C Executive Summary

The Problem: Today's DoD missions operate over a large, heterogeneous, distributed set of computing resources—from personal mobile devices to massively parallel multicomputers managing millions of connections and petabytes of data. These distributed components must cooperate across agencies and across coalitions of allies; each partner brings independently-managed systems of varying reliability and trust into the distributed resource mix, and each has different policies and legal restrictions.

Today, we cannot reliably secure any single system against CyberAttacks, even when it is wholly owned by a single agency with a single mission. Computations can be disrupted (denial-of-service); machines can be co-opted (taken over and used by attacker); data can be corrupted and stolen. The problem is even further beyond the state-of-the-art when considering a coalition of machines under different jurisdictions. There is today no principled way to describe what such systems should be doing and thereby differentiate proper and compliant agents from rogue actors.

SOUND Solution: Our proposal, called *SOUND* (Safety On Untrusted Network Devices) provides a way to compute reliably on distributed coalition systems assembled from a wide range of heterogeneous components while ensuring desired restrictions on information flow (*confidentiality*), by trusted actors (*identification* and *authentication*), preventing information corruption (*integrity*), and maintaining high computational throughput (*availability*) despite the fact that the underlying set of computers and processes offered to perform the computation may be vulnerable to attacks or actively trying to compromise the mission.

We propose to achieve scalable and tunable innate distributed defense (BAA Task Area 1) by implementing shared situational awareness and trust modeling (first two elements of BAA Technical Area 2). Additionally we are proposing an option for Technical Area 6, Technology Demonstrator.

In the following, we list many of the challenges facing the implementation of secure resilient distributed systems today and how the SOUND system proposes to address those challenges. We use the format *Challenge:* description. **SOUND *response:*** response. We also reference numbered items (*e.g.* ①) in the abstract SOUND system represented in Figure 1.

Challenge: Differentially Reliable Elements—modern distributed systems involve a mix of secure and insecure computations, agents that are well behaved and nodes that have been compromised, and both highly reliable elements and unreliable ones. **SOUND *response:*** We provide patterns for factoring computations into agents with *separated privileges* running computations with *least privilege* on physically distinct machines that are *mutually suspicious* such that the misbehavior of any agent will be noticed by its peers and actions can be taken to neutralize the misbehaving agent.

Challenge: Policies and priorities vary by situation—e.g. agencies participating in a disaster relief operation agree to share relevant information during the operation. **SOUND *response:*** We form our machines into dynamically evolving *Communities of Trust* ① based on situational requirements and learned behavior of players. Connections between agents occur via *Introduction* ② such that connections are secret and respect policies and levels of trust.

Challenge: Trustworthiness and Reliability change over time. **SOUND *response:*** Systems within the community share information from their interactions (*e.g.* quality of service and compliance during cooperative tasks) to adapt quantitative trust assessments dynamically ③; well-behaved systems rise in stature in the community; buggy or rogue systems are neutralized.

Challenge: Confidentiality—In large distributed computations among mutually suspicious parties, information should be shared on a least privilege, authorized-to-know basis. Each agency or partner will share only an appropriate subset of their information in a controlled way with their partners. **SOUND *response:*** We use fine-grained data tagging, pervasive encryption, and distributed information flow management to assure that decrypted information flows only to authorized agents according to established policies relevant to the current operational situation.

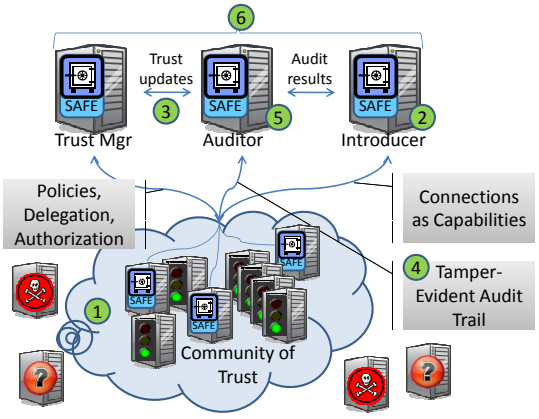


Figure 1: Abstract view of a SOUND System

Challenge: Accountability—If we agree that mutual suspicion is a good idea, upon what basis should trust be measured? **SOUND *response:*** We introduce the concept of *transparent accountability*. Every agent must declare its expected behavior to agents with which it communicates. This declaration is similar to a flight plan or zoning license declaring the kinds and volume of interactions the agent will have both within the community and outside it. Violations of this plan will lower the agent's level of trust within the community. Agents keep a *tamper-evident audit log* ④ that can be independently audited by a third party ⑤. Collaboration with audit and successful verification of audits is necessary to build high trust for the agent within the community.

Challenge: Where to run critical services—We need a "root of trust" in order to ground out the mutual suspicion between all elements and also to provide auditing, secure messaging, and maintain the information flow security requirements of all SOUND elements. **SOUND *response:*** We lower the attack surface area of servers by *specializing* them to dedicated tasks, and removing unnecessary functions and interfaces. We can build these servers on nodes from the SAFE project ⑥, where the core runtime services have been secured and formally verified. Furthermore, we can formally verify the services code written on top of the SAFE system.

Challenge: Formal verification of arbitrary, general-purpose languages remains hard. **SOUND *response:*** We employ specialized, domain-specific languages for these services that makes strong formal verification tractable.

Today we are at an inflection point: ubiquitous computer mediated automation is invaluable, but the cost of insecure computing is large and growing, creating ever greater and more unacceptable risks. Fortunately, we have more, cheap silicon capacity than ever and greater experience with and tools for peer-to-peer systems, systematic audit, cryptography, and formal methods. These technologies create new opportunities, but can only be fully exploited when embedded in the base infrastructure of our systems. Furthermore, our shift from centrally ordained, binary trust of systems to distributed and adaptive development of trust better matches our reality where nodes misbehave and their trustworthiness changes over time. This new perspective, coupled with these modern tools, gives us a chance to transform the way we approach the security of distributed systems.

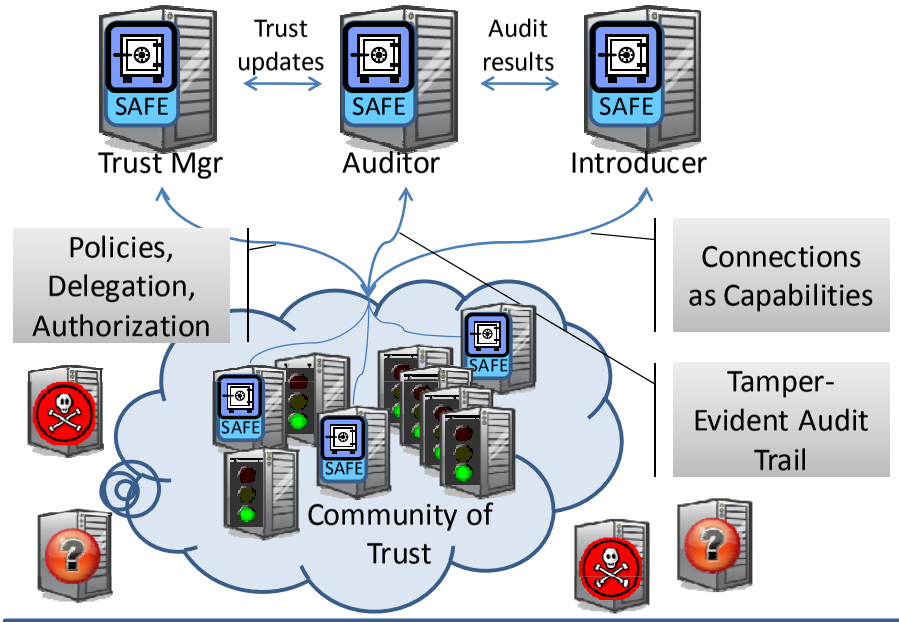
As outlined in Section N (and the Cost Volume), we are proposing a four year project.

D. Summary Chart

SOUND – Safety On Untrusted Network Devices

Objective

Achieve resilient distributed systems by enabling *Communities of Trust* based on mutual suspicion, transparent accountability, formal methods, and differentially more reliable systems based on CRASH SAFE.



Key Innovations

- **Communities of Trust** establish *Collective Immunity* built upon:
  - **Quantitative Dynamic Trust Management** - A distributed architecture for specifying and maintaining context-sensitive policies for delegation and authorization.
  - **Introduction-Based Routing** – a distributed mechanism for dynamically adapting trust based on feedback from performance and audit.
- **Transparent Accountability** for Dynamically Verifying Behavior.
- **Formal methods & languages** – for verified implementations of core components.
- **Extension & Use of SAFE technology** – from the CRASH project, is basis for system-wide *public health infrastructure*.

Impact: Revolutionary Increase in Security and Resilience for Distributed Mission-Oriented Computing

E Proposal Roadmap

Main goals of Safety On Untrusted Network Devices (SOUND):

1. Extend the clean-slate approach we are using to design the SAFE architecture on the DARPA CRASH program to heterogeneous networks of *differentially reliable* nodes. We apply the same concepts of *mutual suspicion* and *separation of privilege* (p.5), with the approaches of *decentralized information flow control* and *formal methods for security and correctness* (p.20).
2. Develop distributed, dynamic mechanisms for trust management based on *Dynamic Quantitative Trust Management* (p. 15), *Introduction-Based Routing* (IBR) (p. 14) and *Accountability* (p. 16).
3. Demonstrate effective use of highly secure hosts to bestow high security and resilience across a heterogeneous, differentially reliable distributed system. In particular, we propose to utilize and extend the SAFE architecture we are developing under the DARPA CRASH program (p. 18).
4. Develop and apply formal reasoning methods to prove critical system properties (p. 20).

Tangible benefits to end users: By investing in a modest amount of highly secure infrastructure (p.18), users gain an immediate increase in security and resiliency of their distributed systems.

Critical technical barriers: The CRASH program is already addressing a large class of vulnerabilities, namely those that arise on individual hosts. In a networked environment the space of potential vulnerabilities is dramatically larger due to the presence of differentially reliable nodes, the difficulty in establishing trust among distributed actors and of safe-guarding the privacy and integrity of information as it flows across potentially unreliable hosts (p.5,9)

Main elements of the proposed technical approach:

1. *Base core infrastructure on highly secure nodes* - general and special purpose SAFE processors combined with formally verified software form a solid foundation for the "public health" infrastructure of a heterogeneous distributed system of differentially reliable nodes. (p. 18)
2. *Accountability* - SOUND components produce evidence of trustworthiness and reliability. (p. 16)
3. *Communities of Trust based on IBR and Quantitative Trust Management* (p. 14, 15).
4. *Formal Methods* - prove core components correct and secure, and design domain specific languages for specifying policies and behavior. (p. 20)
5. *Demonstration Option* - We propose a compelling DoD-relevant demonstration as a basis for integration with other MRC performers (p.23).

Basis of confidence: Blaze has pioneered the field of Trust Management, along with Smith, a pioneer in trusted networks. Haeberlen is a leader in Accountability, Frazier co-invented Introduction-Based Routing, and Zdancewic is a leader in secure information flow. Tolmach is a leader in applying formal methods to software, and Rosenberg has published books on web security and cloud computing. DeHon is leading the SAFE architecture design, and all members have close working relationships with people on the CRASH SAFE team (pp.30,47-68)

Schedule and Deliverables: In year 1, we will focus on pair-wise integration of the core technologies (IBR, QuanTM, Accountability, DIFC). In year 2, we will focus on an integrated demonstration of SOUND, asserting the security of key elements. In year 3, we will add SAFE-based components and provably secure software, and in year 4 we will focus on a realistic integrated demonstration and proofs. For a detailed schedule and short task descriptions, (pp. 45-46). Deliverables are detailed in the Statement of Work (pp. 37-44).

F Goals and Impact

F.1 Goals

The primary goal of the SOUND (Safety on Untrusted Network Devices) project is to produce a scalable distributed computing infrastructure that will substantially increase the *reliability*, *resiliency*, *scalability*, *availability*, and *security* of DoD-relevant distributed systems.

For a modest (with respect to the size of the overall distributed system) investment in secure hardware and software, SOUND aims to deliver a huge improvement in overall security and resilience.

F.2 Innovations

In order to achieve this across-the-board increase in resiliency and security, we rely on the following key innovations:

1. *Communities of Trust:* Using a combination and extension of research results by team members in the areas of *Introduction-Based Routing* and *Dynamic Quantitative Trust Management*, we propose novel mechanisms for dynamically establishing and adapting trust levels among computational agents. This allows well intentioned agents to collaboratively identify and neutralize rogue agents.
2. *Accountability:* Extending our team's research in *Accountable Virtual Machines* and related areas, we propose novel mechanisms for supporting different levels of detailed auditing. We address issues of confidentiality using *Secure Distributed Information Flow Control*. The results of auditing reinforce the Communities of Trust referenced above.
3. *Pillars of Trust:* We propose to both extend and specialize the SAFE architecture we are developing under the DARPA CRASH program in order to create a trusted computing base infrastructure for SOUND distributed systems.
4. *Formal Specification and Model-Based Programming for Distributed Systems* We extend some of the formal methods for proving correctness and security that we are using on the SAFE project into the distributed computing domain. We will use formal methods to verify the implementation of core SOUND infrastructure. Furthermore, we propose novel Domain Specific Languages for high-level secure specification of distributed system behavior.

F.3 Comparison to Related Work

F.3.1 Accountability

Accountability in distributed systems has been suggested as a means to achieve practical security [Lam00], to create an incentive for cooperative behavior [DFM01], to foster innovation and competition in the Internet [LC06, AMIS07], and even as a general design goal for dependable networked systems [YC04, WABL+08]. Several systems are available to provide accountability for specific distributed applications, including network storage services [YC07], peer-to-peer content distribution networks [MSG07], and interdomain routing [ABF+08]; PeerReview [HKD07] and AVM [HARD10] are accountability systems that can be applied to arbitrary applications. However, existing systems have two major limitations that prevents their use in the context of SOUND. First, most existing systems cannot easily handle confidential data, since they assume that any node is allowed to know the data and expected functionality of any other node. The exception to this is NetReview [HARD09], which offers confidentiality of function (but not data) for the special case of interdomain routing. Second, existing systems cannot hold nodes accountable for leaking

information to a confederate, since they only consider behavior that is observable from a correct node [HK09]. The proposed research will remove both limitations.

F.3.2 Dynamic Trust Management

Trust management [BFL96] is an approach to authorization that uses credentials. These credentials can specify arbitrary actions and are validated with digital signatures. The signatures are checked to ensure that an action is authorized by an appropriate chain of authorizers (who have the authority to delegate actions). Several systems, notably PolicyMaker and KeyNote [BFK98], have been designed to validate this approach and it has been applied to contexts such as distributed firewalls and the Apache Web server. More recently, an approach called Dynamic Trust Management (DTM) [BKL+09] was suggested to extend trust management to dynamic environments, in which credentials might be interpreted in context, e.g., sensitive information might be made more or less accessible in a fire zone than back at headquarters. A further extension is Quantitative Trust Management (QTM) [WAC+09], which unifies credential-based authorization with another approach (confusingly also called "Trust Management") based on reputation scores. The advantage of QTM versus simple reputation management is that credentials can provide an initial basis for trust, specification of complex policies and accurately reflect organizational bases for authority (e.g., from CINC to SecDef to .... to Sergeant Rock). The advantage of QTM over simple Trust Management is that it provides a decentralized dynamic assessment of nodes reputations, to the point where credential revocation might be trivially accomplished with a reputation score of zero.

There is extensive work in reputation management; a good survey is that of Li and Singhai [LS07] and there are many systems (e.g., EigenTrust [KSGM03]) that have been devised. The integration of trust management and reputation management allows for a continuum of specified and learned information justifying an access decision.

There are two other approaches to integrated systems in addition to QTM. They differ in key respects. The work by Colombo, et al. [CMM+07] is focused on the GRID networked supercomputing paradigm; they use RT [LM03] as a role-based access control logic. They store reputation as a credential while we use a reputation database; we believe the database approach eases interfacing with other applications.

PACE [SET05] discusses a generalized model that, like QTM, is application-driven and supportive of decentralized applications. The paper presents the model, but neither an implementation nor any detailed use cases. The visibility constraints offered by their model are straightforward to incorporate into QuanTM and thereby into SOUND.

F.3.3 Formal Methods & Languages

Formal methods have been heavily applied to verification of abstract protocols, but much less to actual implementations of distributed systems. Sewell and colleagues have formalized existing TCP at both protocol and service layers [BFN+05, RNS08]. Kreitz [Kre04] formalized an implementation of the Ensemble network architecture and proved correctness of protocol stack optimizations. For programming the SOUND extensions to SAFE, we will draw on the many high-level language mechanisms for distributed programming developed in functional-language systems such as ERLANG [Arm03] and ACUTE [SLW+07], OO-inspired systems such as E [MTS05], and multi-paradigm systems such as Oz [VRHB+97]. Our general approach to creating a model-based DSL for monitoring and controlling SOUND is inspired by Williams et al. [RW06, WIC+04].

G Technical Plan

SOUND (Safety On Untrusted Network Devices) will provide a secure, resilient, scalable distributed computing platform upon which mission-oriented distributed applications can be built that are both resistant to and capable of adapting to an attack to continue a given mission.

In order to achieve this goal, we need two things:

1. *Communities of Trust* that provide an "immune system" for detecting and adapting to misbehaving elements of the distributed system, and
2. Highly reliable and trustable infrastructure nodes from which we can bootstrap a reliable "public health infrastructure" and provide oversight for the large number of mutually suspicious nodes in the network.

To satisfy these two requirements, we take two approaches:

1. For differentially more reliable network elements, we extend and utilize the SAFE architecture that we are developing under the DARPA CRASH program.
2. For detecting and adapting to misbehaving elements in a heterogeneous distributed system, our dynamic *Communities of Trust* are based on a novel integration of research in *Accountability*, *Quantitative and Dynamic Trust Management*, and *Introduction-Based Routing*.

In more detail, SOUND aims to achieve a breakthrough in security and resiliency of heterogeneous networked systems by innovating in the following four areas:

1. *Communities of Trust:* SOUND will implement an innovative combination of *Introduction Based Routing* with *Quantitative Trust Management*. This synergistic combination will support a self-organizing, dynamic web of trust, backed by context-sensitive policies. We will solidify distributed trust management by extending formal methods developed under the SAFE program to the network resulting in dramatically more trusted systems than is possible currently. The combination of these technologies will allow SOUND to implement unforgeable network resources, such as connections between distributed processes.

2. *Accountability and Monitoring with Mutual Suspicion:* We will extend work by Haeberlen [HARD10, Hae09] to enable tamper-evident logs and audit trails. Components will operate according to defined models of expected behavior and, according to the principle of mutual suspicion, will be dynamically audited for adherence to those models. The tension between accountability and privacy will be addressed by extensions to the Distributed Information Flow Control (DIFC) mechanisms developed under the SAFE project.

3. *Pillars of the Community: SAFE hosts as differentially reliable decentralized TCB:* The SOUND project proposes to extend the SAFE implementation from the CRASH program to allow SAFE hosts to operate in a heterogeneous distributed system of differentially reliable elements (like accredited public health system hospitals). SAFE-based elements will consist both of general purpose (but highly trusted) compute servers running critical computing tasks (e.g. auditing, introducing), and also of embedded devices (e.g. routers) obtained by creating *specializations* of the SAFE architecture. A distributed set of SAFE-based nodes will provide a Trusted Computing Base (TCB) for any MRC system.

4. *Formal Specification and Model-based Programming:* SOUND will extend the formal meth-

ods developed under SAFE to the networked SOUND setting. We will construct formal specifications for all services, protocols, and computational platforms in a SOUND system, and use these specifications both to verify the implementation of key components and to validate that the overall design delivers desired network-wide properties. To support programming of control and monitoring agents, SOUND will also include a high-level model-based declarative language that allows direct specification of desired system states and hides the underlying mechanisms.

G.1 Example - Massively Multiplayer Online Game

In order to explain our approach to creating secure, resilient distributed computing systems, we use the scenario of a massively multiplayer online game (MMOG). We hope that the reader can draw analogies between the MMOG example and any distributed system with which they might be more familiar – in short, any system with heterogeneous elements, potentially corrupted nodes, and the need for secure information flow and reliable asset tracking.

Figure 2 represents the computational agents involved: Players (some human players, others robots), World Region Servers (shared state and physics), Banks (tracking players assets), some networking elements represented as Routers; and some SOUND-specific infrastructure nodes such as Auditors, Introduction Servers, and Monitors.

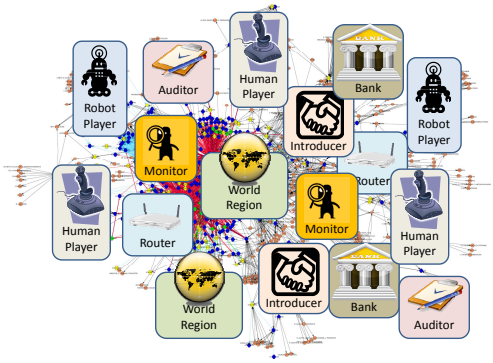


Figure 2: Massively Multiplayer Online Game

Agents vary in trustworthiness from Players (completely untrustworthy *a priori*) to Servers (legacy systems that are initially assumed trustworthy but are subject to compromise) to infrastructure nodes (highly trustworthy because they are newly implemented using SAFE technology, but still subject to fail-stop behavior). Following are some scenarios in our hypothetical MMOG.

G.1.1 Finding a Server

A Player wants to participate in a given *trustworthy* World Region. For this it needs to construct a secure and reliable communication channel with the corresponding Server, in which both Player and Server identities are assured. All host-to-host, or process-to-process, connections in SOUND are established via *Introduction-Based Routing* (IBR, see Section G.2.1), in which every process that can play the role of an *Introducer* maintains a trust vector that is constantly updated based on the activities that result from introductions made. Introducer processes execute on specialized nodes that advertise this service. Connections between elements – e.g. between a player and a server – are *unforgeable* such that no other agent in the system can "guess" the connection parameters in order to listen in or duplicate such network-level resources.

G.1.2 Accepting a Player

Players—a combination of platform, game software, and a human—are *a priori* unreliable because any of these building blocks may be misbehaving. Therefore, servers may have a policy requiring that the Player provides certification of its past trustworthy behavior. For specifying such policies, we build on work in *Quantitative and Dynamic Trust Management* (see Section G.2.2).

One basis for developing positive trust is based on auditing behavior. The server might demand that the Player submit to auditing or present evidence of past audits to show that it has behaved well so far. To support auditing, we build on work in *Accountable Virtual Machines* (See Section G.3). Auditing requires an agent to expose some of its data to other agents in order to participate. Confidence that the data will not be exfiltrated by any other agent comes from SOUND's use of Distributed Information Flow Control (DIFC) [Zda02, ZM02, LMZ03].

G.1.3 Compromise of Region Server

Analogous to Players, World Regions are a combination of platform, game software, and a local manager, and any of these may be be compromised. If a rogue Player is able to exploit such a Server to gain control, there are safeguards to maintain overall environment integrity. Monitor agents observe the overall system looking for anomalous behaviors; if a monitor detects a faulty Server, it: (i) identifies the rogue Player, kicks it off the system, and downgrades its reputation with the trust manager; (ii) shuts down and restarts the corrupt Server; and, (iii) notifies the innocent Players what has happened. SOUND provides a high-level declarative model-based DSL to specify such policies.

Core Research and Implementation Thrusts

The following sections expand on the four core research areas introduced above, (1) Communities of Trust, (2) Accountability and Monitoring with Mutual Suspicion, (3) SAFE hosts as differentially more reliable decentralized TCB, and (4) Formal Specification and Model-based Programming. Furthermore, we outline how the SOUND architecture will provide an ideal platform for integration with research in other MRC Task Areas such as Manageable Diversity, Planning, Repair, etc.

G.2 Communities of Trust

Our mission computations must be performed over a variety of hosts and networks built out of different computers, managed by different organizations (agencies, companies, countries) with different agendas and policies. These networks will include hostile agents that want to discover and undermine the mission; some of the hosts offered by the collaborating agencies will be compromised before or during the mission. How do we protect the integrity of the distributed computation in this heterogeneous, differentially reliable environment?

In the real world, individuals, companies, nations, and agencies come together to successfully complete missions in the face of similar challenges. We maintain a healthy suspicion, build trust between individuals and agencies, form communities, communicate expectations and standard of behavior within the community, and provide ways to detect and punish violators. In SOUND we bring these techniques to our dynamically formed mission-oriented communities of trust.

Specifically, SOUND proposes to integrate two leading edge dynamic trust management technologies—Introduction-Based Routing (IBR) and Quantitative and Dynamic Trust Mananagement (QuanTM)—to serve as the core protocols and infrastructure for community formation. QuanTM provides context-sensitive, adaptive policies, and IBR provides a distributed mechanism for maintaining and applying dynamic trust relationships in the context of emergent network (socio-economic) communities.

G.2.1 Introduction-Based Routing (IBR)

In the physical world, we have a collective response to misbehavior. When it was discovered that toothpaste exported from China contained diethylene glycol, public outrage pressured the U.S. government, which in turn pressured the Chinese government, which (then) enforced product safety laws with the exporters. The communal response forced action because the *supply chain* was visible. But in the Internet Protocol there is no "supply chain" for a packet.

The IBR protocol allows network communities to form, elicits socio-economic incentives for responsible networking, and enables a communal response to observed misbehavior. In much the same manner that the economic system forced Chinese toothpaste manufacturers to change, IBR prevents a node from repeatedly launching observable attacks.

Central to an economic system is choice; participants practice good behavior because their fellows will not interact with them otherwise. The simplest attack on the network today—denial-of-service (DoS)—shows how IP fails in this most basic tenet. An attacker simply overwhelms the victim with so much network traffic that it cannot access the network. Under IP, the target of the attack has no choice but to receive the packets. To prevent DoS attacks a node must only put packets into the network that the recipient is willing to receive: it requires a connection-oriented protocol.

The IBR Protocol Under the IBR protocol, a packet can enter the network if and only if it is traveling on a *connection* (comparable to a VPN between two nodes). Having entered the network, it can reach only the node at the other end of the connection. Both parties to a connection must consent to participate, and either party can *choose* to close the connection at any point. To establish a new connection, a node must be *introduced* by a third party with connections to both the node and the neighbor-to-be. Communication then proceeds only if an unforgeable token was provided during the introduction. Since no node will be connected to every other, forming a new connection may require multiple consecutive introductions. When a connection is closed, the two endpoints send binary feedback to the introducer indicating whether the other node was/is well-behaved. The introducer processes this feedback, and if the subject of the feedback was introduced to it, the feedback is forwarded to that introducer in turn. To bootstrap participation in the network, a node must have at least one *a priori* connection (a connection that does not require an introduction).

In the simple model, a node maintains a score for every neighbor to which it has a connection. We call that score a *reputation*. When a target is offered an introduction, it decides whether or not to establish the connection based on the reputation of the *introducer*—not the requester, for whom the target does not have a reputation model. Therein lies the economic incentive in IBR. Introducing a misbehaving node damages the introducer's subsequent ability to provide introductions driving nodes to exercise vigilance over those they are willing to introduce.

In IBR all members of the community are protected from a given type of misbehavior even if not all nodes are equipped to detect it. Analogous to vaccinations in a public health scenario, if 20% of the hosts in a network do not possess SPAM detectors, it will still be impossible[1] for an attacker to launch a SPAM attack. Like human population immune response, IBR naturally leverages the benefits of heterogeneity.

The defensive strength of IBR rests on the underlying economics of the system. The vast majority of the misbehavior on the Internet is high-volume, low-margin transactions. The bad guy launches

[1] This ceases to be true of the attacker can predict which hosts are not protected, of course.

many attacks, investing very little in each attack, still profiting from the low rate of success he gets. There are a number of well-known vectors that are used for these "commodity misbehaviors": spam, SQL injection, cross-site scripting, phishing, and port scanning. Detectors exist for these forms of misbehavior but there is little notion of communal defense. It is every-man-for-himself. If those detectors were integrated with IBR such that they reported misbehavior, these forms of misbehavior would be eliminated and all nodes would be protected. IBR isolates attacking hosts from the rest of the network as we showed in [FDWP11]. When a node ignores the "social norms" of the network and continues to introduce misbehaving hosts, it loses its ability to be an introducer.

G.2.2 Quantitative Trust Management (QuanTM)

A major challenge in determining trust in a decentralized setting is selecting robust *Trust Evidence*. How do we know who is making a request and if they are authorized to perform such a request? With what confidence can we draw these conclusions? Such evidence can be static, such as possession of cryptographic keys or a cryptographically signed certificate as in the original Trust Management proposal (implemented in KeyNote [BFK98]). However, as we deal with varying policies among different agencies assembled to contribute to a common mission, the evidence must be context-dependent, as in Dynamic Trust Management (DynTM) [BKL+09], with its sophisticated, context-sensitive authorization policies. When agents are subject to compromise and when we must continually make judgements about new agents in a system authorized by decentralized authorities, the evidence and judgements may need to be policy-, context- and behavior-dependent, as in Quantitative Trust Management (QuanTM) [WAC+09]).

Figure 3 represents the QuanTM architecture. The three boxes (demarcated with dashed lines) represent the Trust Management (TM), Reputation Management (RM) and Decision Management (DM) subsystems of QuanTM, moving from left to right in the figure.

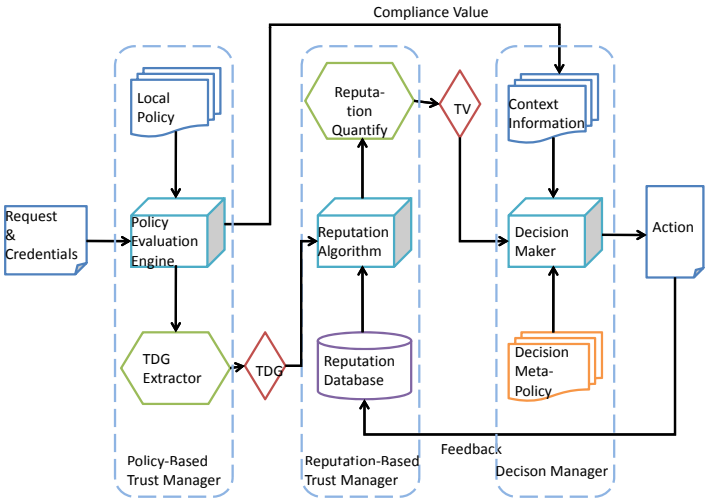


Figure 3: Architecture for Quantitative Trust Management

On the left side, a request and credentials authorizing that request are presented. An example credential is represented in Figure 4.

This credential represents that an authorizer `SecDef` authorizes two operations `query` and `update`, to three Licensees, for some database access. Compliance values are computed using the conditions values, with `True` meaning trust, `False` meaning don't trust, and `Maybe` meaning the system must consult additional policy rules to make a trust decision.

The TM subsystem parses the KeyNote language and computes a Compliance Value (CV) that is passed (over the RM subsystem in the figure) to the DM subsystem. The TM subsystem also constructs (using the dependencies inherent in delegated authorities) a trust dependency graph (TDG) annotated with compliance values computed using local policies. This TDG is what is passed to the RM subsystem.

```
Authorizer: SecDef
Licensees: SecARMY || SecNAVY || SecAIR
Conditions:
  operation == "query" -> "True";
  operation == "update" -> "Maybe";
Signature: "rsa-sig:1294..."
```

Figure 4: An example keynote credential

The RM subsystem consults observations stored in a Reputation Database that represent learned knowledge in QuanTM. A Reputation Algorithm is applied to the TDG object, computing trust values (TVs) using the KeyNote Compliance Values and the behavioral data from the Reputation Database; the Reputation Quantifier produces a final Trust Value (TV), which is passed to the DM subsystem. In the QuanTM paper, the Reputation Database was assumed to be pre-populated; in our proposed SOUND effort, an initial database and feedback mechanism derived from IBR and audit (Section G.3) will flow into the system at the arc on the lower right, feeding into the Reputation Database.

In the DM subsystem, the CV is combined with the context information (e.g., indications of a high alert) before being passed to the Decision Maker. This injection of context is where the Dynamic Trust Management is achieved; the Decision Maker combines the information from the TM subsystem and the Reputation Management subsystem using a Decision Meta-Policy (example in Section G.5.2) and determines an Action, shown emerging from the right hand side of the figure. The Decision Meta-Policy is application-dependent, but might represent the application's preferences for trust evidence and how these are affected by context and observed behaviors. Feedback on the action is incorporated into the Reputation Database.

Chang, et al. (from UPenn) [CVW+11] have demonstrated the application of a behavior-based reputation system to the Border Gateway Protocol (BGP), a highly decentralized information exchange for Internet path information. The derivation of probabilistic measures of trust for Autonomous Systems (ASs) on the Internet is shown to be useful to increase routing performance and dampen the effect of misbehaving nodes.

We propose to integrate IBR with an enhanced QuanTM such that the "Request" is for introductions, and the Reputation Database is gradually populated with information about Clients and Introducers generated both from IBR interactions and from audit. Our initial integration goal for SOUND, has a default notion of reputation management, but SOUND will be open to integration with other reputation algorithms, such as [JHP06].

G.3 Accountability and Monitoring with Mutual Suspicion

If we are going to build a *Community of Trust* out of a population that includes malicious and compromised agents, how do we establish trust? If agents trustworthiness can change over time (*e.g.* they are compromised, or a sleeper agent awakens), how do we detect misbehavior and reassess trust? A key part of community building and maintenance in SOUND is a framework for transparent accountability based on declared intentions and systematic audit.

Every element of a SOUND system produces tamper-proof evidence sufficient for users of the element to check whether the element is performing satisfactorily. Computational elements range from individual instructions on a host, to functions, to processes, to ensembles of processes on a single host, to a host as a black box, to distributed ensembles. The proposed research will build on research on *accountability* by Haeberlen et al. [HKD07, HARD09, HK09, BDHU09, HARD10, Hae09].

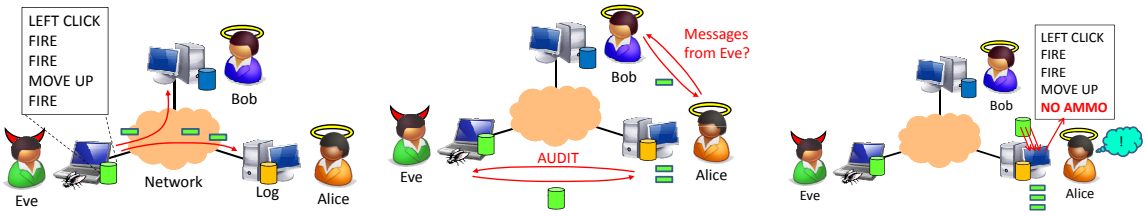


Figure 5: Simplified example: Accountability in a multiplayer game

In an accountable system, each node is responsible for performing some specific function, such as storing files or forwarding packets. This function is called the node's *expected behavior*. However, it is assumed that some nodes may deviate from their expected behavior, e.g., because they are faulty or have been compromised. The goal of accountability is a) to detect when such misbehavior occurs; b) to identify misbehaving nodes; and c) to produce evidence that irrefutably links the misbehavior to a specific node.

The SOUND system can use this evidence to update its trust relations: A long period of correct behavior can increase trust, while evidence of misbehavior decreases it. However, to avoid a cyclic dependency, SOUND cannot rely on the trust relationships to produce evidence. Instead, we conservatively assume that an *unknown, arbitrary subset* of the nodes (whether trusted or not) is misbehaving, and that the type of misbehavior itself is not known – a compromised node can *arbitrarily deviate* from its protocol, send information to untrusted nodes, or even 'cry wolf' and implicate a node that is behaving correctly. Since there is no node that is fully and universally trusted, we must rely on a system of 'checks and balances'.

Existing accountability techniques [HKD07, HARD10] generate evidence as follows. First, each node is required to maintain a *tamper-evident log* of its local actions, such as sending or receiving messages, certain processing steps, etc. The logs of different nodes are intertwined such that, if a node tampers with its local log or maintains an incomplete log, at least one other node is guaranteed to detect this. Second, each node is associated with a set of *witnesses* (auditors) that periodically audit that node's log. Thus, as long as each node has at least one correct witness, misbehaving nodes cannot escape detection: If they misbehave but maintain a correct log, the witnesses will discover the misbehavior; if they attempt to cover their traces, this will be discovered.

Figure 5 illustrates this approach in the context of a multiplayer game. Each of the three players maintains a tamper-evident log of his or her actions; Eve has modified her game software to gain unlimited ammunition (left). When Alice becomes suspicious of Eve's good performance, she can audit her log and check it for tampering (middle); she then replays the log using her own copy of the game software to check for cheating (right). Since Alice's copy of the game software has not been modified, the behavior during replay will be different. Alice now knows that Eve must have been cheating, and she can use the copy of Eve's log to convince Bob.

However, the existing techniques typically assume that each node knows the expected behavior of all other nodes, and that each node is authorized to see the log of any other node. This leads to the following additional challenges in the context of SOUND:

- **Partial specifications:** Witnesses and other nodes may not know the entire expected behavior of a given node (e.g., because some of it is secret), but only some aspects of it.

- **Confidential data:** Even if a node knows another node's expected behavior, it is not necessarily authorized to see the specific data that the node is operating on.
- **Information flow:** Nodes should be accountable not only for what they do, but also for what they *do not* do – such as failing to keep a secret from another node.

Even if no individual witness knows a given node's entire expected behavior, we can still hold that node accountable for its behavior as long as all the witnesses *collectively* know the expected behavior. However, existing accountability techniques [HKD07, HARD10] detect misbehavior by deterministically replaying a node's log using a reference implementation, which only works if the entire behavior is known. We plan to develop a new accountability technique for SOUND that can verify an abstraction (i.e., a more coarse-grained specification) of the expected behavior. One way to accomplish this would be to use higher-level declarations of intent, e.g., a description of the node's expected communication patterns. These would effectively constitute a 'flight plan' that can then be checked at runtime. We have already solved this problem for a special case in [HARD09].

To protect the actual data that a node is processing, we can use at least two approaches. The first approach is to encrypt any confidential data that the witnesses are not allowed to see. For instance, if the game is routing chat messages between players, we can encrypt the text of the messages but leave the headers intact, so that the witnesses can still verify that the messages were routed correctly. The other approach is to choose the witnesses such that a) each confidential item is accessible by at least one witness, and b) auditing can be compartmentalized. For instance, each region server can be audited by the players that are currently present in its region, since the corresponding state is visible to these players anyway. In some cases, the witnesses may have to cooperate: for instance, if several players cast secret spells and the game server must choose and reveal the strongest spell, we can use ring signatures to verify that the chosen spell was actually cast (but not by whom), and each player can individually check whether his own spell was weaker than the chosen spell (without learning the others' spells). Cryptographic commitment schemes can be used to ensure that players or servers cannot change their spells after the fact, or deny having cast a spell.

Existing accountability techniques cannot easily protect information flow because there is no way to detect when a compromised node communicates outside of the system, e.g., by using another network interface to send secret data to a confederate. However, even with minimal support from SAFE, we should be able to do better. For example, we can use trusted hardware to ensure that a SOUND node cannot send messages without logging them first, and that the messages cannot be decrypted before the recipient has logged their arrival. We plan to extend existing accountability techniques to take advantage of such features.

G.4 Pillars of the Community: SAFE hosts as differentially reliable decentralized TCB

If no one trusts anyone else, community building is very difficult. In particular, what happens if we cannot trust the key community building services like Introducers, Auditors, and Trust Managers? Certainly, we can distribute these functions and build consensus, and we should do that. However, since these services are critical to healthy communities, we consider how we can provide greater assurance that they are trustable—that they serve as true Pillars of the Community. The key is that it only takes a few of these highly reliable and trustable services to supervise a large community, and these highly trustable servers only need to perform a few specific functions.

We can leverage hosts developed in the SAFE project to serve as these highly trustworthy servers. Since the scope of the CRASH project under which SAFE is being developed is limited to the host

itself, here we expand the SAFE hosts to interact with distributed systems including:

- developing formally verified, secure networking code
- developing and integrating formally verified IBR support into the network stack (Section G.2.1).
- adding network-aware trust management (Section G.2.2)
- using cryptographic signatures and hashes with the network-aware trust management to safely extend fined-grained tagged information flow across a distributed collection of SAFE systems working with the consistent trust managers. Keys from the trust manager are used to establish identity and maintain confidentiality between machines.
- developing formally verified, tamper-evident audit log for SAFE based on [HKD07] (Section G.3)

A key feature of SAFE design is that it assumes considerable internal mutual suspicion and performs self checking. As such, a SAFE node is more likely to detect, isolate, and report a misbehaving agent. Even if an agent subverts a core SAFE component, internal self checking means the most likely result is that the SAFE node will shut down rather than perform any corrupted operations. As such, should a SAFE node become non-responsive, that should throw suspicion both on the SAFE node and the agents it was interacting with immediately prior to its last response.

Differential reliability does not imply that the more trustworthy nodes are otherwise generic computing platforms, where if a vulnerability does appear (say in an application) then it affects all nodes. Rather, we envision a SOUND configuration where the highly reliable nodes carry out distinct, specialized functions. For example, several functions we have discussed such as auditing and authority checking can be carried out by single-purpose network elements. The specialization of purpose leads quite naturally to a decentralized, networked analogue of separation of privilege with dedicated nodes carrying out one or at most a few functions. Thus, even in the unlikely case where the enhanced reliability nodes are vulnerable in some way, the physical and logical isolation between functional elements provides an additional aspect of prevention in depth.

To deploy SAFE hosts to play these specific and dedicated roles in the infrastructure, we will develop tools for creating *specialized* SAFE servers that only run the minimum software and services necessary to support their server task. This reduces the surface area for attack by removing unnecessary interfaces and functionality. For example, paths for code modification, principal creation, authority change, and rule insertion may no longer be needed and can be removed completely. These specialized servers will be more amenable to audit. Candidates for specialized SAFE-based components include:

- **IBR-aware Router** – this supports IBR-as-capabilities throughout the network. The router will not transport packets between machines unless an introduction has been made.
- **Introduction Server** – collects pedigree information on network components and agents, and provides introduction to trustworthy service providers.
- **Trust Manager** – a highly trustworthy KeyNote server
- **Auditor** – performs audit, protecting sensitive information that may be necessary to review as part of audit.
- **Persistent Store** – securely declared intentions (models) for network players and holds information necessary for audits and recovery.

G.5 Formal Specification and Model-based Programming

Design, implementation, and management of SOUND networks will require explicit attention to trust management, auditability, and information flow, on top of standard concerns such as reliability and performance. These added dimensions lead to a huge space of design alternatives and operating characteristics. To cope with the complexity of design, we will make pervasive use of formal methods to establish key properties, both at the level of component interfaces and implementations, and system-wide. To address the complexity of programming the system, we will define declarative Domain-Specific Languages (DSLs) that encapsulate monitoring and control algorithms.

This commitment to formal methods is practical because, as noted in the previous section, a few highly reliable infrastructure services can supervise and provide assurance for the entire community. These servers are a high point of leverage for applying formal methods—there are just a small number of them performing very limited tasks, but if we secure those tasks, they can provide assurance for a very large number of hosts running a huge range of applications.

G.5.1 Specification and Verification

We will give formal specifications (e.g., in the Coq Theorem Prover) for all services, protocols, and computational platforms. Specifications will use operational semantics to describe intended behaviors. Such specifications: (i) provide clear and unambiguous documentation of the interface between service provider and client; (ii) embody a high-level notation for refining design and testing the interactions among service layers; (iii) can be used to test or formally verify correctness of an implementation; (iv) can be used to test or formally verify properties of a client. Formal specifications are widely used for abstract protocols, and they have been applied *post hoc* to existing network protocol implementations [BFN+05, RNS08]. But pervasive use of specification throughout the protocol design and implementation process will be novel.

Our primary focus for verification of low-level implementations will be network-level infrastructure software, such as basic channel communication primitives and the OS and network device drivers that support them. Following recent work on verifying runtime system services by Tolmach and others [MCT10, YH10], we will embed network operations as primitives into a low-level programming language; the operational semantics of the language specify the behavior of these operations, thus providing a formal basis both for mechanical verification of implementations of these operations, and for reasoning about programs that use them.

At a system-wide level, we will focus on proving whole-network properties such as security, trust maintenance, and reliable delivery under appropriate assumptions. These proofs may rely on the assumption that SOUND routers are implemented using SAFE technology, and therefore enjoy fail-stop behavior (i.e., their only failure mode is to halt), though proofs of system-wide properties will show progress and adaptation.

G.5.2 Model-based Programming

We will design and implement Domain-Specific Languages (DSLs) for maintaining "public health" of distributed computations. These DSLs will be inspired by the control languages used in the model-based programming paradigm [RW06], but specialized to the description of distributed computations and the use of trust-based decision making.

To illustrate the intended scope and power of these DSLs, we give an example for a Decision

Manager (DM) node in the QTM architecture (Figure 3), instantiated for a handling new MMOG Player Introduction requests. We suppose that the DM has the goal of balancing load among Servers as well as limiting participation by untrustworthy Players; thus, the Context information for the DM includes current Player-Server connections and the estimated current load on each Server. Our DSL is used to specify the Decision meta-policy used by the DM to approve or deny Introduction requests.

For this example, we use the architecture and modeling approach described by Williams et al. [WIC+04]. (In practice, we will experiment with a variety of model formalisms and corresponding model-based program evaluation techniques.) In this approach, we provide a *stateful model* of the system and a *control language* that specifies how to configure the system via imperative updates to the model state, guarded by tests on that state. The implementation of the control language maps reads and writes of model state to the underlying sensing and action primitives of the real computing system.

In this case, the stateful model includes connection information maintained in the DM Context and Trust information from the Trust Manager. Let $s \in S$ be Servers and $p \in P$ be Players. Let T be a trust value and L be a load, both represented by real numbers in $[0, 1]$. The modeled state of the system consists of

- a relation $Session \subset P \times S$ associating Players to Servers
- a function $Trust : S \cup P \to T$ mapping agents to current trust levels
- a function $Load : S \to L$ mapping Servers to current load levels

The underlying actions in the real system are to approve or deny a Player's request for a new Introduction to a particular Server. Underlying sensor inputs include things like: new connections are reported by a Server, or the reputation of an entity changes in the Trust Manager. These actions and inputs are reflected indirectly by changes in the modeled state. The control language is used to program the *policy* being enforced by this DM, in terms of the modeled state. Based on the current (unprimed) values of the state, it describes the desired future (primed) values. For example, the following fragment might describe how to react to changes in trustworthiness of different agents:

$\forall p, \texttt{when}(\textit{Trust}(p) < 0.5)\{\, \nexists s, (p, s) \in \textit{Session'}\}$
$\forall s_0, \texttt{when}(\textit{Trust}(s_0) < 0.75)\{$
$\quad \nexists p, (p, s_0) \in \textit{Session'} \wedge$
$\quad \forall p, (p, s_0) \in \textit{Session} \Rightarrow \exists s, (p, s) \in \textit{Session'} \wedge$
$\quad \forall s \neq s_0, \textit{Load'}(s) \leq 1.1 \times (\textit{Load}(s) + \textit{Load}(s_0)/\mid s \mid)\}$

This program states that if a Player's trust score drops below 50%, it should eventually be removed from the game (no longer appear in any session). On the other hand, if a Server's trust score drops below 75%, the game state should eventually change so that the Server no longer appears in any session, but the Players it was connected to have been connected to new Servers, distributing the additional load evenly over all remaining servers.

The intent expressed by this program can be achieved over time by approving or denying Introduction requests appropriately. The important point is that this code doesn't have to give details of *how* to change the connection graph; it just says "it shall be so!" and leaves the details to the control language implementation. Similarly, the details of how trust and load levels are maintained

are hidden. Of course, this example represents just one possible division between model-visible concepts and underlying functionality hidden in the substrate. We will explore a range of possible architectures, possibly resulting in multiple DSLs.

G.6 SOUND as an Open Experimental Platform for Mission-Oriented Resilient Computing

Just as SAFE aims to develop a platform with best-possible "innate" immunity and an architecture amenable to incorporation of "adaptive" immunity mechanisms, SOUND has similar goals for distributed systems. That is, SOUND aims to provide a mission-aware distributed computing core platform that supports both the incorporation of differently-trusted systems, but also a wide variety of adaptive network technologies.

We intend SOUND to be a core distributed computing platform for the MRC program. That is, SOUND should be amenable to the addition of all sorts of dynamic, adaptive monitoring, diagnosis, and planning technology. SOUND provides fundamental primitives—networking, audit and trust management—that are configurable by application-specific and enterprise-specific policies. One can imagine, for example, an application that adapts its behavior to its assessment of trust elsewhere in a SOUND system by demanding additional credentials or even contacting multiple copies of a specialized service to vote their values in an attempt to gain additional protection from diversity, Many more applications suitable to both non-DoD and DoD environments will emerge using the SOUND core platform for distributed computing.

As an exemplar application we propose a pathfinder application that exercises SOUND platform capabilities and enhances an existing application already used in the Intelligence Community (See Section H, below).

G.7 Spiral Approach to Development and Integration

As laid out in Section L (Schedule and Milestones), we plan to have multiple integration and demonstration points for each year of the proposed four year project. We will start with simple network applications such as "chat" and integrate IBR and Accountability. We will also work on integrating IBR and QuanTM over the first two years, with planned spirals ending with "spikes" (integrated demonstrations). In parallel, we will work on specifying provable secure network components and specializing the SAFE architecture for SOUND components. We will also work on formalizing the combined protocols and derived policy languages, with a goal of proving secure information flow and progress properties. After 18 months, we plan to be able to demonstrate integration of IBR, QuanTM and Accountability running on a heterogeneous system including some (simulated) SAFE components as well as various conventional, less trusted components.

To assess our proposed technology efforts we offer to create a SOUND integration demonstration scenario (costed as Task 3) that illustrates a DOD relevant collaboration use-case influenced by the Polestar domain (described in Section H) and current concerns around the spate of "Wikileaks" like attacks. We will create a networked actor collaboration scenario in which a set of intelligence analysts and field sources interact. We will model good actors, bad actors, and "nosy" actors. SOUND will shut down the bad actors as they attempt to perform unauthorized/undeclared activities. Our monitoring will detect that the "nosy" actors are performing actions they are entitled to, but that eventually the volume and range of their data access is suspicious and will lower their trust and eventually quarantine their network activities.

P Appendix B: Bibliography

References

[ABF+08] David Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable Internet protocol (AIP). In *Proceedings of the ACM SIGCOMM Conference*, pages 339–350, August 2008. `http://www.cs.cmu.edu/~dga/papers/aip-sigcomm2008.pdf`.

[AMIS07] Katerina Argyraki, Petros Maniatis, Olga Irzak, and Scott Shenker. Loss and delay accountability for the Internet. In *Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP'07)*, October 2007. `http://infoscience.epfl.ch/record/111741/files/AudItIcnp07.pdf?version=5`.

[Arm03] Joe Armstrong. *Making reliable distributed systems in the presence of errors*. PhD thesis, Royal Institute of Technology, 2003.

[BDHU09] Michael Backes, Peter Druschel, Andreas Haeberlen, and Dominique Unruh. CSAR: a practical and provable technique to make randomized systems accountable. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS'09)*, Feb 2009.

[BFK98] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1998.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society, 1996.

[BFN+05] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Rigorous specification and conformance testing techniques for network protocols, as applied to tcp, udp, and sockets. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 265–276, New York, NY, USA, 2005. ACM.

[BKL+09] Matt Blaze, Sampath Kannan, Insup Lee, Oleg Sokolsky, Jonathan M. Smith, Angelos D. Keromytis, and Wenke Lee. Dynamic trust management. *IEEE Computer*, 42(2):44–52, 2009.

[CMM+07] Maurizio Colombo, Fabio Martinelli, Paolo Mori, Marinella Petrocchi, and Anna Vaccarelli. Fine grained access control with trust and reputation management for globus. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2)*, volume 4804 of *Lecture Notes in Computer Science*, pages 1505–1515. Springer, 2007.

[CVW+11] Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West, Sampath Kannan, Boon Thau Loo, Oleg Sokolsky, and Insup Lee. As-trust: A trust quantification

scheme for autonomous systems in bgp. In McCune et al. [MBP+11], pages 262–276.

[DFM01] Roger Dingledine, Michael J. Freedman, and David Molnar. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Accountability. O'Reilly and Associates, 2001.

[FDWP11] Gregory Frazier, Quang Duong, Michael P. Wellman, and Edward Petersen. Incentivizing responsible networking via introduction-based routing. In McCune et al. [MBP+11], pages 277–293.

[Hae09] Andreas Haeberlen. A case for the accountable cloud. In *Proceedings of the 3rd ACM SIGOPS International Workshop on Large-Scale Distributed Systems and Middleware (LADIS'09)*, October 2009.

[HARD09] Andreas Haeberlen, Ioannis Avramopoulos, Jennifer Rexford, and Peter Druschel. NetReview: Detecting when interdomain routing goes wrong. In *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI'09)*, Apr 2009.

[HARD10] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*, October 2010.

[HJLT05] Thomas Hallgren, Mark P. Jones, Rebekah Leslie, and Andrew Tolmach. A principled approach to operating system construction in haskell. In *Proceedings of the tenth ACM SIGPLAN international conference on Functional programming*, ICFP '05, pages 116–128, New York, NY, USA, 2005. ACM.

[HK09] Andreas Haeberlen and Petr Kuznetsov. The Fault Detection Problem. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS'09)*, December 2009.

[HKD07] Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Oct 2007.

[JHP06] Audun Jøsang, Ross Hayward, and Simon Pope. Trust network analysis with subjective logic. In Vladimir Estivill-Castro and Gillian Dobbie, editors, *ACSC*, volume 48 of *CRPIT*, pages 85–94. Australian Computer Society, 2006.

[Kre04] Christoph Kreitz. Building reliable, high-performance networks with the nuprl proof development system. *J. Funct. Program.*, 14:21–68, January 2004.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW*, pages 640–651, 2003.

[Lam00] Butler W. Lampson. Computer security in the real world. In *Proc. Annual Computer Security Applications Conference*, December 2000. http://research.microsoft.com/en-us/um/people/blampson/64-SecurityInRealWorld/Acrobat.pdf.

[LC06] Paul Laskowski and John Chuang. Network monitors and contracting systems: Competition and innovation. In *Proc. ACM SIGCOMM*, pages 183–194, September 2006. http://doi.acm.org/10.1145/1159913.1159935.

[LM03] Ninghui Li and John C. Mitchell. A role-based trust-management framework. In *DISCEX (1)*, pages 201–. IEEE Computer Society, 2003.

[LMZ03] Peng Li, Yun Mao, and Steve Zdancewic. Information integrity policies. In *Proceedings of the Workshop on Formal Aspects in Security and Trust (FAST)*, September 2003.

[LS07] Huaizhi Li and Mukesh Singhal. Trust management in distributed systems. *IEEE Computer*, 40(2):45–53, 2007.

[MBP+11] Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres, editors. *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*, volume 6740 of *Lecture Notes in Computer Science*. Springer, 2011.

[MCT10] Andrew McCreight, Tim Chevalier, and Andrew Tolmach. A certified framework for compiling and executing garbage-collected languages. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, ICFP '10, pages 273–284, New York, NY, USA, 2010. ACM.

[MSG07] Nikolaos Michalakis, Robert Soulé, and Robert Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Proc. NSDI*, April 2007. http://www.usenix.org/event/nsdi07/tech/full_papers/michalakis/michalakis.pdf.

[MTS05] Mark S. Miller, E. Dean Tribble, and Jonathan Shapiro. Concurrency among strangers: programming in e as plan coordination. In *Proceedings of the 1st international conference on Trustworthy global computing*, TGC'05, pages 195–229, Berlin, Heidelberg, 2005. Springer-Verlag.

[RNS08] Tom Ridge, Michael Norrish, and Peter Sewell. A rigorous approach to networking: Tcp, from implementation to protocol to service. In *Proceedings of the 15th international symposium on Formal Methods*, FM '08, pages 294–309, Berlin, Heidelberg, 2008. Springer-Verlag.

[RW06] Paul Robertson and Brian C. Williams. Automatic recovery from software failure. *Commun. ACM*, 49(3):41–47, 2006.

[SET05] Girish Suryanarayana, Justin R. Erenkrantz, and Richard N. Taylor. An architectural approach for decentralized trust management. *IEEE Internet Computing*, 9(6):16–23, 2005.

[SLW+07] Peter Sewell, James J. Leifer, Keith Wansbrough, Francesco Zappa Nardelli, Mair Allen-Williams, Pierre Habouzit, and Viktor Vafeiadis. Acute: High-level programming language design for distributed computation. *J. Funct. Program.*, 17(4-5):547–612, 2007.

[VRHB+97] Peter Van Roy, Seif Haridi, Per Brand, Gert Smolka, Michael Mehl, and Ralf Scheidhauer. Mobile objects in distributed oz. *ACM Trans. Program. Lang. Syst.*, 19:804–851, September 1997.

[WABL+08] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Communications of the ACM*, 51(6):82–88, 2008. `http://doi.acm.org/10.1145/1349026.1349043`.

[WAC+09] Andrew G. West, Adam J. Aviv, Jian Chang, Vinayak S. Prabhu, Matt Blaze, Sampath Kannan, Insup Lee, Jonathan M. Smith, and Oleg Sokolsky. Quantm: a quantitative trust management system. In Evangelos P. Markatos and Manuel Costa, editors, *EUROSEC*, pages 28–35. ACM, 2009.

[WIC+04] Brian C. Williams, Michel D. Ingham, Seung Chung, Paul Elliott, Michael Hofbaur, and Gregory T. Sullivan. Model-based programming of fault-aware systems. *AI Mag.*, 24:61–75, January 2004.

[YC04] Aydan R. Yumerefendi and Jeffrey S. Chase. Trust but verify: Accountability for internet services. In *ACM SIGOPS European Workshop*, Sep 2004. `http://doi.acm.org/10.1145/1133572.1133585`.

[YC07] Aydan R. Yumerefendi and Jeffrey S. Chase. Strong accountability for network storage. *ACM Transactions on Storage*, 3(3):11, 2007. `http://doi.acm.org/10.1145/1288783.1288786`.

[YH10] Jean Yang and Chris Hawblitzel. Safe to the last instruction: automated verification of a type-safe operating system. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '10, pages 99–110, New York, NY, USA, 2010. ACM.

[Zda02] Stephan A. Zdancewic. *Programming Languages for Information Security*. PhD thesis, Cornell University, August 2002.

[ZM02] Steve Zdancewic and Andrew C. Myers. Secure information flow via linear continuations. *Higher Order and Symbolic Computation*, 15:2002, 2002.

Q Appendix C: Links to Relevant Material

Q.1 Preliminary Design of the SAFE Platform

Safe is a clean-slate design for a secure host architecture, coupling advances in programming languages, operating systems, and hardware, and incorporating formal methods at every step. The project is still at an early stage, but we have identified a set of fundamental architectural choices that we believe will work together to yield a high-assurance system. We sketch the current state of the design and discuss several of these choices.

Benoit Montagu, Gregory Malecha, Andre DeHon, Ben Karel, Robin Morisset, Benjamin Pierce, Greg Morrisett, Jonathan Smith, Olin Shivers, Randy Pollack, Jr. Thomas F. Knight, Sumit Ray, and Gregory Sullivan. Preliminary design of the safe platform. Submitted to Programming Languages and Operating Systems 2011, 2011.

`http://www.crash-safe.org/sites/default/files/plos11-submission.pdf`

Q.2 Accountable Virtual Machines

In this paper, we introduce accountable virtual machines (AVMs). Like ordinary virtual machines, AVMs can execute binary software images in a virtualized copy of a computer system; in addition, they can record non-repudiable information that allows auditors to subsequently check whether the software behaved as intended. AVMs provide strong accountability, which is important, for instance, in distributed systems where different hosts and organizations do not necessarily trust each other, or where software is hosted on third-party operated platforms. AVMs can provide accountability for unmodified binary images and do not require trusted hardware. To demonstrate that AVMs are practical, we have designed and implemented a prototype AVM monitor based on VMwareWorkstation, and used it to detect several existing cheats in Counterstrike, a popular online multi-player game.

Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10), October 2010.

`http://www.cis.upenn.edu/~ahae/papers/avm-osdi2010.pdf`

Q.3 QuanTM: a quantitative trust management system

Quantitative Trust Management (QTM) provides a dynamic interpretation of authorization policies for access control decisions based on upon evolving reputations of the entities involved. QuanTM, a QTM system, selectively combines elements from trust management and reputation management to create a novel method for policy evaluation. Trust management, while effective in managing access with delegated credentials (as in PolicyMaker and KeyNote), needs greater flexibility in handling situations of partial trust. Reputation management provides a means to quantify trust, but lacks delegation and policy enforcement.

This paper reports on QuanTM's design decisions and novel policy evaluation procedure. A representation of quantified trust relationships, the trust dependency graph, and a sample QuanTM application specific to the KeyNote trust management language, are also proposed.

Andrew G. West, Adam J. Aviv, Jian Chang, Vinayak S. Prabhu, Matt Blaze, Sampath Kannan, Insup Lee, Jonathan M. Smith, and Oleg Sokolsky. 2009. QuanTM: a quantitative trust management

system. In Proceedings of the Second European Workshop on System Security (EUROSEC '09). ACM, New York, NY, USA, 28-35. DOI=10.1145/1519144.1519149

http://doi.acm.org/10.1145/1519144.1519149

Q.4 Incentivizing Responsible Networking via Introduction-Based Routing

The Introduction-Based Routing Protocol (IBRP) leverages implicit trust relationships and per-node discretion to create incentives to avoid associating with misbehaving network participants. Nodes exercise discretion through their policies for offering or accepting introductions. We empirically demonstrate the robustness of IBRP against different attack scenarios. We also use empirical game-theoretic tech- niques to assess the strategic stability of compliant policies, and find preliminary evidence that IBRP encourages the adoption of policies that limit damage from misbehaving nodes. We argue that IBRP scales to Internet-sized networks, and can be deployed as an overlay on the current Internet, requiring no modifications to applications, operating systems or core network services, thus minimizing cost of adoption.

Gregory Frazier, Quang Duong, Michael P. Wellman, and Edward Petersen. Incentivizing responsible networking via introduction-based routing. In McCune et al. [MBP+11], pages 277-293.

http://dx.doi.org/10.1007/978-3-642-21599-5_21