes. Such systems

may distribute data in order to guarantee that no single node or subset of nodes has access to enough segments of the data to reconstruct the complete object.

The Secure Storage group also includes systems that perform Secure Function Evaluation (SFE). These systems evaluate known functions without exposing the function inputs or intermediate results.

We will refer to these two models as type A for Availability systems and type S for Secure Storage or Evaluation systems.

9.2 Attacks on distributed systems

A number of attacks are common to almost all types of distributed systems. In this section we will explore the range of such attacks and note their applicability to our two system types.

In any distributed system, there is a risk of the constituent nodes halting or misbehaving during system operation. Such faults have been categorized as Random faults [20] and Fail-stop attacks [20]. The loss of these nodes may be due to influences outside of the systems control. For example, one or more nodes might fail due to a power outage. All current distributed systems are designed with such failures in mind. The focus is on localizing the impact of the loss of these nodes such that the number of nodes affected by the loss of specific nodes is a linear function of the number of lost nodes.

By far the most common and effective attack on distribute systems are Denial of Service (DOS) attacks. In both type A and type S systems, such attacks suffice to deny the members of the server access to system resources. DOS attacks usually consist of flooding one or more system nodes with traffic such that those nodes become unavailable to the rest of the system. By denying these services to the system, the attackers hope to disrupt the entire operation.

There are a number of communication techniques for limiting the effectiveness of DOS attacks. Most current research in self-replication defenses choose to ignore specific defenses against DOS attacks. At most, researchers attempt to distribute all key services such that the loss of some number of service nodes does not result in a total system failure.

The previous two issues dealt with the loss of system nodes and resources. Another area of attack is called a Spam Attack [20]. Here we may assume that system nodes are active and operational. Malicious nodes attempt to subvert the system by providing false or inaccurate information to other nodes by using the existing system protocols. A simple example is the delivery of an intentionally corrupt file in a file-sharing system. The transfer itself operates according to all the system rules. Yet the process is flawed since the resulting data is worthless.

All distributed systems have some mechanism for routing messages between nodes. Such mechanisms may be part of the underlying network fabric or may be implemented by the system itself in the form of an overlay network. In general, underlying network failures are assumed to be indistinguishable from node failures and hence their affect should be limited to the loss of those nodes.

Overlay networks introduce a new failure type that we call Routing attacks. Control of a routing node may enable additional attacks including re-routing packets to unintended recipients or handling packets locally instead of routing them to their intended recipients. Systems which defend against routing attacks may implement controls to encrypt messages such that only the sender and recipient can understand the message content. Other approaches include having multiple routing paths such that the control of a small number of paths will not deny access to an large fraction of the system.

The attack mentioned in [10] may be categorized as a Lurking attack. The previous attacks attempt to deny access to resources or nodes. They may also attempt to destroy or corrupt existing data or operations. Lurking attacks are less obvious in that they attempt to violate privacy constraints. Such attacks attempt to discover private information through participation in the network. Malicious nodes maintain all system protocols and perform as expected.

Lurking attacks may be more effective in distributed systems because collusion between otherwise legal system nodes can result in unintended information sharing and hence in exposing private data. Self-replication attacks decrease the cost of this collusion such that all collusion data is available locally to the attacker. In such an attack, no externally visible network traffic need occur in order to collude. Unless the system can restrict the number of self-replicating identities, there will be no external

evidence that a Lurking attack is occurring.

9.3 Centralized Solutions

The surest way to implement identities in distributed systems is to impose those identities from outside of the system. This removes the onus of identity management from the system itself to some external entity which is not bounded by the system constraints. In Farsite [3], a centralized certification authority is predicated for this purpose. Nodes that wish to join the system must convince a human operator to allocate a new identity and its associated cryptographic keys. In practice, Farsite utilizes the Windows Domain registry as its certificate authority. Registration requires the presentation of an appropriately capable account and that account's password.

This centralized authority increases the cost of entry to the system such that nodes are unable to implement self-replication attacks without having administrative access to the certificate authority. It also provides a revocation service that allows nodes to check the current status of any other system node.

On the negative side, the Windows Domain registry represents a single point of system failure. It is susceptible to DOS attacks and to connectivity issues such as network partitions. It relies on the trust of its operators to maintain system integrity. It has been claimed [27] that over 80% of all security breaches are caused by insiders. If this is indeed so, then relying on human administrators may be more costly than the solutions proposed by this paper and by other distributed systems researchers.

9.4 Trust Based Systems

With the excitement and popularity of distributed file sharing systems such as Gnutella [2] and Freenet [5], significant research has focused on recognizing and identifying well behaving system nodes. Such nodes are said to be "Trusted" such that new nodes may reasonably interact with these trusted nodes to share files. Ideally, non-trusted nodes are those that either have no background within the system or that have previously delivered some corrupt file to other system nodes.

In order to implement a trust mechanism, it is necessary to have a mechanism with which to uniquely identify each node. In this way, it is possible to reference that node when events both positive and negative occur. In [19], a mechanism is proposed based on public key cryptography and one-way functions. The authors create a computationally expensive key generation mechanism which they claim makes identities both difficult to create and difficult to falsify. Revocation is performed by flooding the network with evidence of an untrusted identity.

In relation to self-replicating attacks, this method fails to limit the number of identities because of the ability of a node to collect identities over time. Even though the cost of creating an identity is high, there is no maintenance cost.

The XRep protocol [6] assumed that node identity is controlled through the use of public key encryption. In addition, all resources in the form of shared files are also signed using a hash functions. Each node maintains a table of both resources and other nodes, associating with each item a trust value. When a resource is located, the querying node broadcasts a vote request for both the resource and the supplying node. Peer nodes are expected to reply with their trust values. The querying node then collates the votes and updates its own trust tables.

The authors argue that both resource and node reputations are necessary to ensure efficient system operation. Resource reputation alone is insufficient because there are both significantly more resources than nodes in the system and because new resources are introduced on a frequent basis. Node reputation alone is insufficient because the node might be unknowingly serving a corrupt file.

The authors of this paper recognize the potential for self-replication attacks on trust values. They attempt to mitigate the issue by additionally associating node identifiers with the IP address of that node. Nodes are grouping into IP address groups and trust is shared between all nodes in each unique group. One challenge is to identify the correct scale for such IP groups. Clearly, using Class B Internet addresses with 65k addresses per group as the IP grouping is too gross since it will throw out too many correct nodes. On the other hand, assuming a Class C address with 256 addresses per group may be too limiting.

In the previous system, all nodes were allowed to vote on reputations. In [25], a system call TrustMe is proposed which attempts to enable anonymity for voting

nodes. The authors of TrustMe argue that one could perform a denial-of-service attack against nodes that voted against you. They base their system on providing anonymity for voters in order to protect votes from retribution.

The TrustMe system assumes that all communications between nodes is by broadcast. They stipulate that there is no overlay routing in order to avoid dealing with routing and snooping attacks. The system also assumes a bootstrap server which holds all public key signatures for each identity. For each node in the system, the bootstrap server allocates a set of randomly chosen vouching nodes which are required to listen for anonymous signed reputation reports for their nodes.

The TrustMe protocol defines reputation reports as a signed interaction between the reporter and the node in question. A cryptographic protocol is presented which interweaves signatures from both of these nodes to create an un-reputable report.

The vouching nodes will also respond to voting requests by broadcasting their local results. Only a small subset of system nodes are thus responsible for maintaining state. That subset can change as nodes enter or leave the system since the bootstrapping node can manage their efforts.

This proposal has a number of drawbacks. The bootstrap server is a critical resource for this system. It holds both the public key values for all nodes and the mappings for all vouchers. This node thus represents a single point of failure for the entire system.

The TrustMe protocol is susceptible to self-replication attacks by having one node create multiple identities (and hence multiple keys). Each fake identity can interact with its other fake identities to create positive reputation reports. As long as the number of fake interactions are greater than the real negative interactions, the trust value will be positive and hence the false nodes will be acceptable to the system.

The previous systems assumed both a broadcast capability for all system messages and/or a central authority for certificates and for identifying system nodes. The Eigentrust system [15] is similar to our Sisyphus protocol but relies on trust metrics instead of cryptographic challenges. The system is designed to provide file sharing services and as such all threats are to the ability to download files. The paper focuses mainly on the mathematics aspects of trust calculation but does provide an interesting approach to distributing this information within the system.

In the Eigentrust system, nodes are assumed to be organized under a distributed hash table (DHT). The reputation values for any given node are managed by a unique set of “score managers” which are allocated using multiple hashes into the DHT space. The benefit of this approach is that there is no need for a central authority to map hashed node values into actual nodes. Before communicating with a node, the requesting node should access the score managers for the destination node. Similarly, once the file transfer has completed, those same score managers should be updated with the transfer status.

The Eigentrust authors admit vulnerability to self-replication attacks and suggest that the attacks may be averted through the imposition of a high cost to generate node IDs on entrance to the system.

9.5 Self-replication defenses

Recent research into self-replication attacks has focused on unique challenge algorithms. Newsome et al. [21] present the special problems associated with Sybil attacks on sensor networks. Most of the reviewed problems are related to routing costs associated with such networks. Newsome suggests that we may utilize the assumption that each node has only a single radio source to identify self-replication attacks. They provide an algorithm which can identify self-replicating nodes because a radio cannot send and receive on different channels at the same time. During a given time slice, each node assigns its neighbors unique channels on which to broadcast. Legal nodes have no problem broadcasting on that channel. Nodes with multiple identities will be unable to broadcast on more than one channel and hence duplicate identities will be discovered.

Another aspect covered in Newsome’s research is related to the ability to implement self-replicating attacks on systems where the initial configuration of nodes is predetermined. In general, sensor networks are deployed once. There is little support for nodes which are subsequently deployed. In such a system, it is possible for the deploying agency to preallocate unique identities to each node and to share cryptographic key information at deployment time. The self-replication problem occurs because an attacker is assumed to compromise one or more existing nodes. That node

can then utilize the captured information to create and maintain multiple identities. The paper provides a number of cryptographic approaches and analysis showing that different key allocation strategies have unique defensive properties.

Finally, Newsome suggests that location information may also be sufficient to limit node identities. This suggestion is based on the limited radio range of each sensor node and the fact that in current sensor networks each sensor is non-mobile and hence statically located in space.

Chapter 10

Conclusions

This thesis has reviewed the topic of self-replicating attacks on distributed systems. The discussion began with an overview of vulnerable systems, followed by a discussion of the problems associated with this type of attack. A number of possible solutions were proposed along with the positive and negative features of those solutions.

The Sisyphus defense was described in detail with proofs of its effectiveness. A simulation model was created and implemented along with a number of simple attacks.

Finally, a review of attacks on distributed systems was provided, covering basic approaches to identity maintenance in current research.

As a final word to system implementors, I would suggest that in most systems, a centralized certificate authority is the appropriate mechanism for maintaining node identities. Although this requires careful attention to cryptographic and security details, it is sufficiently standard to be implementable on existing substrates.

Designers who cannot implement a centralized certificate authority should clearly understand their system requirements before embarking on a quest to limit self-replication identities. In most cases, the attacker can overwhelm a system using brute force resource intensive attacks. Today's news is full of script-kiddies and spammers that have created large pools of zombie computers for denial of service attacks and mail spamming. An attacker would be much more likely to use such a brute force attack than to use a self-replication attack. At most, self-replication would provide an added efficiency on top of the existing massed zombie attack.

Bibliography

- [1] M. Abadi, A. Birrell, M. Burrows, F. Dabek, , and T. Wobber. Bankable postage for network services. In *Proceedings of the 8th Asian Computing Science Conference*, Dec 2003.
- [2] E. Adar and B. Huberman. Free riding on gnutella. Technical report, Xerox PARC, Aug. 2000.
- [3] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, Dec. 2002.
- [4] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *5th Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, pages 299–314, Dec. 2002.
- [5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009:46–66, 2001.
- [6] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216. ACM Press, 2002.

- [7] A. Datta, K. Aberer, and M. Hauswirth. Handling identity in peer-to-peer systems. Technical report, Ecole Polytechnique Fédérale de Lausanne, Sept. 30 2002.
- [8] Y. Z. Ding and M. O. Rabin. Hyper-encryption and everlasting security. In *19th Annual Symposium on Theoretical Aspects of Computer Science*, volume LNCS 2285. Springer-Verlag, 2002.
- [9] D. Dolev. The byzantine generals strike again. Technical Report CS-TR-81-846, Stanford University, Department of Computer Science, Mar. 1981.
- [10] J. Douceur. The sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, Mar 2002.
- [11] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Lecture Notes on Computer Science 740 (Proceedings of CRYPTO'92)*, pages 137–147, 1993.
- [12] M. K. Franklin and D. Malkhi. Auditable metering with lightweight security. In *Journal of Computer Security* 6, pages 237–256, 1998.
- [13] P. Golle, S. Jarecki, and I. Mironov. Cryptographic primitives enforcing communication and storage complexity. In *Proceedings of Financial Cryptography*, volume LNCS 2357, pages 120–135. Springer-Verlag, Mar. 2002.
- [14] S. Haber and W. Stornetta. How to timestamp a digital document. In *Journal of Cryptology*, volume 3, pages 99–111, 1991.
- [15] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the twelfth international conference on World Wide Web*, pages 640–651. ACM Press, 2003.
- [16] R. S. L. Lamport and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

- [17] D. Malkhi, M. Merritt, and O. Rodeh. Secure reliable multicast protocols in a WAN. In *International Conference on Distributed Computing Systems*, pages 87–94, 1997.
- [18] U. M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.
- [19] T. Murphy and A. K. Manjhi. Anonymous identity and trust for peer-to-peer networks. Class Project, Carnegie Mellon, 2002.
- [20] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, February 2003.
- [21] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis and defenses. In *Proceedings of the third international symposium on Information processing in sensor networks (IPSN-04)*, pages 259–268, New York, Apr. 2004. ACM Press.
- [22] M. O. Rabin. Hyper encryption and everlasting secrets: A survey. In *CIAC: Italian Conference on Algorithms and Complexity*, 2003.
- [23] M. O. Rabin. Hyper-encryption and provably everlasting secrecy. Presentation at the Hebrew University, March 2004.
- [24] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, 2001.
- [25] A. Singh and L. Liu. Trustme: Anonymous management of trust relationships in decentralized p2p systems. In *Proceedings of the third IEEE Conference on Peer-to-Peer Computing*, 2003.
- [26] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

- [27] D. Verton. Analysts: Insiders may pose security threat. *Computerworld*, Oct. 15 2001.
- [28] D. S. Wallach. A survey of peer-to-peer security issues. In *Software Security – Theories and Systems, Next-NSF-JSPS International Symposium, ISSS 2002*, volume 2609 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2003.
- [29] M. W. Young, D. S. Thompson, and E. Jaffe. A modular architecture for distributed transaction processing. In *Proceedings of the Winter 1991 USENIX Conference*, pages 357–363. USENIX, Jan. 1991.