

Towards Survivable Intrusion Detection System

Dong Yu, Deborah Frincke

Center for Secure and Dependable Software, University of Idaho
dongyu@csds.uidaho.edu, frincke@cs.uidaho.edu

Abstract

Intrusion detection systems (IDS) are increasingly a key part of system defense, often operating under a high level of privilege to achieve their purposes. Therefore, the ability of an IDS to withstand attack is important in a production system.

In this paper, we address the issue of survivable IDS. We begin by categorizing potential vulnerabilities in a generic IDS and classifying methods used to enhance IDS survivability. We then propose an efficient fault tolerance based Survivable IDS (SIDS) along with a systematic way to transform an original IDS architecture into this survivable architecture. Key components of SIDS include: a dual-functionality forward-ahead (DFFA) structure, backup communication paths, component recycling, system reconfiguration, and an anomaly detector. Use of the SIDS transformation should result in an improvement in IDS survivability at low cost.

1. Introduction

Intrusion detection systems (IDS) originated as a mechanism for managing the detection of system misuse through the analysis of activity [20]. Modern IDS extend this concept through the full range of system defense: gathering data used to identify misuse, analyzing the data, responding to data, and more recently as part of the forensics analysis for past attacks [1-3]. IDS may act only on an individual host, within a network of hosts, within an enterprise, or between enterprises [19]. They are found both “inside” and “outside” perimeter defenses such as firewalls [21].

As with any security mechanism, this increased level of responsibility comes with a danger. Many IDS operate under a high level of privilege, particularly when they are directly involved in gathering sensitive security-related data and when managing responses to attack. Because of their important role in system

defense, and also because of these additional privileges, sophisticated system attackers may attempt to disable an IDS [1,4], or take over (or influence) its capabilities so that the system’s IDS becomes part of the **attacker’s** own arsenal. Thus, the ability of the IDS to withstand attack is an increasingly important part of a production system [1].

To date, although security and survivability is a key part of what makes an IDS useful [1], most research has emphasized the contributions an IDS can make to system defenses rather than what should be done to make the IDS itself more defensible. In this paper, we focus upon the security, reliability, and survivability of the IDS itself. We begin by identifying some vulnerabilities in a generic IDS structure and list several attacks on the generic IDS. We then propose a model that will better protect the IDS from some of these attacks, and a systematic way to transform existing IDS into that architecture.

IDS architectures are highly varied. Researchers have implemented their key components of data gathering, analysis, and response across the gamut of centralized, decentralized, coordinated, and independent. There are now IDS that range from individual host-based systems that gather and analyze all data locally, up through network-based or even cross-enterprise collaborative systems such as [14,19,22] that perform both local and distributed data gathering, analysis, and response. Although these systems all have different requirements, the core functionality of an IDS may be illustrated with the architecture described in Figure 1, only slightly different from what Denning originally proposed [29].

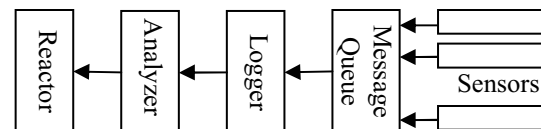


Figure 1: Structure of typical IDS

Even highly distributed systems such as [23] fit within the context of Figure 1; the activities may be

distributed across multiple hosts or combined in alternate ways.

Analysis of Figure 1 yields several potential vulnerabilities. These vulnerabilities can be divided into the categories shown in Table 1. These categories are based both on vulnerabilities documented by other researchers [1,4,8] and those we postulate based on analysis of Figure 1. Note that this list is potential not actual -- not all these vulnerabilities exist in all IDS since the architectures of some [5-9,11,12,14,22,23] either were deliberately designed or “naturally” mitigate some vulnerabilities listed.

The vulnerabilities listed in table 1 may be exploited in several ways as indicated below; this is not a comprehensive list.

- V1: Attackers may seek to shut down or slow individual primary components. For example, an opponent might: shut down or delay sensors most likely to pick up his/her activity; eliminate the response function so as to avoid ejection or retribution or more detailed scrutiny; an opponent might shut down or slow analysis and thus slow down discovery.
- V2: Attackers may conduct targeted DOS attacks to stop the service of the IDS overall – all functionalities from data gathering to analysis to response to reporting -- and so the intrusions are not observed and reported. This is related to V1 but emphasizes the interference with the ability of components to work together in a timely way rather than disruption of individual component functionality.
- V3: Attackers may corrupt an analyzer through modification of the “rules” or “models” used to identify misuse and/or anomalies. This can occur a priori – as when a long-term insider succeeds in “training” an anomaly-based system to “recognize” misuse behavior as normal. This may also occur by direct changing of rules within the analyzer itself (for instance, changing a specific misuse pattern). The result here may be anything from a false positive to a false negative when the analyzer fails to properly categorize sensor data. Similarly, a reactor’s rules may be modified so that, for instance, it fails to respond appropriately.
- V4: We consider the communication channel to include all activities involved in moving information from one component to another. For instance, in a distributed system this might involve taking data from a local database, transmitting it across a network, storing it on the (new) local host, and then providing it to the next component for utilization. Attackers may tamper with the communication channel itself, or with the data

while it resides in storage; in either event, components may not have the correct data presented to them for processing.

- V5: Attackers may blind, corrupt, or delay sensors so that data is either not provided for analysis, inaccurate data is provided, or accurate data is provided “too late” to detect misuse in progress. This is related to V1.
- V6: Attackers may bypass the detection mechanism by operating at a level lower than that which auditing occurs. This is not so much a deliberate attack on the IDS as it is making use of the IDS characteristic of “granularity” of observation.
- V7: Attackers may perform Insertion and Evasion attacks [4] to fool the IDS. For example, the attacker may send a series of packets to hack a system. The combination of all these packets (which is what observed by the sensors) is valid. However, the packets are sent in a way that only part of them actually reach the target. The combination of the packets reach the target is an attack.

Target	Vulnerabilities
<p>Component Failure</p> <p>A major component fails to perform to specification.</p>	<p>V1: Each individual component is centralized and vulnerable to single point of failure¹.</p> <p>V2: The IDS is vulnerable to denial-of-service attacks [1].</p>
<p>Component Corruption</p> <p>Components are (deliberately) corrupted</p>	<p>V3: Components are caused to behave erroneously, perhaps making inaccurate reports or coming to inaccurate decisions.</p>
<p>Communication Failure</p> <p>Messages are not received accurately and/or in a timely way.</p>	<p>V4: The communications between components may be broken.</p>
<p>Sensor Failure</p> <p>Sensors fail to provide accurate observations of all relevant conditions.</p>	<p>V5: Sensors are vulnerable to corruptions and crashing.</p> <p>V6: Sensor auditing may be bypassed.</p> <p>V7: Data collected by sensors are vulnerable to Insertion and Evasion attacks [4].</p>

Table 1: Vulnerabilities

The remainder of the paper is as follows. In Section 2, we survey current approaches used to enhance the survivability of the IDS and classify them as: access

¹ Some of the components (such as reactor) are centralized even in distributed IDS or agent based IDS.

control strengthening, detection and identification obstructing, and fault tolerance through redundancy. In Section 3, we propose a Survivable Intrusion Detection System (SIDS) architecture, which serves as additional protection by assuring the uninterrupted correct behavior of the IDS. Specifically, we introduce the dual-functionality forward ahead (DFFA) structure, which serves as the basic architecture in the SIDS, and present a cost-based failure detection algorithm used in the SIDS. Finally, we discuss the strengths and weaknesses of the model, and indicate our future work.

2. Related Work

We divide existing approaches to IDS vulnerabilities into three categories: access control strengthening; detection and identification obstructing; and fault tolerance through redundancy. Each is described in the following sections.

2.1. Access Control Strengthening

Access control strengthening methods, as the name indicates, seek to improve IDS survivability through use of strong(er) access control methods. As mentioned in Section 1, correct operation of an IDS involves interactions between several components, usage of sensor data collected from different locations, and interaction between the IDS and the OS. Controlling which components or processes are allowed to communicate with other components in the IDS, as well as which data can be trusted by the IDS is very important to mitigate several vulnerabilities indicated in Table 1, and other researchers have proposed a variety of ways to accomplish this.

Onabuta et. al. [5] proposed an IDS protection mechanism based on the Low Water-Mark mandatory access control with the addition of a limited access mode. The main purpose is to restrict the access to the IDS in hopes this will help prevent attackers from gaining direct access.

Evans and Frincke [14] use a bitmap-based access control list in the early stages of the Hummer project to control which sites (hosts and/or networks) are trustable at what level. A more complex model involving trust, integrity, and cooperation is introduced in [19].

2.2. Obstructing Methods

Obstructing and obscuring methods, as applied from the perspective of the IDS, are intended to make it more difficult for an attacker to notice the presence of an IDS. This form of “security by obscurity” is a

delaying mechanism only, not useful against savvy insiders or wary attackers, but delaying mechanisms can be quite useful in many other situations. Although it should always be assumed that a determined attacker (or insider) working to gain control of a specific target will eventually be able to determine the precise IDS that is present, not all attacks are of this nature. For example, many modern “attacks” are pre-programmed to run on a broad scale without a specific target in mind. Even those attacks that are (or that will be) enhanced by automated code looking for typical IDS traces may well miss the presence of an IDS properly employing obscuring methods. Obscurity can be provided by many means – for instance, either by hiding as many aspects of an IDS as possible, or by creating the “false” appearance of one form of IDS in order to prevent an attacker from discovering the actual IDS in place.²

Bace and Mell [6] indicate the need for silent, reliable monitoring of attackers. If, for instance, an IDS broadcasts alarms and alerts in plaintext over the monitored network, careful attackers or well-written automated attack code can easily detect the presence of the IDS and perhaps even discern the particular IDS in operation. The suggestion provided in [6] is to use encrypted tunnels or other cryptographic measures. The intent is to hide and authenticate IDS communications to secure and ensure the reliability of the IDS. This approach would be especially useful where there are other activities that also employ encryption, so that the mere presence of encrypted traffic does not itself become a red flag.

Bhattacharya and Ye proposed a “random hopping technique” [7]. The strategy is to remotely install and dynamically launch an IDS agent at a specific site for a given duration. This IDS agent randomly hops over a set of sites, so that no single site is vulnerable for a prolonged period. Attackers might compromise such a system by attacking all sites or a selected set of sites at the same time.

Takada and Koike propose a similar method to “random hopping” in the context of log file protection in NIGELOG [8]. The strategy is to increase the difficulty for an attacker who seeks to corrupt log files by making multiple copies and placing those copies in arbitrary locations, and then periodically moving them around.

² In the extreme situation, fake IDS “traffic” might serve the same purpose as a “Fake security camera” outside a store – attackers looking for easy access may choose to avoid a system that seems to have an IDS. However most attacks are not currently of this nature and so it is doubtful that this approach would be very useful in the modern environment. As automated attacks increase their use of stealth technology and probes, such techniques may become more cost effective.

Wang and Knight propose obscuring of the IDS code itself [11]. The method they describe is to obstruct program analysis with control-flow transformation and data-flow transformations.

2.3. Fault Tolerance through Redundancy

A third approach is the use of fault tolerance, through means such as redundant agents or reconfiguration. These methods primarily seek to mitigate results of a successful attack rather than preventing the attack from occurring.

The CERIAS researchers, Zamboni, Spafford, Balasubramaniyan, have written several articles regarding the AAFID architecture (Autonomous Agent For Intrusion Detection) [9,24], which makes good use of fault tolerance through redundancy. Agents both collect and analyze information independently. This prevents the single point of failure in the above architecture. These agents are organized into layers, further enhancing the scalability of the IDS, and making it more likely to successfully tolerate Denial of Service attacks. Furthermore, each agent only collects data from a single host and hence insertion and evasion attacks as in [4] are more difficult. However, although these agents run independently, they are identical instances of the same code base and thus subject to same malicious attacks. The assumption that those agents will only fail independently does not hold under malicious attacks specifically targeting the code. This is an argument for using heterogeneous code, rather than homogeneous agents. Use of agents in general also introduces the aspect of coordinating them – another area that could potentially be attacked by slowing or disrupting communication.

Cheung [12] proposes a reconfiguration approach to identify misbehaving components. The strategy is to reconfigure the system to avoid using components that are diagnosed as misbehaving. Cheung claims that with this approach, the system only need to invoke these relatively expensive measures when the system is under attack. This approach would not be as useful when the target causes damage that is irrecoverable such as leakage of confidential information, but could be quite useful in the face of a broad-based attack on availability.

We summarize these categories and the associated vulnerabilities in Table 2.

3. Survivable Intrusion Detection System

In this section, we propose SIDS, an architecture suitable for a Survivable Intrusion Detection System, and introduce a way to transform an existing IDS into

SIDS. We begin by defining our goals. A survivable IDS "...must be able to recover from system crashes, either accidental or caused by malicious activity. Upon startup, the IDS must be able to recover its previous state and resume its operation unaffected." [9]. We have chosen to address this issue by requiring that detection mechanisms be built into the IDS itself. These are intended to detect effects such as corruption and crashing. If the IDS is in a failure state, the intent is for the IDS to automatically recycle and restart the corrupted components or reconfigure the communication paths. In this section, we will describe the DFFA structure, and the cost based detection algorithm proposed for use in the SIDS.

Categories of IDS Secure Approaches	Vulnerabilities the Approaches Targeting
Access Control Strengthening	V2, V3, V4, V5, V6, V7
Detection and Identification Obstructing	V1, V3, V4, V5, V7
Fault Tolerance through Redundancy	V1, V2, V3, V4, V5, V7

Table 2: Categories of IDS Secure Approaches and Vulnerabilities

3.1. The DFFA Structure

The SIDS architecture is based on a DFFA structure intended to mitigate vulnerabilities V1-4 of Table 1. An IDS is translated into SIDS as follows. We begin by assuming that the original system can be separated into components $\bar{C}_1, \bar{C}_2, \dots, \bar{C}_N$ each having only one function (so, if we were dividing an agent-based system that relies on individual agents to perform data gathering and analysis, each agent would be comprised of at least two components). We label component functionality of each component as F_1, F_2, \dots, F_N respectively. We indicate this pairing between the functionality F_j of component \bar{C}_i as (\bar{C}_i, F_j) .

Figure 2 depicts the original structure of a four component IDS. We may think of these components as straightforward implementations of the generic diagram Figure 1, clearly subject to single point failure of components and connection. Elimination of such single point of failure has been discussed in fault tolerant IDS architectures such as [9,12].

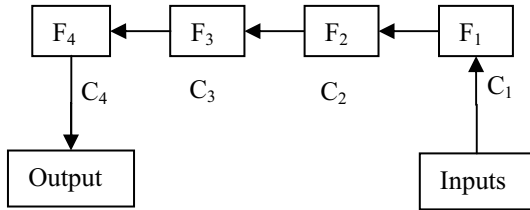


Figure 2: Original structure of a four component system

Each component \bar{C}_n from the original system originally corresponded to a single functionality F_n . In the DFFA, this is modified so that each component now contains *two* functionalities, and each of these is implemented in consecutive components. C_n contains functionalities F_n and $F_{R(n+1)}$ where $R(n) = ((n-1) \bmod N) + 1$.

For example, C_1 contains implementations of F_1 and F_2 , and C_N implements F_N and F_1 . Also, F_2 is implemented in both C_1 and C_2 . The two implementations of each functionality are different as suggested in [18]. This provides redundancy and diversity, so that attacks against one implementation will not necessarily succeed against both. The more dissimilar the implementations, the more likely that a different strategies would need to be applied to each one before the overall functionality would be impaired. To make discussion easier, we indicate the connection between (C_i, F_j) and (C_m, F_m) as $(C_i, F_j)-(C_m, F_m)$, or simply the link L_{ij-nm} .

In the DFFA restructuring of the original IDS, components are ordered counter clockwise into a circle. Within each component, the output of one functionality is passed directly as input to the other one it contains. For instance, F_n is passed to $F_{R(n+1)}$. The exception is that the output of F_N is passed to the output device, and the input of F_1 comes from the input devices (here, the sensors). Under normal processing conditions, the output of $F_{R(n+1)}$ in C_n is forwarded one component ahead to $F_{R(n+2)}$ in $C_{R(n+2)}$ as shown in Figure 3. For instance, F_2 in C_1 forwards its output to F_3 in C_3 (instead of C_2), and F_3 in C_2 forwards its output to F_4 in C_4 .

SIDS also includes backup connections between components. These backup connections provide paths between functionality pairs F_n and F_{n+1} that located in different components; backup connections are not used under normal conditions. For example, there is a backup communication path between F_2 of C_1 to F_3 of C_2 . Figure 4 displays backup connections in a four-component DFFA system. Any connection that links two consecutive functionalities and not used in the normal operation is considered a backup connection. There are $2N-2$ such connections.

If a communication path is discovered to be in failure mode and the related component is in safe mode, components in the DFFA system use backup connections as a reconfiguration.

For example, if L_{12-33} is unavailable, the DFFA system can use L_{12-23} and L_{23-34} as the replacement of the broken path. In general, connection $L_{nR(n+1)-R(n+2)R(n+2)}$ can be replaced with $L_{nR(n+1)-R(n+1)R(n+2)}$ and $L_{R(n+1)R(n+2)-R(n+2)R(n+2)}$. Failure detection of communication is not a focus of this research but is briefly discussed in section 3.2.

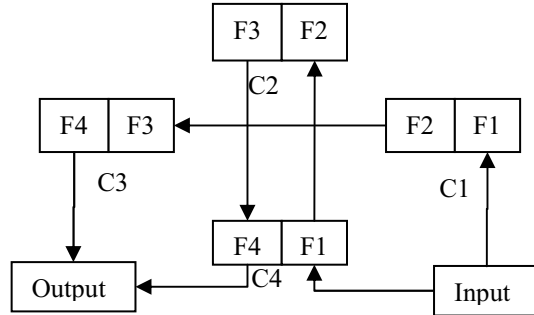


Figure 3: a DFFA structure for a four-component system

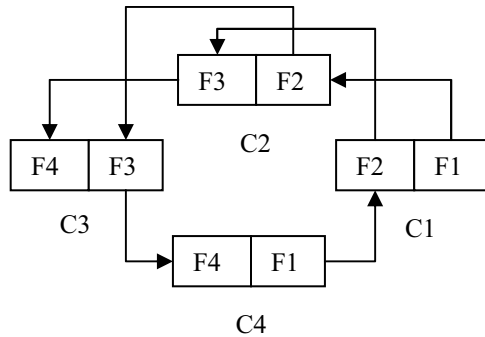


Figure 4: backup paths in a four-component DFFA structure

The SIDS is designed to include process recycling. If a component fails but the communication paths are still working, the DFFA system recycles the failed components. This recycling process should allow the fast return of services in the faulty components.

Process recycling is intended to use checkpoint technology to restart disabled components and bring them back to their original states. Rather than replace components with identical (unbroken) versions, the architecture specifies that process recycling will change the executables loaded into memory by reconfiguration. A simple way of doing this is to load different components at different locations and rename the process. That would deflect attacks aimed at well-known process names. A more complicated, but less

easily defeated approach, involves replacing some of the components with equivalent components, or enabling different code paths within the same component, perhaps randomly. The intent is to make the code a moving (or perhaps cyclic) target, so that an attacker will have a harder time identifying the specific weaknesses that are in place at a given instant. This will not foil an attacker well versed in the specific local technology indefinitely, but it will decrease the likelihood of a repeated attack being repeatedly successful, at least initially. A third alternative is to make use of the newer techniques involving code randomization [28,30].

Under normal conditions, the number of inter-component communication paths in the DFFA structure is still N , as in the original structure. Little additional inter-component communication overhead is introduced unless the integrity of the IDS is attacked.

To summarize, the proposed SIDS version of the original IDS is intended to provide a higher survivability than the original, though it will not be impenetrable. The first observation is that this new SIDS structure meets the well-known criteria that the failure of any one component or one connection should not put the whole system into failure mode [9] even if the original architecture lacks this feature. The SIDS system will be subject to failure only when the original architecture would fail, and if these new conditions are met: when at least two components are corrupted (or crashed), or two communication paths are blocked, or a communication path and a component fail at the same time. Even when two components fail at the same time, the system as a whole will not necessarily fail. This is clear when one considers that it is both the functionality within a component and the way that the functionalities are interwoven in the original IDS that provide survivability.

In the simplest case, where no redundancy is provided by the original IDS architecture, the SIDS restructuring gives redundancy through the replication of functionalities in different components. In more sophisticated original IDS, such as distributed IDS that themselves include redundancy of key components, the SIDS replication of functionality acts to further increase those existing survivability properties. It is therefore necessary for a malicious attacker who wishes to remove a functionality (such as detection or response) to do two things. First, to provide complete "coverage" by attacking all of the SIDS components that include the key functionality (or functionalities) that s/he desires to eliminate, and second, to address any underlying features of the original IDS that provide replication for that same functionality. For example, in Figure 3, the original simple system is subject to single point of failure. However, in the SIDS

version, the failure of components C2 and C4 will not put the system into the failure state since F1 and F2 in component C1 and F3 and F4 in component C3 are sufficient to process all messages. In a more complex implementation of an IDS than the one in Figure 3, for instance one that allows overlapping and distributed analysis components, the attacker would need to crash or slow **all** of the SIDS components that included the replication at the same time. Essentially this "doubles the workload" if the implementations of the replicated functionality are sufficiently different. At minimum, it increases the opportunity to detect an attacker who is seeking to "hide" by attacking the IDS. We anticipate that component recycling and communication path reconfiguration will help achieve still more reductions in the overall failure rate of the SIDS system as compared to the original.

3.2. Anomaly Detector

In the DFFA structure, reconfiguration and repair are triggered when either component or connection failure is detected. A good and efficient detection algorithm is very important to the success of the system – an overly aggressive detection algorithm will result in "too many" reconfigurations (and higher cost), while a detection algorithm that is too conservative may leave the IDS in failure mode. In terms of efficiency – the detection of failure must happen rapidly and with low overhead, or it will slow the IDS.

We propose the following detection strategy in our architecture. A "light" detector will be placed inside each component or else integrated with its development (as with code instrumentation). The first task of the detector is to observe the behavior of the component in which it is placed. When component integrity is "in doubt", the detector will report it to the next component in the circle. That next component logs the failure, then recycles and restarts the failing component. The Internal Sensor and Embedded Detector strategy discussed by Zamboni in [15] is an example of a technique that should prove effective – there are many instances of code instrumentation being a useful way to detect security violations and software flaws that would also be appropriate [25-27].

It is possible that a component will crash or significantly slowed for an appreciable time period before the internal detector detects the failure, or that the failure will also affect the internal detector. When that happens, it is up to detectors inside other components to detect the failure. This might be done by observing inconsistencies in output, or delayed response to a query, or through adding a "heartbeat" function produced by the component, for instance.

We next address fault detection in communication. Fault detection in general has been discussed extensively elsewhere as in [17,31-33]. Common methods include duplication, error-detecting codes, checksums, self-checking and fail-safe logic, watchdog timers and bus timeouts, consistency and capability checks, processor monitoring, and program monitoring.

For our purposes, we propose use of a “Watchdog timer” for the SIDS. This would be based on a cryptographic approach to aid detecting the functioning status of other components. In addition, as in [6], observable traffic between components should be encrypted to mitigate vulnerability V7 – similarly, any traffic vulnerable to tampering may also benefit from encryption and/or use of digital signatures.

Not all communication paths need to be handled identically. We deliberately limit our recommendation to traffic that is “observable” or vulnerable to tampering. By our definition, observable traffic includes information that is transmitted utilizing a communication path that can be directly examined, or whose contents can be inferred, by those we consider potential attackers. The more channels that are encrypted and the stronger/more elaborate the encryption and encryption protocols used, the greater likelihood that there will be a performance penalty. Further, the method used to protect traffic should vary depending on the attacker behavior we are concerned with. For instance: some of the simpler techniques for protecting communication can be broken by a determined attacker, for instance by capturing earlier messages and inserting them. However, such measures may suffice if the survivability goal does not include thwarting attacks of that nature, but rather to make it more difficult for an attacker to defeat the IDS. This is likely to be the situation when the IDS is faced with a pre-programmed broad based worm, for instance. In situations where traffic is likely to be observed or tampered with, the more expensive algorithms may indeed be warranted. In other situations, lesser measures may suffice. The decision about which to select should be made in the context of to an analysis of the goals of the system (and consequences of IDS failure) as well as the predicted threat environment.

There are several ways to handle securing channels. Our initial simple model is intended to work as follows: When the system is initialized, each component C_n will establish a secure channel with components $C_{R(n+1)}$ and $C_{R(n+2)}$. This channel is used by both the normal connections and the backup connections later on. The secure channel will include a session key and a pair of random numbers R_{AB} and R_{BA} . When C_A sends a message to C_B , it follows this protocol:

```

CA → CB
    RAB = RAB + 1
    Send MAB=(A, Encrypt (RAB, A, message))
CB → CA
    EncryptionKey=Key(AB)
    (new-RAB, A, message)=Decrypt (MAB)
    RBA = RBA + 1
    Send MBA=(B, Encrypt (RBA, B, new-RAB))
    
```

Each time a message is sent, a different R_{AB} is transmitted to prevent replay attacks. C_B can verify the identity of the sender by verifying that the correct encryption is used and the correct ID is encrypted in the message. C_B can also verify the freshness of the message by checking the freshness of the R_{AB} and detect whether any message is lost by comparing the received R_{AB} with the stored R_{AB} . Likewise, C_A can verify the message returned from C_B .

Successful transmission provides some reassurance that both the connection and the communication components are operable. For instance, C_A can thus detect the working state of C_B . The converse is much harder. Bianchini and Buskens [10] introduce an efficient algorithm to detect failure components in distributed system. Note that other algorithms are also possible. We propose using a modified version of [10] that queries the status of other components at varied time intervals. If C_B does not receive messages from C_A for an interval of time, D_B in the C_B sends a query message to the detector D_A in the C_A to check the status of the C_B . The detection process is summarized as following:

D_B side:

Function OnCorrectMessage

```

    StopTimer
    OK(CA)=true
    OK(LAB)=true
    StoreMessageArrivalTime(TM)
    TimeToCheck = EstimateTimeIntervalToCheck
    StartTimer(TimeToCheck)
    
```

End Function

Function OnTimer

```

    StopTimer
    Send M=(B, Encrypt(RBA, B)) to CA
    Receive(M, maxTime)
    If (timeout) then OK(CA)=false; Return
    End if
    (new-RAB, A)=Decrypt (M)
    if (new-RAB==RAB) then
        OK(CA)=true
        StartTimer(TimeToCheck)
    Else if (new-RAB > RAB)
        WaitFor(pre-determine time)
        OK(LAB)=Message new-RAB is received
    
```

```

    StartTimer(TimeToCheck)
  End if
End Function

```

D_A side:

```

Function OnQuery
  Send  $M=(A, Encrypt(R_{AB}, A))$  to  $C_B$ 
End Function

```

D_B asks D_A and D_A responds to D_B with the last message C_A sent to C_B as indicated by R_{AB} . If C_A is alive, D_B receives the R_{AB} from D_A . If R_{AB} received by D_B matches the stored R_{AB} , C_A is considered “safe”. If the R_{AB} received by D_B does not match, there are two possibilities: either because a message is still in transmission, or the communication path is failing. In our algorithm, C_A waits for a predetermined time and assumes that the communication path is broken if the specified message is not received by then. This is a relatively safe decision since reconfiguring the communication path should be efficient. If C_A is corrupted or crashed, D_B timeouts and assumes C_A is corrupted.

A key parameter here is the timer interval used to determine when the component should initiate a status query. If the time interval is too long, it will take longer to discover a failure and this might result in missing an attack (in a sense, a false negative). If the time interval is too short, the system will spend unnecessary resources on status checking. We propose using long time intervals while no failure component and connection is detected and short time interval otherwise. This suggestion is based on the fact that a single component failure cannot cause the whole system to fail. Thus the cost incurred due to the missed detection is low. If a component has already failed, however, an additional failure does have the potential to cause the whole system to fail (which causes high cost) and so needs to be detected as soon as possible.

When a detector discovers that a component is not working, the information is logged and the faulty component is recycled. An occasional false alarm does not harm security since the result is to bring the component back to safe mode. If a communication path is incorrectly marked broken, the system returns to the original path when the false alarm is cleared.

3.3. Other Components

Two additional considerations for SIDS functionality remain: Message Queuing and Survivable Storage.

Message Queuing – Information sent by sensors go through the Message Queuing components. All

messages put into the queue are persistent through use of secure storage as in [13]. Messages removed from the queue are marked as finished in the storage and eventually deleted. Thus, messages can be retrieved even after system failure as long as the message storage is not comp

Survivable Storage – Storage should be survivable, perhaps using the methods discussed in [13].

4. Conclusions

As IDS become more widespread and more powerful, and thus more attractive targets, their survivability becomes increasingly important. This paper proposed SIDS, which can be used as a target architecture via transformation of some existing IDS to obtain an IDS with improved survivability qualities. The core of the system is the DFFA structure, which provides the strengths and weaknesses indicated in Table 3. We argued that such a system has improved survivability characteristics in comparison with the original system through both redundancy (of functionality and communication) and reconfiguration capacity, as well as through internal monitoring of behavior. Each component in the system contains a detector and a process recycler. We introduced a novel detector algorithm based on cryptographic messages and risk analysis to determine whether other components are working correctly. If the component is corrupted or crashed, the process recycling component restarts the corrupted components to a backed up safe state. The structure we introduced here can be applied to other software systems in addition to IDS.

We project several areas for future work. First, there are clearly many ways to dynamically estimate “watch dog” intervals, and it would be useful to determine when alternate methods are more appropriate. Second, the performance effects of the internal detector should be assessed to ensure that it does not excessively degrade the system – results by other researchers indicate that such is likely to be the case. Third, the system should be assessed against real attacks to determine whether predicted survivability characteristics are realized in a cost-effective way in practice. Finally, the issue of communication securing raised in Section 3.2 should be extended to formally incorporate the concept of changing protocol based on vulnerability analysis and protocol expense.

Strengths	The SIDS can maintain high availability of the IDS by detecting and restarting faulty components of the IDS, and reconfigure the system.
-----------	--

	The SIDS can restore the IDS to safe state with information persisted in the secure storage.
	The SIDS can prevent single point of failure.
Weaknesses	Parameters such as “watch dog” interval are difficult to estimate and are system dependent.
	The system may still be vulnerable to V6.

Table 3: Strengths/Weaknesses of SIDS

References

[1] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner, “State of the Practice of Intrusion Detection Technologies”, *Technical Report CMU/SEI-99-TR-028*, 1999.

[2] S. Axelsson, “Research in Intrusion Detection Systems: A Survey”, *Chalmers University of technology, Technical Report TR: 98-17*, Revised in 1999

[3] T. Lunt, “A Survey of Intrusion Detection Techniques”, *Computers & Security*, Vol 12, 1993, pp 405-418.

[4] T. H. Ptacek, T. N. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”, *Technical Report, Secure Networks, Inc*, 1998

[5] T. Onabuta, T. Inoue, and M. Asaka, “A Protection Mechanism for an Intrusion Detection System Based on Mandatory Access Control”, *13th FIRST Conference*, 2001.

[6] R. Bace, P. Mell, “NIST Special Publication on Intrusion Detection Systems”, http://www.21cfrpart11.com/files/library/government/intrusion_detection_systems_0201_draft.pdf

[7] S. Bhattacharya, and N. Ye, "Design of Robust, Survivable Intrusion Detection Agent", *In Proceedings of the 1st Asia-Pacific Intelligent Agent Technology Conference (IAT'99)*, pp.274 - 278, 1999.

[8] T. Takada, and H. Koike, "NIGELOG: Protecting Logging Information by Hiding Multiple Backups in Directories", *International Workshop on Electronic Commerce and Security (in conjunction with DEXA'99)*, *IEEE CS Press*, pp.874 -878, Sep, 1999.

[9] J. S. Balasubramanian, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D.Zamboni, "Architecture for Intrusion Detection using Autonomous Agents", *COAST Technical Report, COAST Laboratory, Purdue University*,1998.

[10] R. Bianchini and R. Buskens, “Implementation of On-Line Distributed System-Level Diagnosis Theory”, *Trans. Computers*, Vol. 41, No. 5, May 1992.

[11] C. Wang, J. C. Knight, “Towards Survivable Intrusion Detection”, *Third Information Survivability Workshop*, 2000.

[12] S. Cheung, "An Intrusion Tolerance Approach for Protecting Network Infrastructures." *Ph.D. Dissertation, University of California, Davis*, September 1999.

[13] G. R. Ganger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, S. Oguz, V. Pandurangan, C. A. N. Soules, J. D. Strunk, J. J. Wylie. “Survivable Storage Systems”, *DARPA Information Survivability Conference and Exposition (Anaheim, CA, 12-14 June 2001)*, pp 184-195 vol 2. IEEE, 2001.

[14] J. Evans, and D. Frincke, “Trust Mechanisms for Hummingbird”, *IEEE Crossroads*, Issue 2-4, March 1996.

[15] D. Zamboni, "Using Internal Sensors for Computer Intrusion Detection", *CERIAS Technical Report 2001-42, CERIAS, Purdue University, West Lafayette, IN*, August 2001.

[16] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, 4 (3), pp.382-401, July 1982.

[17] D.P. Siewiorek, and R.S. Swarz, *Reliable Computer Systems: Design and evaluation*, 2nd edition, Digital Press, Bedford (MA), ISBN 1-555-58075-0, 1992.

[18] A. Avizienis, "The Methodology of N-Version Programming," Chapter 2 of *Software Fault Tolerance*, M. R. Lyu (ed.), Wiley, 23-46, 1995.

[19] D. Frincke, “Balancing Cooperation and Risk in Intrusion Detection”, *ACM Transactions on Information and System Security (TISSEC) 3(1): 1-29 (2000)*.

[20] J.P Anderson, “Computer Security Threat Monitoring and Surveillance”, *Technical report, James P Anderson Co., Fort Washington, Pennsylvania*, April 1980.

[21] S. Northcutt, L. Zeltser, S. Winters, K. Fredrick, R. W. Ritchey, “Inside Network Perimeter Security: The Definitive Guide to Firewalls, Virtual Private Networks (VPNs), Routers, and Intrusion Detection Systems,” *Pearson Education*, June 2002.

[22] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", *1997 National Information Systems Security Conference, October*, 1997.

[23] R. Gopalakrishna, E. H. Spafford, "A framework for distributed intrusion detection using interest driven cooperating agents", *Paper for Qualifier II examination, Department of Computer Sciences, Purdue University*, May 2001.

[24] E. H. Spafford and D. Zamboni, “Intrusion detection using autonomous agents”, *Computer Networks*, 34(4):547–570, October 2000.

[25] A. W. Krings, S. Harrison, N. Hanebutte, C. Taylor, and M. McQueen, “Attack Recognition Based on Kernel Attack Signatures”, *Proc. of the International Symposium on Information Systems and Engineering, (ISE'2001)* , June 25-28, pp. 413-419, 2001.

[26] S. Elbaum and J. Munson, "Software Black Box: an Alternative Mechanism for Failure Analysis", *International Symposium on Software Reliability Engineering* , IEEE, 365-376, October 2000.

[27] S. G. Elbaum, J. C. Munson, "Intrusion Detection Through Dynamic Software Measurement", *Workshop on Intrusion Detection and Network Monitoring*, 1999.

[28] G. S. Kc, A. D. Keromytis, and V. Prevelakis. "Countering Code-Injection Attacks With Instruction-Set Randomization," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 2003.

[29] D. Denning, "An intrusion-detection model", In *IEEE Symposium on Security and Privacy*, pages 118-131, Oakland, USA, 1986.

[30] D. Evans, and D. Frincke, "Security Enhancement Through Randomization," *Proceedings 7th Cybernetics and Informatics*, Orlando, July 2003.

[31] K. Vijayananda, "Distributed fault detection in communication protocols using extended finite state machines", *1996 International Conference on Parallel and Distributed Systems (ICPADS '96)*, June 03 - 06, 1996

[32] P. Stelling, I. Foster, C. Kesselman, C. Lee, G. v. Laszewski, "A Fault Detection Service for Wide Area Distributed Computations," *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, August 1998.

[33] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard, "Fault Detection and Diagnosis in Distributed Systems: An Approach by Partially Stochastic Petri Nets", *Discrete Event Dynamic Systems*, Volume 8 , Issue 2, Pages: 203 - 231, June 1998