Inf Syst Front (2015) 17:987–995
DOI 10.1007/s10796-015-9569-y

Surviving advanced persistent threats in a distributed environment – Architecture and analysis

Ruchika Mehresh[1] · **Shambhu Upadhyaya**[1]

Published online: 5 June 2015

Abstract Designing robust mission-critical systems demands bringing together fault tolerance and security. The emergence of Advanced Persistent Threats (APT) has further added to the challenge of meeting mission assurance goals. Despite the advances in mission survivability, the existing solutions remain ineffective against APTs. In this paper, we propose a novel survivability architecture against APTs in a distributed environment. It involves tamper-resistant and surreptitious detection and node-to-node verification of suspicious events. The solution aims to identify Attacker Intent, Objectives and Strategies (AIOS) and to design targeted recoveries that promote survivability. Its security strength has been theoretically analyzed, while the performance and scalability aspects are measured via simulation. Our simulations demonstrate high scalability with respect to network size and application runtime and the time overhead for long running applications can be easily kept under 1 % of original runtime by carefully adjusting the security strength.

Keywords Intrusion detection · Mission-critical systems · Simulation · Tamper-resistant monitoring

✉ Shambhu Upadhyaya
shambhu@buffalo.edu

Ruchika Mehresh
ruchika.mehresh@gmail.com

[1] University at Buffalo, The State University of New York, Buffalo, NY 14260, USA

1 Introduction

Recent advances in cyber warfare such as the emergence of Advanced Persistent Threats (APT) (Brewer 2014) have witnessed a decline in the effectiveness of traditional mission assurance solutions. APTs are usually characterized by extreme stealth, advanced skillset, vast resources and hence, a markedly high success rate. Survivability "is the capability of a system to fulfill its mission in a timely manner in the presence of attacks, failures, or accidents" (Ellison et al. 1999). This paper presents a mission survivability architecture to deal with APTs in a distributed environment. The proposed solution leverages concepts of deception (Cohen et al. 2001; Yuill et al. 2006) in a hardware-based security setup (Mehresh and Upadhyaya 2012; Kamhoua et al. 2013). If suspicious activity is detected but any imminent danger is not perceived, our approach refrains from sending clear signals to the attackers such as raising alerts or terminating a session. Instead, the system continues to behave normally and make observations to better understand the Attacker Intent, Objectives and Strategies (AIOS). This approach has two advantages. First, it assists in the designing of targeted recovery procedures which are lightweight. Second, by maintaining status quo, it prevents any harmful reactions from the attacker. The system uses built-in replication and fault-tolerance to compensate for any side-effects that may be caused by this apparent inaction. Our mission survivability solution works as follows. Instead of relying on a generic mix of traditional solutions, it assists in the design of a more effective (targeted) recovery in response to an advanced attack. In case of stealth attacks, our approach buys the system time to figure out the AIOS while the attacker remains oblivious of any detection efforts. We successfully avoid using any heavyweight solutions such as honeypots for achieving this goal. Note that no current or prior survivability solutions work in this manner; the general rule being

detect anomalies/attacks and raise alerts to initiate a response. However, we delay response (where one could afford, e.g., long running missions), cover the losses using replication, buy time to profile an attack and earn all the aforementioned advantages. The major contribution of this paper is the development of a novel architecture to handle APTs in a distributed environment by extending our prior works on tamper-resistant security monitoring. We analyze the proposed solution for its security and tamper-resistance properties and measure the overheads associated with adopting the proposed survivability solution.

We review the background in Section 2 and present a detailed attack model in Section 3.1. Section 3.2 describes the architectural requirements and Section 3.3 details the proposed architecture. The operational details of the architectural framework are given in Section 3.4. We evaluate our architecture in Section 4 from the point of view of security and performance, which is followed by a discussion and conclusion in Section 5.

2 Background

This paper extends our prior research (Mehresh et al. 2011) on mission assurance wherein the solution approach relies upon a centralized, replicated architecture. All replicas are assumed to be executing in lockstep, regulated by a central authority, also called the coordinator. The coordinator distributes workload among these replicas, each running on separate hardware. A host-based Intrusion Detection System (IDS) such as Tripwire (Kim and Spafford 1994) monitors each node for signs of intrusion. Any intrusion attempts are translated into an integrity-signature and stored away in the hardware (e.g., the Trusted Platform Module (TPM) (Trusted Comp. Group 2007)), where it stays hidden from the attacker. Replicas send periodic checkpoints to the coordinator for verification. The coordinator responds back with a go-ahead signal if nothing out-of-ordinary is detected. The integrity-signatures stored in the hardware are piggybacked to these checkpoints and are surreptitiously submitted to the coordinator for verification. Thus, the coordinator knows about the possibly compromised replicas and can ignore their submissions in the consensus protocol. However, it still sends go-ahead signals to all the replicas. The idea is to hide any detection information from an attacker until the extent of the compromise is fully known to the administrator. This prevents any unfavorable reactions by an aggressive or desperate attacker (as discussed in Sec. 1). Note that this wait-and-watch approach will only be employed if imminent damage to the system is not expected.

The base scheme described above can be extended to design a survivability solution in a distributed environment. However, there are two major problems that need to be addressed first – (i) how to ensure tamper-resistance of the IDS at each node? (*Problem 1*), and (ii) how to ensure secure communication between nodes? (*Problem 2*). The solution draws from our prior work (Mehresh et al. 2011), a brief description of which is provided below.

Problem 1: IDSs are usually deployed in user-space making them as vulnerable as the processes they are monitoring. Mehresh et al. (2011) proposed a coveillance-based watchdog framework to this classic problem of 'who watches the watcher.' The chosen solution involves lightweight processes called process-monitors monitoring each other in a cyclic fashion. Since cycles do not have any loose ends, all participating process-monitors are monitored at all times. The simplest (ring) topology of this framework for a multicore processor is shown in Fig. 1. In a uni-core environment, all process-monitors in the ring run on the same processor. A process-monitor from this ring is assigned an additional responsibility of monitoring the IDS (e.g., Tripwire). Any tampering of Tripwire is reported by the process-monitor and captured as an integrity-signature, stored in the hardware as discussed in the base scheme. Processes monitor each other using kernel facilities such as a *kqueue* subsystem in a tamper-resistant configuration.

Problem 2: The TPMs that come integrated into the motherboard of modern computer systems can be used to implement secure node-to-node communication. A TPM is a secured micro-controller with cryptographic functionalities. Its capabilities include random number generation, RSA encryption/decryption, SHA-1 hash calculations and limited NVRAM (Bajikar 2002). The TPM contains 16 internal memory slots called Platform Configuration Registers (PCR) that are initialized to known values. A function call PCRExtend is the only way a software can update the values of PCRs. Once updated, these values are securely stored in the TPM. This call can be used by the IDS to commit an integrity-signature for storage in hardware. API calls like Seal/Unseal/Bind/Unbind/Quote, etc., can be used to avail authentication, confidentiality and integrity features provided by the TPM.

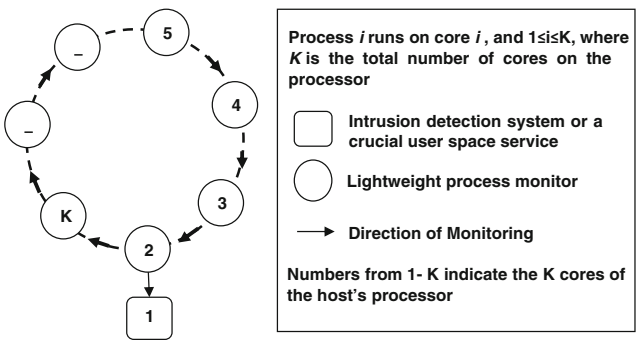

Fig. 1 Simple ring topology for tamper-resistance monitoring of an IDS on a multi-core processor

3 A distributed architecture for survivability

3.1 Attack model

We now narrow down the broad category of APTs to a limited and workable set of salient components. The APTs are mostly multi-phased and stealthy. APTs are known to follow a lifecycle with the following stages (Repik 2008; Gragido and Pirc 2011; Mcwhorter 2013):

- *Initial compromise:* Social engineering, phishing, zero-day viruses, etc.
- *Establish foothold:* Install backdoors, Trojan horses, etc.
- *Escalate privileges:* Gain administrator privileges using exploits, password cracking, etc.
- *Internal reconnaissance:* Collect confidential infrastructure information.
- *Move laterally:* Compromise more internal systems.
- *Maintain presence:* Ensure continued control over channels and credentials without raising red flags.
- *Complete mission:* Steal data or compromise mission at an appropriate time.

3.2 Architectural requirements

We now list down the desired features of a good survivability solution:

- The solution (base scheme) by Mehresh et al. (2012) worked only in a centralized setting. However, with the advent of cloud technology, any survivability solution cannot be considered of generic use until it runs efficiently in a distributed environment. Thus, it is desirable to include features that make a solution highly adoptable in a distributed environment.
- The solution should be platform-independent and support the Commercial Off-The-Shelf (COTS) paradigm. We do not want to restrict the use of our solution to a closed system or to have security policies that restrict the set of systems it can communicate with.
- The solution should itself be tamper-resistant.
- The solution should deal with the APT model described in Sec. 3.1. As a result, it should consider the repercussions before acting a certain way or divulging information observable by an attacker.
- The solution must ensure mission survivability. It should be lightweight to preserve mission's timeliness property. However, predictable delays are allowed and can be accounted for in the original mission timeline.

3.3 The architecture

To achieve the objective of mission survivability in a distributed system, the following features are essential:

- *Tamper-resistant monitoring of critical processes at each node:* For this purpose, we employ the solution of Problem 1 as described in Sec. 2. All critical processes (including the IDS) are provided with tamper-resistant monitoring using a lightweight, cyclic watchdog framework.
- *Tamper-resistant and surreptitious capture and storage of integrity-signature at each host:* This can be achieved by using the TPM as discussed in solution for Problem 2 under Sec. 2.
- *Tamper-resistant and surreptitious reporting of alert information to a remote authority:* This can be achieved by using a TPM and software-based services as discussed in solution for Problem 2 under Sec. 2.

We design each node in our distributed framework to have the following components as shown in Fig. 2.

- *Hardware Replicas:* Replication provides the redundancy required for fault-tolerance. Our solution delays recovery (or any other response if immediate danger is not perceived) until it has assessed the extent of system compromise. Moreover, even if some replicas are compromised near mission completion, mission can still survive with the help of the remaining untampered replicas.
- *Effective isolation of replicas:* It is essential that replicas are diverse and well isolated to increase their robustness against repetitive attacks. An ideal environment should have each replica running on a separate physical machine. However, virtual machines are good alternatives as well.
- *TPM daemon:* The TPM does not know what to do with the data that it stores and processes. A TPM daemon is the logical layer over the TPM that provides useful functionalities for other applications. For instance, parsing and structuring the data decrypted by the TPM to a form usable by other applications, scheduling tasks to TPM, etc.
- *Voting service:* Based on the information provided by the TPM daemon, the voting service decides which replicas are to be trusted. From among the results reported by these trusted replicas, a voting (majority, random selection, etc.) is conducted to determine the final result.

3.4 Operational details

Let S be a distributed network of nodes $\{S_i\}_{i=1}^{n}$ running a particular mission critical application such as a process control

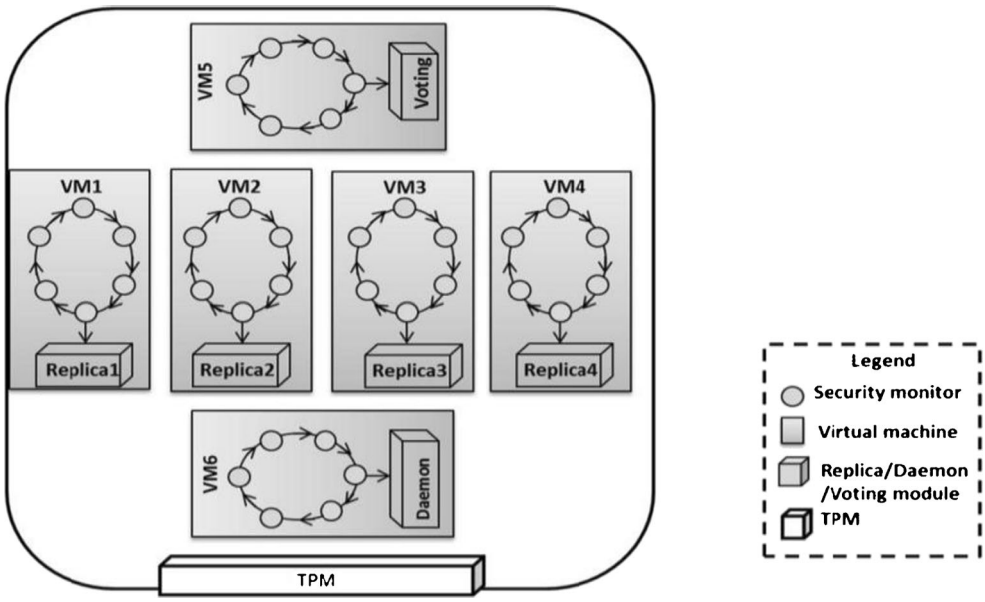

Fig. 2 Various sub-components of a node in the proposed distributed survivability solution

system or a peer-to-peer computation requiring certain level of mission assurance. Each node S_i calculates an intermediate result and submits it to another node in the distributed platform. Every node has *m* replicas running the same application in lockstep. Therefore, it submits $\{\rho_j\}_{j=1}^{m}$ copies of a result. A node can run in any one of the following modes: Process, Send, Receive and Vote.

In 'Process' mode, a few threads (δ) run replicas while others run a tamper-resistant cyclic watchdog framework monitoring each replica. A participating node combines input received from another node ($\widehat{\rho}$), if applicable, with its own data. This data is then processed by the local replicas. Meanwhile, if the monitoring framework senses intrusion, it commits a change in the value of integrity-signature ψ stored at the hardware level. After the processing at all replicas is complete, the node switches to 'Send' mode. In this mode, it securely sends the set of results $\{\rho_j\}_{j=1}^{m}$ along with the integrity-signature ψ to the next node as determined by the application. This secure communication between nodes is TPM-assisted. If the node has any further tasks pending, it switches to 'Receive' mode and waits for further input; otherwise, it stops processing.

Upon receiving data, a node switches to 'Voting' mode. Here, it uses ψ to derive a set C of replicas suspected to be compromised at the sending node. It disregards all votes from these replicas and performs voting on the rest to derive the result $\widehat{\rho}$. It then switches to 'Process' mode. At application startup, all nodes are either in the 'Receive' or 'Process' mode depending on the design. These operational steps are formally described in Fig. 3.

An instance of our mission survivable architecture is shown in Fig. 4. Here, we consider a sample peer-to-peer application consisting of four geographically distributed nodes which represent Processing Elements (PE) and a client (laptop). All PEs gather data from their respective environments. They process this data and forward a set of results to their peers chosen by the application. The receiving PE combines the received information with its own data, processes it and forwards its set of results to the next PE. The node at location 1 has several replicas running the application. These replicas are being monitored by their respective cyclic watchdog frameworks that commit an integrity-signature to the TPM if suspicious activity is

```
1: while (System = 'Running') do
2: if (MODE = 'Process') then
3:    Run parallel threads δ for replicas and process- monitors
4:    if (δ = 'Replica') then
5:       if (Pending tasks ≠ NULL) then
6:          if (ρ̂ ≠ NULL) then
7:          Combine ρ̂ with node's data
8:       end if
9:       Feed data to replicas for processing
10:      Wait for processing to complete
11:   end if
12:   Switch to MODE = 'Send'
13:  end if
14:  if (δ = 'Process-Monitor') then
15:    while (TRUE) do
16:      if (Suspicious activity detected) then
17:        Change the value of ψ accordingly
18:      end if
19:    end while
20:  end if
21:else if MODE = 'Send' then
22:  SecureSend({ρ_j}_{j=1}^m, ψ)
23:  if Pending tasks ≠ NULL then
24:    Switch to MODE = 'Receive'
25:  else
26:    END
27:  end if
28:else if MODE = 'Receive' then
29:  Receive({ρ_j}_{j=1}^m, ψ)
30:  Switch to MODE = 'Vote'
31:else if MODE = 'Vote' then
32:  Analyze ψ to identify C, a set of compromised replicas on the sending node
33:  Vote on {ρ_j}_{j=1}^m where j ∈/ C
34:  Switch to MODE = 'Process'
35:  else
36:  Error: Unsupported MODE
37:  end if
38: end while
```

Fig. 3 Pseudocode - Operational details

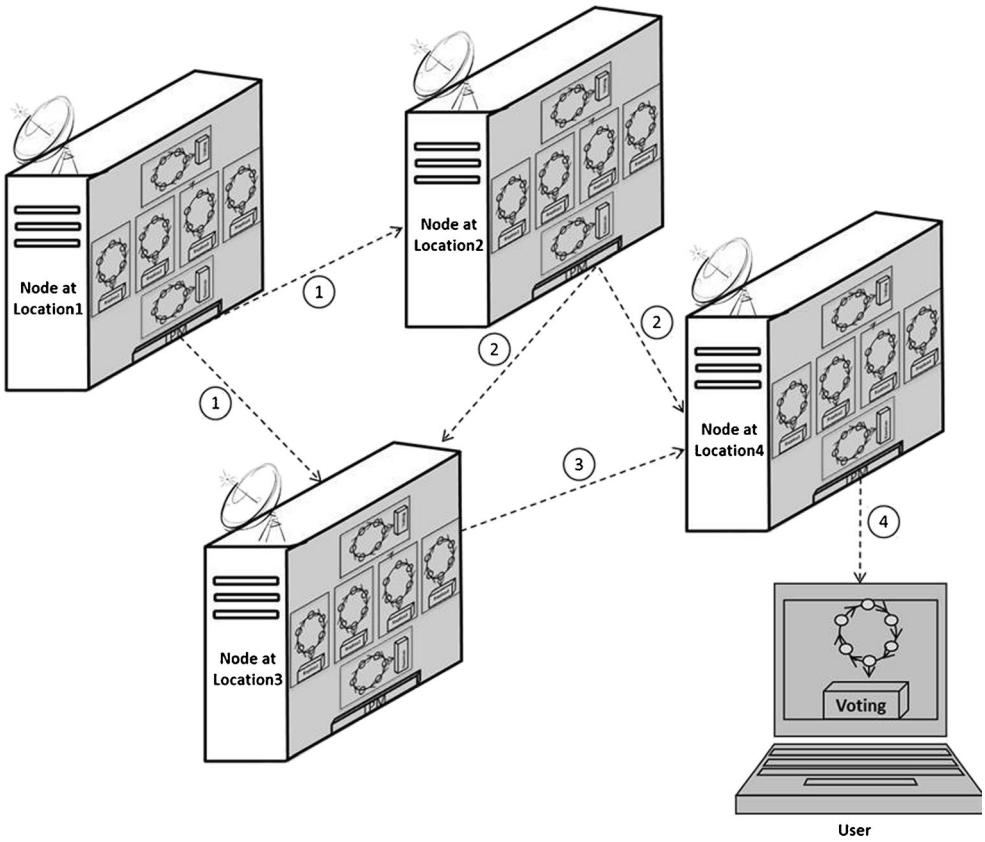

Fig. 4 A distributed architecture for mission survivability against APT

detected. A set of results from Node 1 is sent to nodes at locations 2 and 3. Along with the set of results, the daemon sends over TPM information as well. The TPM daemon at the destination node parses this information and delivers it to the voting service. The Voting service determines which results (replicas) are to be trusted and uses them in further processing. Such exchanges go on until a set of results is finally delivered by node 4 to the client. Client, in this case, needs to run a TPM, a daemon and a voting service in order to get the final result.

4 Evaluation

4.1 Security analysis

The solution presented in this paper shows a combination of reactive and proactive strategies and is adaptive. Along with reporting the abnormal integrity signatures to remote inspectors, it also decides how to respond to such detections locally. If a threat is perceived to have a high severity, such as privilege escalation or changing important system files, an immediate action is taken (e.g., blocking the malicious process). However, suspected activities that score low on the severity level are just recorded and reported but no response is invoked locally. Behind the scenes, this reported activity is analyzed to determine the AIOS and devise a targeted response and recovery strategy. Such a strategy involves identifying all the attacker footholds and actions required to block them out all at once.

4.2 System modeling

In this section, we model the security dependencies in our system to analyze its security strength using propositional logic as described by Schryen et al. (2011). The end goal is to determine the trustworthiness of a distributed system that employs our solution to deliver the correct result while fulfilling a set *r* of certain security requirements.

Each node S_i, where $1 \leq i \leq n$, has the following components (as described in Sec. 3.3):

- *m* number of replicas running the same application on separate physical or virtual machines, collectively represented as $\{R_j\}_{j=1}^{m}$. Each replica is monitored by a cyclic watchdog framework $\{P_k\}_{k=1}^{\{c_l\}_{l=1}^{m}}$, where P_k is a process-monitor and c_l is the total number of process-monitors monitoring each replica. This setup is easily adoptable by both uni-core and multi-core platforms.
- TPM hardware at node *i*, represented as T_i.

- TPM logical layer daemon at node *i*, represented as D_i, monitored by a cyclic watchdog framework $\{P_s\}_{s=1}^{a}$, where P_s is a process-monitor and *a* represents the number of participating process-monitors.
- Voter at node *i*, represented as V_i, performs voting functions and is monitored by a cyclic watchdog framework $\{P_t\}_{t=1}^{b}$, where P_t is a process-monitor and *b* represents the number of participating process-monitors.

When a client submits a request to this distributed system, it trusts that all its nodes will satisfy *r*.

$$S = S_1 \wedge S_2 \wedge S_3 \wedge \ldots \wedge S_n \tag{1}$$

For each S_i to fulfill the security requirements *r*, all its components should satisfy *r*.

$$S_i = T_i \wedge D_i \wedge V_i \wedge R_1 \wedge R_2 \wedge R_3 \wedge \ldots \wedge R_m \tag{2}$$

Tamper-resistant functioning of these components relies on the tamper-resistant functioning of their respective coveillance-based watchdog frameworks. Thus,

$$S_i = T_i \wedge \underbrace{\{P_i \vee P_2 \vee \ldots\ldots \vee P_a\}}_{D_i} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_b\}}_{V_i} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_{c_1}\}}_{R_{i_1}} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_{c_2}\}}_{R_{i_2}} \wedge \ldots . \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_{c_m}\}}_{R_{i_m}} \tag{3}$$

Assuming a majority voting, we require a minimum of three untampered replicas for a voter to deliver result. Thus,

$$S_i = T_i \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_a\}}_{D_i} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_b\}}_{V_i} \wedge \underbrace{\{P_i \vee P_2 \vee \ldots\ldots \vee P_{c_1}\}}_{R_{i_1}} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_{c_2}\}}_{R_{i_2}} \wedge \underbrace{\{P_1 \vee P_2 \vee \ldots\ldots \vee P_{c_m}\}}_{R_{i_3}} \tag{4}$$

should satisfy the set of security requirements *r*.

In a cyclic watchdog configuration, there is a high probability that all process-monitors are monitored at all times (Mehresh et al. 2011). Thus, they cannot be tampered without raising an alert. Therefore, they can be assumed to be tamper-resistant with a probability approaching 1.

$$S_i = T_i \wedge \underbrace{\{1\}}_{D_i} \wedge \underbrace{\{1\}}_{V_i} \wedge \underbrace{\{1\}}_{R_{i_1}} \wedge \underbrace{\{1\}}_{R_{i_2}} \wedge \underbrace{\{1\}}_{R_{i_3}} \tag{5}$$

Combining (1), (2) and (5), we get that

$$S = \{T_1 \wedge T_2 \wedge T_3 \wedge \ldots \wedge T_i\} \tag{6}$$

should satisfy the security requirements *r*.

From (6), we conclude that reliability of this system depends on the reliability of its TPM components to satisfy *r*. In general, TPMs are considered to be highly reliable (Bajikar 2002) which indicates the high reliability of our solution.

4.3 Simulation

We simulate a generic Peer-to-Peer (P2P) distributed network to measure the performance of the proposed solution. All nodes are connected to one another directly or through a string of peers. Clients can submit their tasks to any of the participating nodes and get a result back from the same node.

We use a 32-bit Windows system with 2 GB RAM and Intel Core Duo processor for running SimJava (2.0) simulations. Simulation parameters are derived using the approach recommended by Mehresh et al. (2010). The goal of simulation is to evaluate time performance and scalability of our solution. Space overhead is not considered particularly because the memory requirement is not significant if replicas are already a part of system's fault-tolerance setup. This is because all the other system components (daemons and TPM) are extremely lightweight in terms of memory (Mehresh et al. 2011).

For the evaluation, we consider missions that run for a long time because APTs usually have heavier impact on such missions. Long missions, if interrupted at a later stage, require lots of resources and time for re-runs (if re-runs are possible at all). Short missions can always be reset, recovered or replayed if found corrupted or compromised.

Note that there are no existing systems of this kind to provide a benchmark for a reliable comparison. That is why we use a multi-step evaluation as described by Mehresh et al. (2010). The results are an approximation of the real-world values and not relative to any baseline.

In Fig. 5, we depict the scalability of our solution with respect to mission's runtime. The x-axis represents average runtime per node while the y-axis represents time overhead as a percentage of original mission runtime on a logarithmic scale. We run this simulation for five different network sizes. For evaluation purposes, we assume a simple majority voting. In Fig. 5, we do not employ RSA for encrypted and authenticated communication between nodes. Instead, we use a simple challenge-response authentication mechanism with no encryption. This communication security setup assumes a weak APT model where the attacker has less or no control over the communication channel. We observe that the (time) overhead percentage decreases with increasing mission runtime. This is because for a fixed network size, time overhead increases at a much slower rate than mission runtime, making our solution extremely scalable. However, as the network size increases (from 10 to 90 nodes), we notice a slight increase in time overhead. We will investigate this scalability aspect shortly. A major takeaway from this simulation is that for any network

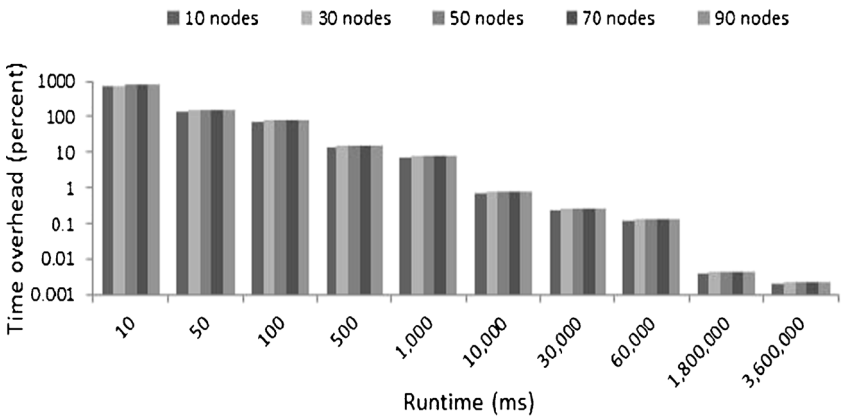

Fig. 5 Time overhead for different network sizes in the absence of RSA-based TPM security

size, missions with runtime greater than 1 s (average per node) incur a time overhead of less than 10 %. Furthermore, missions with an average runtime of greater than 10 s per node have a time overhead of less than 1 %. This number decreases further as the mission runtime increases.

Employing RSA for communication security (using TPM) incurs additional time overhead. Encrypting the communication in previous simulation with an RSA key-size of 512 bits and data-size of 20 bytes, we get the measurements as shown in Fig. 6. Here, the time overhead observed follows the same trend as in Fig. 5. However, the 10 % time overhead cutoff is for missions where average runtime per node is 10 s or greater. The 1 % time overhead cutoff is for the missions with 30 s or greater of average runtime per node.

We now analyze the scalability of our solution with respect to network size (number of nodes). Thus, in Fig. 7, the x-axis represents network size while the y-axis represents time overhead as a percentage of original mission runtime on a logarithmic scale. This simulation uses a simple challenge-response mechanism for authentication (no RSA). We observe that time overhead increases drastically as the network size expands from 3 to 10 nodes; it increases very slowly afterwards. Note that we work with average runtime per node. This means that when we consider a bigger network, we also get a longer runtime. This explains the trend we see for this simulation. For a mission with an average runtime of 1 s or greater per node, the time overhead is bounded by 10 % irrespective of the network size. For a mission where the average runtime per node is 10 s or greater, this bound decreases to 1 %. These observations demonstrate the high scalability of our solution for long running missions irrespective of the network size.

Figure 8 measures the same parameters as measured in Fig. 7, except that it uses RSA for communication security. We use a key-size of 512 bits and data-size of 20 bytes. The same trends are observed in both cases as shown in Figs. 8 and 7. However, because of the increased time overhead, for achieving the 1 % bound, a mission needs to have at least 30 s of average runtime per node.

Note that we have used a specific key and data size for our evaluations above. However, if we need the TPM to provide better security strength, it may need to use stronger RSA parameters. Figure 9 measures the time overhead for the different levels of RSA-based TPM security.

In this figure, the x-axis represents network size and the y-axis represents time overhead as a percentage of original mission runtime. Key-size and data-size are the two parameters that determine the security strength of RSA. Table 1 lists the various cases of RSA security settings for the TPM. Case 1

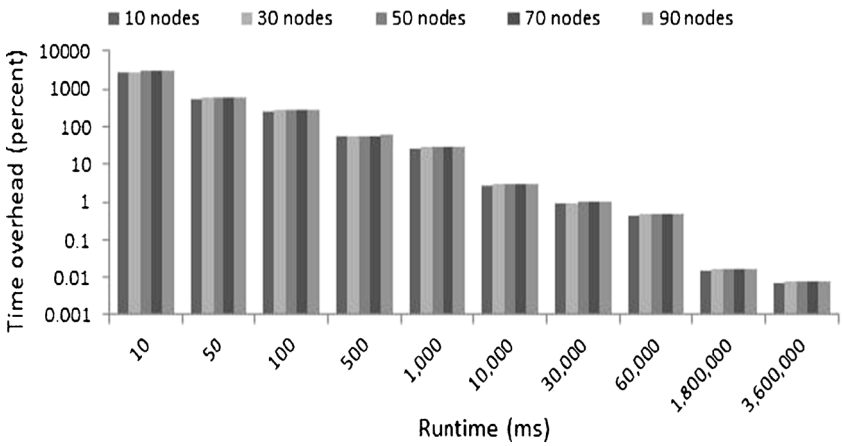

Fig. 6 Time overhead for different network sizes in the presence of RSA-based TPM security

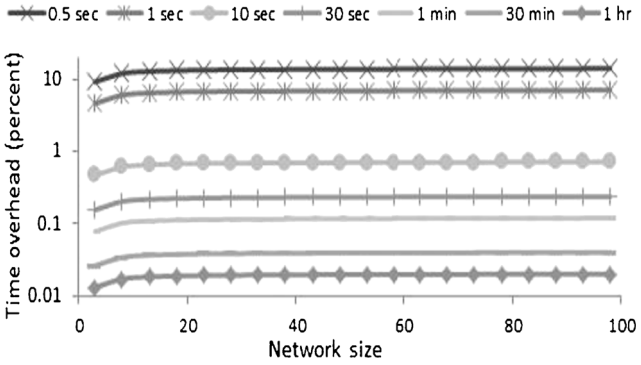

Fig. 7 Time overhead for different average mission runtimes (at each node) in the absence of RSA-based TPM security

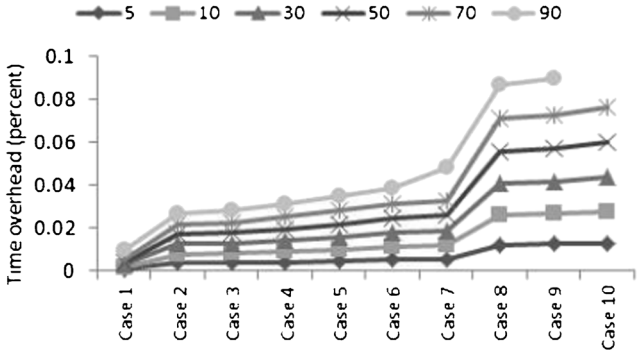

Fig. 9 Time overhead plotted against levels of security provided by TPM for different network sizes

refers to the scenario with challenge-response based authentication and no RSA. Rest of the cases use RSA with the specified parameters. We observe that there is a slow increase in overhead when data-size increases and key-size remains the same. However, an increase in key-size increases the time overhead drastically. Cases 2, 5 and 8 with steeper positive slopes of time overhead mark the increasing key-size. All other cases with a gradually increasing slope are for the increasing data-size. Note that the time overhead shoots up as the key-size reaches 2048, which provides very strong security but may cause excessive time overhead. The use of AES instead of RSA for encryption would definitely bring down the time overhead considerably, however, we considered RSA due to the strong support of Public Key Infrastructure (PKI) by the TPM.

5 Discussion and conclusion

We have presented a framework for mission survivability against advanced persistent threats in a distributed environment. The basic idea is to buy defense enough time (if imminent damage is not perceived) to figure out the precise nature of an attack. This helps in the designing of targeted recovery procedures. The framework also helps to disguise the knowledge of detection from the attackers. This 'false assurance' could prevent the attacker from switching to an aggressive strategy or a contingency plan.

We have described an extensive attack model and explained how the proposed solution deals with it. We have simulated and evaluated a generic P2P network that employs our solution. The measurements indicate that our solution incurs a time overhead that constitutes only a small fraction of mission's runtime. This fraction decreases drastically as the mission runtime increases. This solution has low time and performance overhead because it leverages many features that should already be present in a mission-critical system (such as fault tolerance and replication). It also leverages hardware and other lightweight frameworks. The only processing-intensive operation is AIOS mapping of the suspicious activity. This, however, can be offloaded to dedicated remote servers and should not impact the primary application. Because the response to an attack is delayed in our framework, a maximum bound on the number of recoveries can be proactively calculated and accounted for in the original mission timeline. This prevents the reactive recoveries, whenever initiated, from violating mission survivability's timeliness property.

An important fact to keep in mind is that existing IDSs are not highly effective in detecting APTs. There are two major factors that lead to this decreased effectiveness of IDSs. First

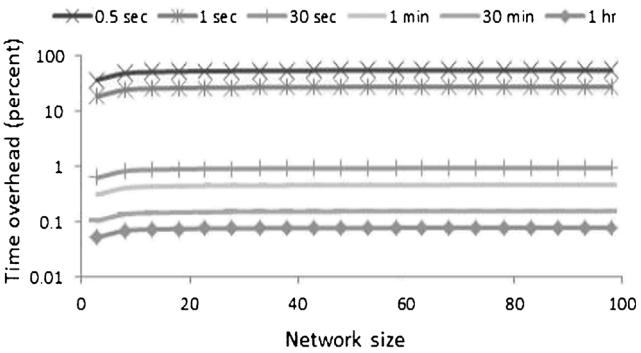

Fig. 8 Time overhead for different average mission runtimes (at each node) in the presence of RSA-based TPM security

Table 1 Cases for different levels of RSA-based TPM security

	RSA key-size (bits)	Encryption data-size (bytes)
Case 1	–	–
Case 2	512	20
Case 3	512	50
Case 4	512	100
Case 5	1024	20
Case 6	1024	50
Case 7	1024	100
Case 8	2048	20
Case 9	2048	50
Case10	2048	100

is their inability to operate at a high-sensitivity level because that usually leads to more false positives. Second is the lack of real-time data collection and correlation. Usually, IDSs follow a detect-and-react approach that may lead to Denial of Service (DoS) if a high number of false-positives are generated. Because our framework follows a wait-and-watch approach, it enables the IDSs to operate at a high-sensitivity level without the risk of experiencing DoS. This also leads to a real-time accumulation of data that can be analyzed to detect APT patterns and generate an AIOS profile. Therefore, although our framework does not directly improve the quality of the existing IDSs it allows them to operate with increased effectiveness.

The system proposed in this research is comprised of numerous components. Most of these components like the IDS and TPM are already used in large commercial systems. Our future work includes the development of methods to classify attacks based on their severity followed by a detailed mathematical analysis of security to determine the accuracy of the framework and the actual implementation of the framework in a prototype form and its evaluation.

Acknowledgments This research is supported in part by National Science Foundation Grant No. DGE -1241709. A preliminary version of this paper has been presented at the 6th Secure Knowledge Management Conference at Dubai, UAE, December 2014 and a PhD research paper has been presented at the 10th International Conference on Cyber Warfare and Security ICCWS-2015 at Kruger National Park, South Africa, March 2015.

References

Bajikar, S. (2002). Trusted Platform Module (TPM) based security on notebook PCs. White Paper, Mobile Platforms Group, Intel Corporation.

Brewer, R. (2014). Advanced persistent threats: minimizing the damage. *Network Security, 4*, 5–9.

Cohen, F., Lambert, D., Preston, C., Berry, N., Stewart, C., & Thomas, E. (2001). A framework for deception. Computers and Security (IFIP-TC11), pp. 3–40.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longstaff, T. A., & Mead, N. R. (1999). Survivability: protecting your critical systems. *IEEE Internet Computing, 3*, 55–63.

Gragido, W., & Pirc, J. (2011). *Cyber Crime and Espionage: Seven Commonalities of Subversive Multi-vector Threats*. Elsevier.

Kamhoua, C., Kwiat, K. A., Chatterjee, M., Park, J. S., & Hurley, P. (2013). Replication and diversity for survivability in cyberspace: a game theoretic approach. In Proceedings of the 8th International Conference on Information Warfare and Security (ICIW), pp. 116.

Kim, G. H., & Spafford, E. H. (1994). The design and implementation of tripwire: a file system integrity checker. In Proc. Second ACM Conference on Computer and Communications Security, pp. 18–29.

Mcwhorter, D. (2013). APT1: exposing one of China's cyber espionage units. Mandiant Corporation.

Mehresh, R., & Upadhyaya, S. (2012). A deception framework for survivability against next generation cyber attacks. International Conference on Security and Management (SAM).

Mehresh, R., Upadhyaya, S., & Kwiat, K. A. (2010). A multi-step simulation approach toward secure fault tolerant system evaluation. International Workshop on Dependable Network Computing and Mobile Systems (DNCMS) in conjunction with IEEE Symposium on Reliable Distributed Systems, pp. 363–367.

Mehresh, R., Rao, J. J., Upadhyaya, S., Natarajan, S., & Kwiat, K. A. (2011). Tamper-resistant monitoring for securing multi-core environments. International Conference on Security and Management (SAM).

Mehresh, R., Upadhyaya, S., & Kwiat, K. (2012). Secure proactive recovery - a hardware based mission assurance scheme. *Journal of Network Forensics, 3*(32–48), 2011.

Repik, K. A. (2008). Defeating adversary network intelligence efforts with active cyber defense techniques. No. AFIT/ICW/ENG/08-11, Grad. School of Eng. and Management, Air Force Inst. of Tech.

Schryen, G., Volkamer, M., Ries, S., & Habib, S. M. (2011). A formal approach towards measuring trust in distributed system. In Proceedings of the ACM Symposium on Applied Computing, pp. 1739–1745.

Trusted Computing Group Incorporated. (2007). TCG Software Stack (TSS) Specification version 1.2.

Yuill, J., Denning, D., & Feer, F. (2006). Using deception to hide things from hackers: processes, principles, and techniques. *Journal of Information Warfare*, 26–40.

Ruchika Mehresh received her doctorate degree in Computer Science and Engineering from State University of New York at Buffalo, 2013. She has several publications in international conferences and her research interests are security and mission survivability in distributed systems. She currently works as a software engineer in the security group of Core Technology Division at EMC Corporation.

Shambhu Upadhyaya is a professor of computer science and engineering at the State University of New York at Buffalo since 1986 where he also directs the Center of Excellence in Information Systems Assurance Research and Education (CEISARE), designated by the National Security Agency and the Department of Homeland Security. His research interests are information assurance, computer security and fault tolerant computing.