Benefits of Deploying Inherently Secure Nodes Within a Distributed System

Joseph Fahey, Howard Reubenstein, David Wittenberg
BAE Systems
Burlington, MA
David.Wittenberg@BAESystems.com

Gregory Sullivan
Charles Stark Draper Laboratory
Cambridge, MA

***Abstract*— We present simulation results that show that placing security-related intrusion monitors on SAFE processors (as opposed to on vulnerable conventional systems) dramatically increases the overall health of a distributed system. We model a distributed system protected by a combination of network- and host-based detection and remediation infrastructure. The model is parameterized according to the accuracy of attack detection, the likelihood of attack success, and the ratio of security-related resources to general computation resources.**

Keywords— security, network, gateway, resilience

I. Introduction

Current computers are inherently insecure, causing cyberattacks to be a major concern to both commercial and defense organizations. Computing hosts still use an architecture that was designed when 8 KBytes was a lot of memory, and internet connectivity was over a decade away. The environment computers are used in has changed, but our computer architectures have not changed. With the underlying hosts having minimal security, the distributed systems built with them are bound to be insecure.

Many approaches have been tried to decrease the threat of compromised computers. One approach is to make individual computers more resistant to attack. This approach is exemplified by SAFE [1], a system developed at BAE Systems with its university partners. Another approach is to have the nodes in a network communicate information about threatening nodes, enabling the entire network to act as an attack detector, instead of requiring each node to detect all attacks without help. This approach is exemplified by SOUND [2], a system being developed at BAE Systems with several university partners. The SOUND system extends research on Introduction-Based Routing (IBR) [3] to manage network connections between nodes in the distributed system and between the distributed system and other networks. This paper describes the advantages of using both approaches together. By using SAFE nodes as the intrusion-detection infrastructure in a SOUND-like system, one gains more reliability than one would expect from simply summing their improvements. A particularly strong version of this is to use a SAFE "dongle" between each computational node and the network. This would act as a gateway or firewall for each node. This is cost-effective, as the dongles would only have to run a network stack, rather than a full stack of applications which would be required if one were to use the SAFE nodes as the primary computation nodes. Even using one SAFE node to protect a 100 node enclave provides significantly more resistance to attack.

We loosely base our security infrastructure on the SOUND system. The results of this experiment, however, should apply to a wide range of security frameworks that attempt to detect and adapt to security incidents.

II. Model

Our system is based on IBR. In IBR, there can be no communication between two nodes until they are introduced. An introducer will only introduce nodes if both of them have good enough reputations. Reputations are completely local, so, except for sending reports of attacks, there is no coordination of reputation between nodes. The IBR protocol is discussed at more length in Section 3.A.1.

Our simulated distributed system is divided into enclaves (each with one or more computing nodes) and each enclave has exactly one proxy (or gateway) node. Proxy nodes represent security operations within an enclave (and are therefore candidates both for attack and for deployment on inherently secure nodes). All traffic into or out of the enclave passes through the proxy. Within an enclave, all nodes are connected to each other. Every proxy node is connected to every other proxy node. We do not model a connection to the wider Internet.

This is the topology of the physical connections. Any node can send a message to any other node based on IP address, but it will be ignored unless a connection has been established.

We model time in ticks. A node can send a message to any node it is physically connected to in one tick.

This material is based upon work supported by the DARPA CRASH program through the United States Air Force Research Laboratory (AFRL) under Contract No. FA8650-10-C-7090. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government

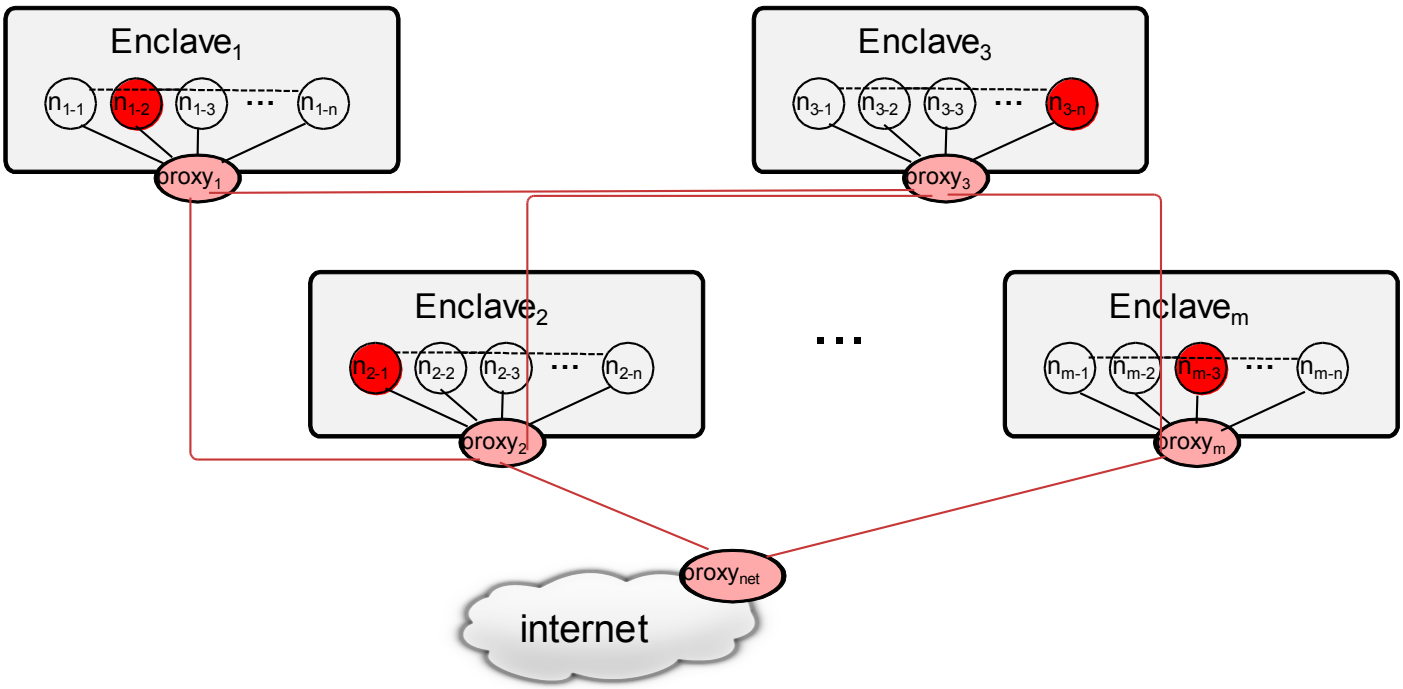

Figure 1 Enclaves of nodes. One proxy per enclave. Red nodes are compromised. All proxies connected to all other proxies.

Messages cannot be forged. That is, the identity of the sender of a message (and the entire path the message took) is what it claims to be.

In our simulations, proxies will force reboot (cleanup) of nodes that are deemed compromised.

A. Network Architecture

Proxy nodes are consulted before establishing connections between nodes; thus proxy nodes can be used to adapt the network (by refusing to make connections) to account for compromised nodes.

Figure 1 is a high-level view of a generic network of enclaves, with one proxy node per enclave. Figure 1 shows the distributed system connected to the broader internet via a proxy node; however, our simulations do not consider the case of connections outside of the distributed system under study.

In our simulation, all nodes are sensors. We consider a "fabric" of sensors, so the entire network acts as a distributed sensor. For the purposes of the model, we assume that any attack is detected at a particular node. We will use the original version of the IBR protocol, and not handle a feature that SOUND adds where nodes communicate about other nodes' reputations.

B. Health Metrics

The metric we use is **Mean healthy nodes**: The fraction of nodes in a "healthy" state when the system has reached a steady state. Steady state means that the fraction of healthy nodes remains more or less constant. It does not imply that any particular node does not change state.

C. Attack Model

When an attack occurs, we characterize it by two probabilities, the probability that the attack is detected by the network fabric, which we model as having been detected by the node which was attacked, and the probability that the attack succeeds. Any attack which is detected is assumed not to succeed. We model attack success probabilistically because even if the hardware and software on the nodes is the same, there will be different users with passwords of varying strength, and with different susceptibilities to phishing.

III. EXPERIMENTS

We can characterize our experiments as exploring the space of the three-way relation:

$$N \times SimParams \rightarrow HealthMetrics$$

where N is the number and allocation strategy of inherently secure nodes, *SimParams* are parameters to the simulation, such as the likelihood of detection of an attack or the likelihood of attack success, and *HealthMetrics* are metrics, such as *mean healthy nodes*, as previously discussed, characterizing the health of the system given the distribution of secure nodes and the simulation parameters.

We show that:

1. For a given set of parameters *SimParams*, adding inherently secure nodes (increasing N) can improve

the metrics *HealthMetrics*. For example, as one can see from Table 7, with no SAFE nodes, and parameter P(detect | attack) = 0.7 we have a steady state of 42% healthy nodes[1]. In that case, we show that adding inherently secure nodes increases the percentage of healthy nodes in the steady state to 85%. Note that this requires one safe node for every hundred computing nodes, so the cost involved is modest.

2. Adding inherently secure nodes (increasing N) increases the ranges of SimParams parameters in which metrics *HealthMetrics* are in a reasonable range. For example, Table 4 shows that with no inherently secure nodes the system reaches a steady state with 74% of its nodes healthy when the parameter P (detect | attack) is = 0.9. In that case, we show that by adding inherently secure nodes, we can reach a similar steady state with much less accurate detection (e.g. P(detect | attack) = 0.5).

A. Connection Protocol

1) IBR

To start, every node has a set of nodes to which it has *a priori* connections. If the graph of *a priori* connections is not connected, then there are nodes which will never be able to communicate with each other. Two nodes can communicate either if they have an *a priori* connection, or if they have been introduced. Once two nodes are introduced, they are said to be connected and can communicate with each other. If a node A wishes to communicate with a node B, to which it is not connected, it must find a node C which is connected to both A and B. It then requests a connection from node C. If C deems both A and B to have good reputations, it introduces them, and they are then connected and can communicate. If C considers either A or B to be unsafe, it will refuse to introduce them. If there is no node C which is connected to both A and B, A can look for a path from A to B, and ask for introductions along the path. The method of finding such a path is outside the scope of this paper.

Every node maintains a *reputation* for each node to which it is connected. A node which communicates with another node will give feedback to the node which introduced them after each communication. A typical way to maintain reputations is to start with a positive reputation, and increase it slightly with each piece of positive feedback, and decrease it greatly with each piece of negative feedback. The node then performs an introduction if both nodes are over a preset threshold. In our system, every node is *a priori* connected to the proxy for its enclave, and all the proxies a priori form a complete graph, so it is never necessary for a path to be of length more than three.

2) Use of Proxies as Introducers

A connection between two nodes is established when one node sends a message to the proxy for the enclave of the other node requesting a connection. If the proxy grants the connection, the two nodes can communicate. For simplicity, we do not distinguish between direct versus multihop communication. We also do not model timeouts on connections, as the only communication we actually model is an attack, which takes only a single message.

Intra-enclave communication: A pair of nodes wishing to communicate within an enclave needs to get a connection between them from a local proxy. Once established, intra-enclave packets are node to node. The requirement to acquire connections allows the security system to isolate problem nodes as they are identified. In Figure 1, the dashed lines represent active intra-enclave connections.

Inter-enclave communication: Packets into and out of an enclave go through proxies. To establish a connection between nodes in separate enclaves, assuming all proxies can talk directly to each other, there are three connections involved: node → proxy1 → proxy2 → node. In Figure 1, the red lines represent proxy-to-proxy connections between enclaves. In theory, a security policy can isolate enclaves by not making connections between proxies.

B. Behavior

We only model “interesting" behavior. The events, and the corresponding messages, of interest in the simulation are:

1. A compromised node attacks another node.
   Message: *attack* from compromised node to victim.
2. An attack is (correctly)[2] reported.
   Message: *report (attacker)* to proxy.
3. A proxy forces a reboot of a (compromised) node
   Message: *reboot* from proxy to compromised node
4. A rebooted node comes back online.
   Message: *online* from rebooted node to proxy.

C. Attacks

We consider what happens when a compromised node attempts to attack a node in another enclave. That is, when an attack propagates from one enclave to another.

When an attack is detected, the proxy of the target node is notified. However, when a proxy receives a notification of an attack from another enclave, it sends the report of the attack to the proxy for the originating enclave. If the receiving proxy is behaving normally, the offending attacker will eventually be rebooted. If the originating enclave keeps sending out attacks, the receiving enclave's proxy may elect to force a reboot of the entire foreign enclave.

[1] Read “P(detect | attack) = 0.7" as “the probability of detection of a given attack is 70%"

[2] We are not modeling false positive attacks reports.

A proxy may send a reboot message to another proxy. Only proxy nodes can send reboot messages. In the following, the policy is that if a proxy receives a reboot message, it in turn sends reboot messages to all of the nodes in its enclave.

A proxy keeps track of reputations for both (1) nodes in its proxy and (2) proxies for other enclaves. It can decide to send reboot messages either to (1) nodes in its enclave, or (2) proxies in other enclaves.

D. Details of Simulation

There are two types of node (proxy and regular), and each node is in one of three states: healthy, compromised, or offline. We start with one compromised regular node per enclave. In Figure 1, the red nodes represent compromised nodes. A healthy proxy node keeps track of a set of suspect nodes, forcing them to reboot when they are deemed compromised. Here any node which is detected as the source of an attack is deemed to be compromised. A compromised node selects a victim (proxy or regular node) from within its local enclave and attacks it.

An attack on a node is detected according to probability parameter P (detect | attacked). If the attack is detected, the attack is reported to the local proxy. If the attack is not detected, the node becomes compromised according to probability parameter P (compromise | attacked, not detected) (which is 0 for SAFE nodes). When a node is rebooting, it ignores all messages (though in the cross-enclave case, it might be rebooted while rebooting).

Attacks on already compromised nodes can be detected if there is a separate attack detection mechanism outside of the nodes being attacked.

At each time step, the following actions take place, according to the type of the node and its state:

Healthy regular:

1. If received *attack* message, flip P (detect | attacked-healthy) coin.
2. If attack detected, send message *report (sender)* to local proxy.
3. Handle *reboot* message. May arrive if local proxy is being rebooted by an external enclave. Transition to offline mode with countdown (reboot-node-countdown).
4. If attack not detected, flip P (compromise | attacked, not-detected) coin.
5. If compromised, transition to compromised regular node.

Compromised regular:

1. Check message queue for *attack* message (assumes attacker cannot distinguish between healthy and compromised nodes).
2. If attacked, flip P (detect | attacked-compromised) coin.

If attacked and detected, send message *report (sender)* to appropriate proxy. Note that the ability to detect attacks on compromised nodes models there being some attack detection in the system that does not necessarily rely on the target being healthy.

3. Check message queue for *reboot* message. If received, transition to offline mode, with count-down timer (node-reboot-time).
4. If not rebooted, select a target node from local enclave. Send *attack* message to target.

Healthy proxy:

1. Check message queue for *attack* message.
2. If attacked, flip P (detect | attacked-compromised) coin.
3. If attacked and did not detect and compromised, transition to compromised proxy node.
4. If attacked and detected
    a. If attacker is local to this proxy's enclave, decrement attacker's reputation.
    b. If attacker is from another enclave, decrement attacker's proxy's reputation.
5. For each incoming *report (attacker)* message,
    a. If attacker is local to this proxy's enclave, decrement attacker's reputation.
    b. If attacker is from another enclave, decrement attacker's proxy's reputation.
    c. If attacker is from another enclave, send *report (attacker)* message to other enclave's proxy.
6. Handle *reboot* message. Send *reboot* message to all nodes in this enclave. Transition to offline proxy with proxy-reboot-countdown.
7. If any nodes have low enough reputations:
    a. If bad node is in local enclave, send it a *reboot* message.
    b. If bad node is another enclave, send *reboot* message to its proxy.
8. For every node to which just sent *reboot* message, reset reputation.

Compromised proxy:

1. Handle *reboot* message. Send *reboot* message to all nodes in this enclave. Transition to offline proxy with countdown.
2. If attacked, flip P (detect | attacked-compromised) coin.
3. If attacked and detected, send message *report (sender)* to attacker's proxy.

Offline regular or proxy:

1. Ignore all messages.
2. Decrement timer.
3. If timer at zero, transition to uncompromised.

IV. RESULTS

A. Data

In Tables 1-5, Frac SAFE is the fraction of Proxy nodes which are immune to attack. Every enclave has exactly one proxy node, in addition to the clients listed. All client nodes can be compromised. The internal table values are the fraction of the nodes which are HEALTHY in what appears

to be a steady state. In almost all cases, the system entered what appeared to be a steady state in 100 or fewer ticks. We continue the run for 1000 ticks to see if it indeed stays steady. In some cases, all the nodes became COMPROMISED, or all the nodes became HEALTHY. In either of those cases, the run is said to have ended when that occurred.

- P_{detect} is the probability that an attack will be detected
- $P_{compromise}$ is the probability that an attack that is not detected will compromise the attacked node.
- All runs have "offline time" (that is, the time between when a node gets a reset signal and when it finishes rebooting) of 5 ticks.
- All runs use a "one strike and you're out" rule for when to force a node to reboot.
- All runs use an "attack delay" (the time between attacks by a single node) of 2 ticks.
- All reported data are the average of 50 runs.

1) 20 Enclaves, 50 Clients/Enclave

In Tables 1 through 5 we plot the fraction of healthy nodes at steady state as a function of P_{detect} and the fraction of proxies which are SAFE nodes at different values of $P_{compromise}$. The only differences between the first five tables are the values of $P_{compromise}$. Most of the runs went for the full 1000 ticks allowed, but a few ended in 10 to 20 ticks. In Table 3 the value 0.02 for P_{detect} = 0:7, no SAFE nodes, comes from 1 run which quickly became all HEALTHY, and 49 runs which relatively quickly became all COMPROMISED.

What we learn from these tables is that even relatively small numbers of SAFE nodes go a long way toward making a resilient network. For example, in Table 2, where $P_{compromise}$ is 0.8, reading a row from left to right, such as when P_{detect} is 0.7 if there are no SAFE nodes, all the nodes quickly become compromised, but, when all the proxies are SAFE nodes (which is still only 2% of the total nodes), one gets 27% HEALTHY nodes in the steady state.

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.00	0.00	0.02	0.03	0.04	0.08
0.3	0.00	0.01	0.02	0.06	0.08	0.13
0.5	0.00	0.02	0.03	0.08	0.11	0.18
0.7	0.00	0.02	0.04	0.09	0.12	0.21
0.9	0.00	0.02	0.05	0.10	0.15	0.22

Table 1: P_{detect} vs. Fraction of SAFE nodes, 20 enclaves, 50 clients per enclave, $P_{compromise}$ = 1.0

1) 10 Enclaves 100 Clients/Enclave

In Table 6, we consider having fewer larger enclaves – 10 enclaves of 100 nodes each. It appears from Table 6 and Table 4 that with a fixed number of SAFE nodes, it almost doesn't matter if you have 10 enclaves of 100 clients each with all proxies being SAFE nodes, or 20 enclaves of 50 clients each, with half the proxies being SAFE nodes.

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.00	0.00	0.01	0.03	0.06	0.09
0.3	0.00	0.03	0.03	0.07	0.11	0.18
0.5	0.00	0.02	0.04	0.09	0.14	0.22
0.7	0.00	0.02	0.07	0.11	0.19	0.27
0.9	0.00	0.02	0.08	0.13	0.20	0.28

Table 2: P_{detect} vs. Fraction of SAFE nodes, 20 enclaves, 50 clients per enclave, $P_{compromise}$ = 0.8

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.00	0.00	0.02	0.07	0.09	0.12
0.3	0.00	0.05	0.08	0.18	0.17	0.23
0.5	0.04	0.03	0.08	0.16	0.20	0.31
0.7	0.02	0.02	0.13	0.18	0.29	0.37
0.9	0.00	0.07	0.21	0.25	0.32	0.43

Table 3: P_{detect} vs. Fraction of SAFE nodes, 20 enclaves, 50 clients per enclave, Pcompromise = 0.6

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.06	0.18	0.21	0.17	0.17	0.32
0.3	0.20	0.16	0.34	0.41	0.53	0.61
0.5	0.32	0.52	0.51	0.64	0.75	0.79
0.7	[3]	0.62	0.79	0.83	0.88	0.85
0.9	0.74	0.83	0.87	0.90	0.91	0.92

Table 4: P_{detect} vs. Fraction of SAFE nodes, 20 enclaves, 50 clients per enclave, $P_{compromise}$ = 0.3

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.40	0.73	0.75	0.82	[4]	0.93
0.3	1.00	1.00	1.00	1.00	1.00	1.00
0.5	1.00	1.00	1.00	1.00	1.00	1.00
0.7	1.00	1.00	1.00	1.00	1.00	1.00
0.9	1.00	1.00	1.00	1.00	1.00	1.00

Table 5: P_{detect} vs. Fraction of SAFE nodes, 20 enclaves, 50 clients per enclave, $P_{compromise}$ = 0.1

[3] 22 runs ended before 30 ticks. Ran for 2000 ticks, 30 runs ended before 90 ticks. After 1000 ticks, the fraction HEALTHYwas dropping steadily. Ran it for 2000 ticks - it was dropping at 1000 ticks, but may have steadied at 0.60 at 2000 ticks.

[4] no steady state. After 1000 ticks, it was at 0.89, but dropping 0.01 every 200 or so ticks

Frac SAFE	0	0.2	0.6	1.0
P_{detect}				
0.1	0.08	0.16	0.19	0.32
0.3	0.28	0.30	0.43	0.60
0.5	0.37	0.41	0.64	0.77
0.7	0.42	0.63	0.88	0.85
0.9		0.85	0.95	0.92

Table 6: Fraction SAFE nodes vs. P_{detect}, 10 enclaves, 100 clients per enclave, $P_{Compromise} = 0.3$

2) 254 Enclaves, 1 Client per Enclave

One good use for SAFE nodes would be to use them as "dongles" each computing node and the rest of the network. They would need only a minimal amount of software, and could be made rather inexpensively. As Table 7 shows this gives a large increase in the usability of a system.

Frac SAFE	0	0.1	0.3	0.5	0.7	1.0
P_{detect}						
0.1	0.02	0.06	0.18	0.29	0.39	0.57
0.3	0.00	0.08	0.21	0.33	0.47	0.63
0.5	0.00	0.11	0.26	0.41	0.53	0.71
0.7	0.03	0.14	0.31	0.43	0.59	0.75
0.9	0.00	0.16	0.38	0.49	0.61	0.79

Table 7: P_{detect} vs. fraction SAFE nodes, 254 enclaves, 1 node per enclave $P_{compromise} = 0.7$

B. Discussion

We have shown that converting even a small fraction of the proxy nodes in a system to SAFE nodes gives a large gain in the usability of a computer network under attack. Since nearly every network is under attack at almost all times, it is clearly cost effective to have routers and similar infrastructure nodes inherently secure. Even better would be to have small secure "dongles" running SOUND or some similar protocol between every node and its network.

REFERENCES

[1] Udit Dhawan, Albert Kwon, Edin Kadric, Cătălin Hriţcu, Benjamin C. Pierce, Jonathan M. Smith, André DeHon, Gregory Malecha, Greg Morrisett, Thomas F. Knight, Jr., Andrew Sutherland, Tom Hawkins, Amanda Zyxnfryx, David Wittenberg, Peter Trei, Sumit Ray, and Greg Sullivan. Hardware support for safety interlocks and introspection. In SASO Workshop on Adaptive Host and Network Security, September 2012.

[2] Michael Figueroa, Karen Uttecht, and Jothy Rosenberg. A SOUND approach to security in mobile and cloud-oriented environments. In 2015 IEEE International Symposium on Technologies for Homeland Security, 2015.

Gregory Frazier, Quang Duong, Michael P. Wellman, and Edward Petersen. Incentivizing responsible networking via introduction-based routing. In Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings, pages 277--293, 2011.