Securing a System-of-Systems From Component Misbehaviour

Nathan Shone, Qi Shi, Madjid Merabti, Kashif Kifayat
School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, UK
n.shone@2007.ljmu.ac.uk, {q.shi, m.merabti, k.kifayat}@ljmu.ac.uk

Abstract—Securing distributed and decentralised complex networks such as System-of-Systems (SoS) is a difficult task. The least secure part of a SoS is the components, primarily because inter-component trust is vulnerable to abuse. Component misuse and misbehaviour is a major security threat to these networks but is often overlooked. Most existing security monitoring techniques and solutions are largely ineffective when applied to a SoS. This paper highlights the problem of component misbehaviour and misuse in a SoS and examines the applicability of existing security monitoring techniques from similar research areas. It also details our proposed initial solution, which aims to provide a method to ensure the security and integrity of Directed SoS compositions by detecting component misbehaviour.

Index Terms—System-of-Systems, SoS, Monitoring, Component Misbehaviour

I. INTRODUCTION

NETWORK security of complex systems is becoming a prominent issue following the growing concerns over critical infrastructure security and the threat of cyber-warfare. Recent high profile breaches and theft of personal data have fuelled such concerns [1] [2]. The question being posed is whether existing security is sufficient for modern computing concepts and technologies. The increased demand for online services is seeing a considerable growth in the number of new online services, with varying levels of implemented security. The increasing complexity found in modern networks, helps in some part to explain the vast scale of the challenge faced in securing existing systems. This poses even more concerns over the security of future and emerging systems, such as System-of-Systems (SoS). In such complex environments, component integration and high levels of security are often essential. The destructive and malicious capability of rogue or misbehaving components is often overlooked, despite being one of the most serious problems faced in such environments.

Most security solutions are incapable or ineffective when detecting component misbehaviour in dynamic and decentralised environments such as a SoS. It is for these reasons that SoS will become more reliant on monitoring for security. This paper will examine our novel initial proposed solution called SSC. It will offer real-time SoS security monitoring, which is capable of maintaining compositional security. This is achieved by using our novel approach to monitor for internal misbehaviour of SoS components, thus facilitating the detection of any abnormal or unwanted behaviour. This enables the SoS to be defended and potential damage to be limited by detecting misbehaviour as it happens. The main novel contributions of this research are in the monitoring methodology, the decision making algorithm and the threshold calculation algorithm.

In Section II, this paper will provide background information on SoS concepts and the problems that currently exist. In Section III, related work is examined and problems in relation to the application within SoS security monitoring are outlined. Section IV examines the proposed solution and its underlying ideas and methodologies. Whilst Section V will focus on the implementation of the solution and the paper is concluded in Section VI.

II. BACKGROUND

The concept of a SoS is to address issues such as efficiency, flexibility and integration, which arise from the increasing complexity found in modern network infrastructures. It involves integrating multiple independent complex systems, to form a large-scale distributed system. The resulting composite system is capable of reducing complexity for users, pooling system resources and achieving greater levels of interoperability. Ultimately, it can achieve levels of functionality that would be difficult or even impossible to replicate on an individual system. A SoS is a dynamic virtual entity, existing only by the continued collaboration of its component systems, similar in many respects to a peer-to-peer network. Components connect together via an interface [3] which enables collaboration and the sharing of services.

SoS is receiving interest from multiple research areas and has also been linked with possible applications in military [4], aerospace and healthcare environments [5]. There are three main classifications of an SoS as outlined by Maier [3], these are Collaborative, Virtual and Directed. Directed SoSs normally have a specific purpose and their membership is controlled. Whereas Collaborative and Virtual are more dynamic and not as much control is retained.

A SoS is a decentralised system, hence it has little in the way of regulation or administration, resulting in each component being responsible for its own behaviour and administration. It is important to understand that although component systems exist as a single entity, they effectively have a dual existence. As they belong to an organisation, their primary allegiance lies with that organisation but they also contribute to the SoS and in some cases this becomes their main function. Component systems contribute to the SoS on a voluntary basis [6], there is no SLA therefore contribution is not always
equivocal or even possible, resulting in dynamic compositions. However, this architecture does aid the speed of deployment, scalability and the development of emerging behaviours [7]. Each component system is responsible for its own behaviour and actions, thus the behaviour and responses cannot be accurately predicted. This means that monitoring is required as irrational behaviour cannot be effectively controlled; this could jeopardise the security or integrity of the system.

The decentralised nature of a SoS places heavy reliance upon inter-component trust. Components trust each other to behave appropriately and to provide efficient and reliable services. We must also assume that this trust can and will be abused. One of the largest threats to a SoS is actually the components themselves. As they are self-regulating, there is nothing to detect misbehaviour or malicious behaviour. The SoS infrastructure is largely based on trust and if this is broken, misbehaving components can cause damage on a catastrophic scale. Components have no contractual obligation to continue working in an environment where misbehaviour may threaten them. In the worst case scenario this could lead to a total collapse of the SoS.

The term misbehaviour is a rather ambiguous term when used in computing. In this paper it is interpreted as any behaviour or action exhibited by a component, which may negatively impact on other components or the SoS as a whole. Examples of this would be: providing bad or false data through services, leeching resources from other components, stealing sensitive information, deliberately consuming too many resources of one or more components, sabotaging communications or data and trying to discredit other honest components.

Component misbehaviour is very difficult to detect, particularly in dynamic environments as behaviour is ad-hoc and random. Therefore, there is not always a defined behavioural pattern or signature to observe and there is no centralised administration. This challenging environment means that the only feasible method for detecting misbehaviour is with capable security monitoring.

Some of the possible implementations of a SoS require high levels of security, such as aerospace, finance and healthcare industries. This reinforces the point that this issue is a high priority and a complex challenge; mainly due to the dynamic, ad-hoc and unpredictable environment in which they exist. A misconceived view is that securing a SoS would be relatively simple. Although each component system may be considered secure and integrated in a secure manner, this does not guarantee a secure composition [8]. Securing a SoS involves securing the components, the SoS, constant monitoring and adjustment to ensure the desired level of security is upheld, as the system is liable to change [9]. A SoS can be protected from external threats through the use of a combination of different systems such as Firewall, IDS, IPS and UTM. However, difficulties arise when protecting from internal based threats, whereby compromised components can negatively affect the SoS.

The unusual nature of a SoS such as being decentralised, ad-hoc, dynamic and unpredictable means that security monitoring is the most efficient method to ensure the security of a SoS. Many existing monitoring methods are ineffective when applied to a SoS, particularly those utilising pre-defined behavioural patterns or signatures. This is due to the dynamicity and unpredictability encountered in such environments.

III. SoS Security Concerns

Within the SoS domain, current research efforts are predominantly focused on differentiating between traditional systems, creation, management and maintenance. The majority of existing work is theoretical with few offering any practical solutions. A number of papers do consider SoS security but these are also largely theoretically based. The SoS environment poses many security challenges but currently there is limited research into SoS security monitoring.

There are some proposed SoS security frameworks which are based on either semantics or trust management. These papers predominantly focus on establishing security within a SoS. In [8] the proposed framework combines context-aware access control with trust management and ontology-based services. This framework focuses on several aspects of SoS security but it fails to address the problem of misbehaving components or providing adequate real-time protection.

In [10] the author’s proposition is that the traditional approach to security in a static environment is insufficient for dynamic and uncertain environments. The paper proposes using a macroscopic security schema, whereby security schemes are created for collections of systems rather than individual systems. This can result in security that allows a system to monitor environment and feed results back to the system to allow for adaptation and alter its position. However, this system does not account for any internal based threats such as component misuse or misbehaviour, which may lead to inaccurate system adaptations.

As SoS security monitoring is a relatively new area, it is necessary to examine areas which have similar aims, requirements and techniques.

A. Host Intrusion Prevention System (HIPS)

Host Intrusion Prevention System (HIPS) is a similar research area, in that it monitors local hosts for signs of malicious or abnormal behaviour. Although the general ideas are similar, HIPS are designed to protect relatively static systems, whereas the aim of this paper is to protect the SoS composition itself.

Some host based systems are user centric, in that they concentrate on monitoring patterns of user behaviour [11], [12]. This is particularly so on *NIX systems, as different processes are run as different users. The main problems when applying this approach to a SoS is that component behaviour is largely unknown due to the dynamicity of tasks undertaken, so user behaviour would also be dynamic and thus difficult to profile into accurate behavioural patterns.

Some existing solutions use policy or description language based approaches, in particular [13], [14] and [15]. BlueBoX [13] is a policy-based solution and aims to define security boundaries more clearly. Its policy rules are based on the understanding of system and software behaviour. It allows the specification of rules relating to system resource consumption.
by particular programs. Although this method would be efficient for detecting novel attacks, it is not a scalable approach. This method would cause issues if a software update caused behavioural change or if a piece of software was used in a different way. When applied to a SoS, the static behaviour this approach uses cannot be relied upon with any accuracy due to the dynamicity of a SoS through both compositional (Collaborative SoS) and system behaviour. This method may also be considered impractical, particularly on larger systems, considering the amount of software that can be installed on any single machine.

Many host based monitoring solutions rely on the use of log files as a source of data to analyse, such as [14]. In [14], the author proposes using pattern matching and neural networks to effectively analyse log files to detect intrusions. Unfortunately, the use of log files does not provide real-time protection, as logs are normally written after the event. If this technique was applied to a SoS environment it would certainly lead to problems. One of the requirements for this project is to monitor in real-time to prevent the realisation of threats, which this technique would not facilitate. There is also the problem that not all system events are recorded in log files so it may be difficult to ascertain all intrusions. It is also possible to intercept system log calls and prevent the writing of an event to a log file.

Detection in host based monitoring systems such as [14], [16] and [17] utilise operating system resource utilisation monitoring. This method monitors for abnormal usage of system resources (such as RAM, processing power and network bandwidth) based on a pre-defined ‘norm’ behaviour profile. As outlined previously, a SoS is very dynamic both in composition (mainly collaborative SoSs) and system behaviour, this would render any “norm” profile largely ineffective. This method does however facilitate the detection of novel threats.

Consequently, most existing HIPS methods are largely ineffective for monitoring SoS, mainly due to the lack of real-time support, inability to monitor dynamic environments and scalability issues.

B. Peer-to-peer (P2P) monitoring

The lack of centrality and administration in P2P networks leads to complications similar to those faced by SoSs. The dynamic, ad-hoc and decentralised nature make security a difficult task in both P2P and SoSs. P2P security frameworks are similarly concentrated on protecting the host system. There are many security frameworks proposed for P2P networks for differing reasons, this paper will examine some that are related to the area of research.

Jin et al [18] propose a method for detecting malicious nodes in a P2P network. Each node is monitored by its neighbours and observations are collected to compute a reputation score for the particular node. Nodes with malicious actions can then be detected. The ideas outlined in the paper are interesting but when applied to a SoS there would be complications. A SoS is a composition of heterogeneous components from different contributors. Each contributor would have their own perception of malicious actions. As the SoS is decentralised it would be difficult to create, maintain and enforce universally accepted definitions of malicious actions. In addition, interactions with the malicious node is required prior to being identified as malicious therefore protection is not provided in real-time. Fang et al [19] also propose a similar solution for detecting malicious nodes. The use of reputation based monitoring is a suitable method for some situations. However, to be efficient in a SoS environment, all components must be equivocally trusted to report honest observations. Otherwise, this would require the use of a centralised server, which becomes a single point of failure and an issue for scalability.

Visan et al [20] propose a similar approach but suggest the use of decentralised trust management to rank participating systems and moderate interactions between them. The system hierarchy would be organised entirely by using the participant rank. The paper proposes a protocol that allows maintenance and accessing of trust information. It is also suggested that peers that compute trust values of other peers are offered anonymity. Trust is a core component of any decentralised network; however, this proposal would encounter several problems when applied in a SoS environment. A SoS should be considered a secure environment, and the risk of compromise whilst the trust level is calculated and the component identified as malicious may be considered too great. The anonymity provided to trust calculating components provides the potential for abuse and poses the question of whether one negative calculation can be accurately indicative of component’s behaviour.

Graffi et al [21] propose a monitoring and management framework for P2P networks; the primary objective is to ensure quality not security but the methods used are still applicable. The author proposes using a statistical representation of the entire network. The quality of the network at certain times is pre-defined statistically and if it falls below this defined quality, a process is initiated to automatically adapt the system configuration. This is similar in some respects to the proposed solution in this paper. However, the effectiveness of this method is reliant on the frequency of the sampling rates, if the rate is too infrequent it may be inaccurate and if it is too frequent this may create unnecessary system load. The proposed automatic adaptation of system configuration in a SoS environment may prove to be counterproductive. In terms of security, falling below a threshold in such a dynamic environment does not necessarily indicate the need to change the system configuration. Random or even erratic behavioural changes can be expected, but if this is a genuine security concern, re-configuration will only help to conceal the problem not solve it. Obviously constant refinement and adjustment is necessary when monitoring in such an environment.

An idea proposed by Rice [22] is the use of incentives to promote the growth of non-malicious P2P networks. This model charges nodes higher prices for choosing “bad” links; these are links that decrease the Pearson coefficient and ultimately degrade the networks resistance to malicious propagation. This is an interesting concept but when applied to a SoS it goes against the principles of a SoS. Components are meant to share what resources they can and contribute
to the collaboration. However, this technique may create prejudice amongst components. This would result in unwanted incentives, which can lead to load instability, a decrease in collaboration and decreased efficiency.

IV. METHODOLOGY

This section outlines SSC, which is our initial proposed solution for monitoring for component misbehaviour in SoSs. It is designed to maintain the security of a SoS composition by monitoring for behavioural abnormalities or component misbehaviour, thus limiting any adverse effects on the SoS. SSC is primarily concerned with the security of the SoS, therefore any action taken is solely in the interest of SoS security. It is designed to ensure the compositional security of a SoS which relies on inter-component trust to secure it from harm. SoSs are very effective for mass distributed collaboration and resource sharing but as with all systems, they are vulnerable to exploitation. SSC is primarily designed for protecting Directed SoSs, which have a set purpose, and membership is often controlled, unlike the Collaborative and Virtual SoSs.

SSC is a host-based solution rather than a network based and is installed on each component. This way it is minimally intrusive but allows for the scalability and dynamics required. Some may argue that host-based solutions are too resource intensive [23]. However, with the abilities of modern quad core processors and hyper-threading technology, this is not such an issue as it once was. SSC is designed to be a lightweight and low resource-consuming framework that shares a common goal, which is to protect the SoS irrespective of what system it is installed on.

A. SSC Overview

SSC is based on the idea that it is possible to detect component misuse or misbehaviour by monitoring for abnormalities in resource usage and changes in system security.

SoS components exist as independent systems and therefore have their own roles and related tasks to perform outside the SoS. Each task performed by the system has its own unique optimal resource usage and system security footprint. Therefore, knowledge of the resource usage and security changes would provide a baseline of what could be considered as “normal” behaviour. Hence, a system snapshot (S²T) is created, which involves the monitoring and logging of certain key metrics at pre-defined intervals for a period of twenty-four hours, which is then repeated for seven days. This will provide detailed information of the system behaviour prior to joining a SoS. The S²T information covers the duration of 7 days, thus obtaining sufficient data to account for any scheduled off-peak activities such as updates.

When components join a Directed SoS, system owners know how much of their system they wish to contribute. SSC requires owners to describe levels of contribution, limitations and restrictions concerning the SoS in an XML based file (S³LA).

By combining a weekly behavioural average (calculated using data from the S²T) and contribution and limitations from the S³LA, it is possible to calculate theoretical “normal behaviour”. We are developing a novel algorithm, which can interpret this data to calculate “normal behaviour” thresholds, threshold tolerances and metric weightings. These can be utilised to monitor for abnormalities in order to detect component misuse or misbehaviour.

Due to the complexity of SoSs, decisions regarding suspected misuse or misbehaviour cannot be based on one piece of evidence. Instead, a decision algorithm must be utilised that considers a more diverse collection of relevant evidence, as in this environment one source of evidence is not sufficient to be fully indicative of the truth. We are also developing a novel decision algorithm, which is triggered when the normal behaviour threshold and tolerances are exceeded. The algorithm considers evidence directly related to the triggering metric, along with metric weightings to calculate a level of certainty regarding the legitimacy of the event. This certainty level is based on a 0 – 1 scale, which is then used to implement a pre-defined action. The higher the certainty level is, the less serious the pre-defined action.

B. SSC Design

![SSC Framework](Image)

Figure 1: SSC Framework

1) This module is used to create a System-of-Systems Snapshot (S²T), which is a profile of system behaviour, resource usage and security changes. The component system is profiled by this module prior to joining a SoS. The usage of selected resources, system behaviour and security changes are monitored for a period of twenty-four hours and repeated for seven days. This provides knowledge of resource usage and the behaviour required for a component’s normal activities (non-SoS). This data is used to create a S²T which is utilised by other parts of the framework.

2) This module creates a Simple System-of-System Service Level Agreement (S³LA) which is created prior to joining the SoS. The S³LA is not an enforceable or contractual SLA, instead it is a method of accurately defining the contribution, restrictions, limitations or known weaknesses of the component.
3) This module contains the novel decision algorithm which calculates the optimal thresholds, threshold tolerance, metric weightings, exceptions and statistical information. The thresholds and tolerances are calculated using the behavioural profile in the $S^2T$ and the contribution definitions in the $S^3LA$. Each metric that is monitored on the component has a different influence on the behaviour of others. It is often possible to identify a problem by examining metrics that are highly influential on each other. As our decision algorithm takes all related metrics into account, it is important to calculate the metric weightings so as to take into account the relationship between them. Weightings are a numerical representation of how influential metrics are on each other and how reliable they would be as an indicator to a problem. If any of the behaviour passes through the threshold tolerances, then events are correlated using the $S^2T$ and the audit log. This allows for the creation of exceptions or the adjustment of the threshold tolerances. Other statistical information can be obtained such as the theoretical maximum rate of change and durations of maximum and minimum resource consumption.

4) The monitoring daemon is a constantly running process that monitors the selected metrics on the live system against the previously defined “normal” expected behaviour. If any deviations are detected then these are reported to the next module.

5) The decision module handles all reported events from the previous module. The novel decision algorithm is implemented which calculates a level of confidence regarding the legitimacy of the event. Depending on the type of event reported, a certain set of evidence is gathered from specified metrics. The decision algorithm uses a combination of data fusion, decision tree and statistical analysis to formulate a decision. This enables the combination of multiple source of evidence in order to formulate a more accurate and reliable level of certainty. Higher certainty of a legitimate event is reflected by a higher returned confidence level value.

6) The confidence level from the previous step is interpreted by the module and an action is implemented. These actions are pre-defined by the user; the lower the confidence level the more severe the actions are.

7) This module is utilises an embedded webserver to provide a web based administration interface for managing the framework. This can be used to start and stop the framework as well as joining and leaving SoSs.

The initial proposed solution will monitor the following metrics on the component system (this list is not exhaustive and subject to change): RAM, SWAP, CPU, bandwidth, processes, ports, permission changes, user account usage and privilege changes.

V. IMPLEMENTATION

The SSC framework is implemented on Linux and written entirely in C, as it offers better integration with lower level operating system functions, greater speed and lower overheads. SSC needs to be able to monitor and perform in-real-time so speed is an essential characteristic. Resource metric monitoring will be conducted using mostly the /proc virtual Linux file system and security metrics will be conducted using a custom Linux Security Module (LSM). As SoS is largely still a theoretical area, there is no true representative SoS framework which can be used to test the developed framework. Instead, we are using web services as an abstraction of a SoS interface thus allowing realistic interaction between components [24]. SSC is deployed on our Linux test-bed, which is illustrated in Figure 2.

Due to the nature of this research, it is possible that some of the experiments conducted may cause loss of service, data or system damage. To resolve this issue, the test bed will be implemented using virtual Linux machines running on a virtual network. This provides an isolated working environment where external interference cannot influence the results. This has significant advantages over using physical machines such as: virtual machines are easily configurable, easy to clone, easy to restore to a working state and can be easily replicated to test scalability.

Initially, we will be using web services installed on each component to simulate inter-component resource, security and system behaviour. Scripts (WSM Replay Scripts) launched from the controller will apply simulated dynamic load onto the components. This will allow us to test SSC’s operation whilst in a constantly dynamic environment. We are also using scripts installed on each component to simulate component misbehaviour. We will be gauging the success of the solution by assessing how quickly and accurately misbehaviour is detected, and whether the requirements and objectives of the research are met.

Unfortunately, at the time of writing this paper, implementation of the framework is still on-going and as a result, we do not have sufficient results to include in this paper.
VI. Conclusion

This paper has highlighted that most existing research into SoS security is largely theoretical and lacking any practical solutions and that work on SoS monitoring is very sparse. It has outlined how most existing monitoring solutions are largely unsuitable when applied to monitor such a dynamic environment such as a SoS, outlining particular problems that may be faced.

This paper has also detailed the design of our proposed novel initial framework (SSC) which offers protection to SoS compositions by monitoring for component misuse or misbehaviour. Currently we do not know of any solutions which adopt this approach to monitoring for misbehaving components to protect SoS compositions. The paper has outlined the main novel contributions of the research, mainly our design methodology, decision algorithm and threshold calculation algorithm.

Our immediate future work will primarily focus upon completing our implementation, initial testing and refining both the framework and the algorithms. This will enable us to obtain the important preliminary results. Ultimately, the long term objective for this research project is to produce a lightweight monitoring framework, which will ensure the security and integrity of Directed SoS compositions.

REFERENCES


