Self-monitoring Monitors

Salvatore J. Stolfo, Isaac Greenbaum, and Simha Sethumadhavan
Columbia University
Computer Science Department
Intrusion Detection Lab
April 27, 2009
Revised: June 3, 2009

Abstract

Many different monitoring systems have been created to identify system state conditions to detect or prevent a myriad of deliberate attacks, or arbitrary faults inherent in any complex system. Monitoring systems are also vulnerable to attack. A stealthy attacker can simply turn off or disable these monitoring systems without being detected; he would thus be able to perpetrate the very attacks that these systems were designed to stop. For example, many examples of virus attacks against antivirus scanners have appeared in the wild. In this paper, we present a novel technique to "monitor the monitors" in such a way that (a) unauthorized shutdowns of critical monitors are detected with high probability, (b) authorized shutdowns raise no alarm, and (c) the proper shutdown sequence for authorized shutdowns cannot be inferred from reading memory. The techniques proposed to prevent unauthorized shut down (turning off) of monitoring systems was inspired by the duality of safety technology devised to prevent unauthorized discharge (turning on) of nuclear weapons.

1. Introduction

Much time and effort have been put into developing systems and processes that can monitor and protect against various types of threats. For example: Anti-virus programs are designed to identify and disable all viruses that attempt to infect a computer. Intrusion detection systems are supposed to notify and protect against unwanted infiltrations into a system.

A knowledgeable user or a system administrator who desires to perpetrate malicious acts can simply disengage a monitoring system designed to identify and detect malicious acts. (If the personnel responsible for securing a system are themselves insider attackers, the monitoring systems that send alerts to them would therefore be rendered useless even if not disengaged.) Hence, all monitoring systems, regardless of how well they meet their stated objectives, suffer from the same fatal vulnerability, they may be ***turned off (or ignored)***. If a less sophisticated attacker begins his attack by initially shutting down the monitors, and that act provides no warning or alarm, the attacker will have free reign to pursue his attack unimpeded by any process. Any relatively unsophisticated subsequent attack can successfully cause damage.

One may presume that a monitoring system designed with a "heart beat" that continuously informs personnel of its own state, and that it is actively at work, will

adequately solve the problem. In some cases this may be so. However, the clever attacker may possibly use a man-in-the-middle attack to ensure the signal is continuously broadcast, but the core monitoring functions may still be disabled. In other words, the techniques proposed herein may still be applicable to the core functions of the monitoring system whether or not it persistently signals its own state.

Furthermore, there may be some circumstances in which the owner of the system may legitimately wish to discontinue the monitoring process without generating alarms, say during a system update process that causes sufficient change to system state sufficient to cause many alarms, or worse lock down the system. Graceful control of the monitoring infrastructure is clearly a desideratum.

The concept proposed herein involves a uniquely formed network of monitors that are created anew when a system is first booted up and each time thereafter. A series of system boot ups creates a diversity of independent monitoring networks. No information persists across each instance of a monitoring network that provides any useful information to an attacker.

The network of monitors consists of a set of monitoring programs each designed to logically connect two at least two other randomly chosen monitors. Each such monitor is required to have at least two other monitors connected to it. The key concept is that if a monitoring program is shut down (eg., by use of the "kill" command on Linux systems, or "end process" command on Windows systems), then at least one of the two monitoring programs will notice the killed process and issue an alarm. The implementation of this alarm function and protecting it from interception can be implemented by a number of designer choices, such as an irrevocable destruction of critical data on a disk to prevent the host from being used any further, alternatively a "secret back channel" can be employed communicating information to a number of security personnel in a manner that survives interception unless all security personnel collude. There may be many methods to implement and protect the alarm, but in this paper we are focused primarily on the means of detecting when an attacker, such as a knowledgeable insider, attempts to shut down the system monitoring function.

It is thus the goal of this paper to present a protocol for dealing with this potential attack in a way that allows an authorized agent to turn off the system but raises an alarm during an unauthorized attempted shutdown. Our goal is to provide an appropriate answer to the question of ***"Who monitors the monitors?"***

2. Attack Models and Assumptions

2.1 Attack Models

The ultimate goal of the attacker in all our models is to disable (by killing or otherwise corrupting) the critical process (CP) that we hope to protect (e.g. AV, RUU monitor).

Given these assumptions, we consider three increasingly sophisticated threat models:

(A) An attacker cannot see inter-process traffic.
(B) An attacker can see inter-process traffic.
(C) An attacker can see inter-process traffic and can shut off every process on one computer simultaneously.

Attack Model A considers an attacker with very limited capabilities and serves as a proof-of-concept model for the validity of the general strategy proposed below. If the proposed protocols cannot identify unauthorized attacks when an attacker is so limited, there is no need to test against more sophisticated attack models.

Attack Model B considers an attack by a slightly more sophisticated attacker. As the proposed protocols are critically based upon interprocess traffic, the next most logical model to consider would be to relax the privacy assumptions regarding that traffic were relaxed. The most basic relaxation would be to allow the attacker to identify the sender and receiver of any traffic, as well as when it is sent. The proposed techniques for addressing Attack Model A fail under these new circumstances and extensions are proposed.

Attack Model C represents a further extension of the previous model. The protocols proposed to address Attack Model B (and A as well) rely on the idea that there must necessarily be a delay between shutting off a process and a second. In this model, we relax this assumption and allow all monitors (or a subset thereof) to be shut down simultaneously. With this capability, an attacker can overcome the techniques employed to counter Attack Models A and B.

We did not, however, consider memory-based attacks. The reasons for this are that we assume that they can be countered by specially designed hardware (TPM for bootup, IDS chip for ongoing monitoring) and software (polymorphic code for the monitors to make each process unique). See **Further Research** for more details about this assumption.

2.2 Assumptions

While this paper primarily focuses on detecting and warning against unauthorized attempts to shut down a critical process, we believe that the techniques detailed herein can be extended to apply to identifying potential attempts to corrupt the code.

With regard to an attacker's capabilities in executing an attack, we assume that an attacker has the ability to see all processes that are running and can kill any of them.

The design of the below procedures depend upon processes being shut down in a specific sequence. In an operational monitoring system, the following characteristics must be implemented to achieve the desired behavior:

1) An attacker cannot read the identity of the next monitor in the sequence by scanning memory. This assumption can be practically implemented using a polymorphic code. See **Further Research** for more details about this assumption.
2) The time between monitoring pings is less than the time required by an attacker to shut down a second process. If that were not the case, an attacker could exploit that weakness by shutting down the monitors in rapid succession and reduce the risk of triggering an alarm.

3) A further potential problem would arise if an attacker could kill a process and set up his own process in its place in the time between "monitoring pings." As in the previous concern, however, it should be possible to adjust the frequency of the pings such that the time needed to set up any new process would be longer than the time to the next ping.

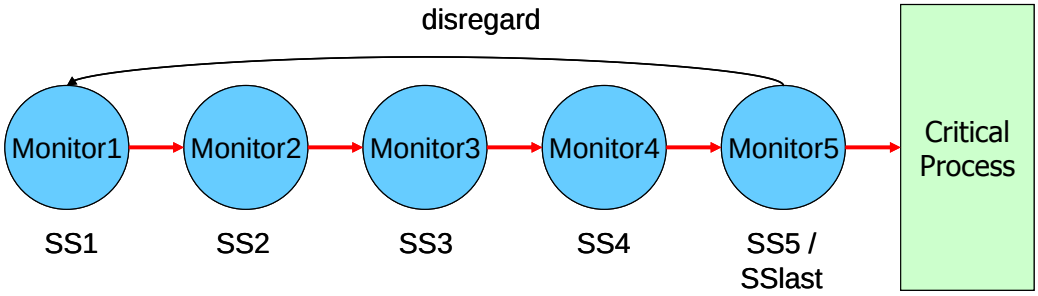
4) Network congestion and computational limitations can lead to false positives when incorporating too many hosts or monitors as "ping" response times may be slowed enough to be considered as non-responses.

5) Any communications between two distinct hosts need to be secure and in such a way that the identities of the two hosts are authenticated. This is to eliminate the proxy attack from consideration as a means of attacking the proposed system.

3. Proposed Solution

Our proposed solution to this problem is inspired by the safety protocol employed in nuclear weapon safety. See Related Research for more details.

The basic underlying framework of our proposed solutions for all three attack models is to create a ***random*** directed cycle of monitors at ***bootup*** using a TPM or other secure bootup process, wherein the randomization is based on a seed known only to users with the authorization to shut the system down. In this cycle, each monitor only cares about the next monitor in the cycle. However, one of the processes must disregard any actions done to the process it is monitoring. The disregarded monitor is thus the first process (SS1) to be killed in the SS. The second process (SS2) to be killed in the SS is the process that (SS1) was monitoring. As the process monitoring it has been killed, no alarms will be raised when SS2 is shut down. Similarly, the third process (SS3) to be killed in the SS is the process that (SS2) was watching. Again, no alarms will be raised as the only monitor that cares about whether it is running, i.e. (SS2) has previously been killed. The remaining order of shutdowns in the SS is constructed similarly; the last process (SSlast) to be shut down in sequence is process (dr). A user that is authorized to shut down the critical process can recover the proper SS by following the logic of the bootup process.



However, if any process is killed in an order other than the SS, the monitor that is watching it will still be alive and will raise an alarm.

If implemented correctly for each threat model, the probability of correctly guessing the SS comprised of n monitors is $1/(n!)$. ***Additionally, with this construct, we realize the important goal that the sequence needed to shut down the monitors without raising an alarm is not stored in memory anywhere, but rather it is stored in the structure of the monitoring network itself.***

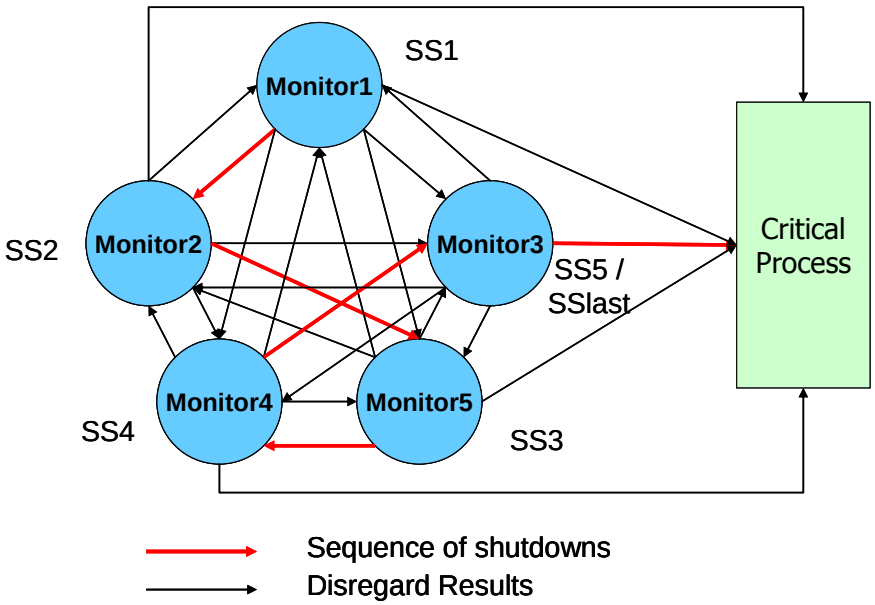
Using this framework, we can then apply them to the three threat models we delineated.

(A) In this threat model, we assume that the attacker can not see the traffic between processes, i.e. the attacker does not know which process is monitoring which other process. Under these assumptions, a simple directed cycle of monitoring would be sufficient to implement the SS. If an attacker cannot see the cycle, then he must guess a random permutation of the n monitors of which there are $n!$ possibilities. Thus, for a large enough monitoring network, a simple loop can reasonably reliably warn against unauthorized attempts to shut down the CP.

(B) However, if we assume that the attacker can see inter-process traffic, then the structure of the entire cycle is known to the attacker. The only safety mechanism remaining is the identity of the first monitor in the SS; once SS1 is known, SS2 and subsequent processes in the SS can be determined with probability 1. The probability of guessing the correct SS would thereby increase from $1/(n!)$ to $1/n$.

Under this threat model, therefore, we propose an extension of the original protocol. We still maintain an underlying directed cycle; but, in addition, we also require all processes to monitor all other processes. Unlike the monitoring between consecutive processes in the SS, monitors will not raise alarms if a process other than the next one in the SS is killed. Effectively, each process

receives information from all other processes, but only reacts to at most one of them.

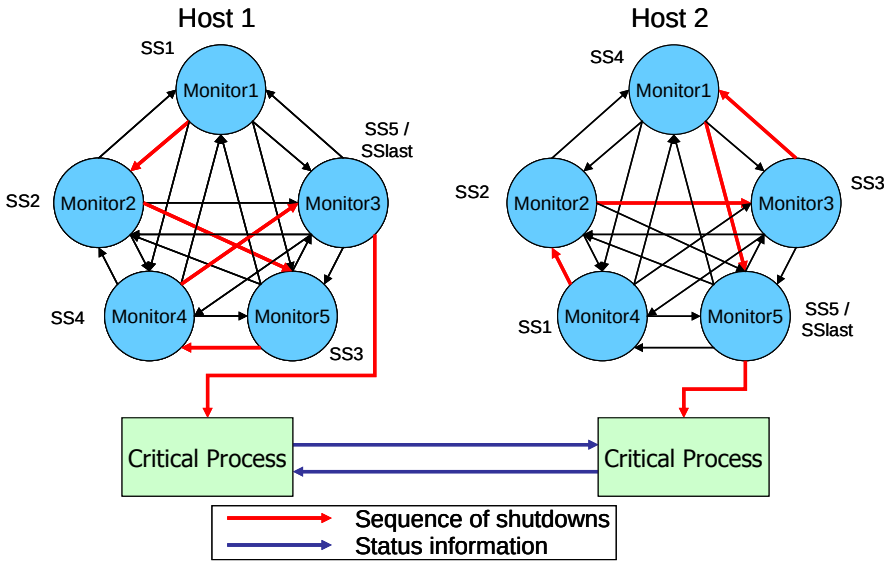


With this extension, the attacker's ability to view inter-process traffic provides no additional information as to the correct SS. Each monitor looks identical: it sends and receives information from every other monitor. Furthermore, as long as a monitor receives "pings" from the monitor it is watching, it reacts to receiving a ping from any monitor in an identical way. It is only when the next process in the SS is shut down that the monitor behaves differently, but by then, an alarm is raised and the attacker cannot glean any information about the sequence. The graph of processes and monitors effectively is complete and the true SS is therefore obfuscated by meaningless, yet symmetric, inter-process traffic. A correct identification of SS1 will therefore provide no information regarding the identity of SS2; the probability of guessing the correct SS therefore returns to $1/(n!)$.

(C) As in threat model (B), we assume the attacker can see inter-process traffic. Thus for this model, on any given computer, we should use the protocol established in (B) such that the probability of a correct guess remains $1/(n!)$. However, with the additional assumption that the attacker can gain control of the computer (via a root-kit, or some other means) and the ability to shut off all monitors at once, the attacker can disable the CP without the requirement of figuring out the SS.

To counter this additional threat, the protocol must be extended to share the monitoring load across multiple remote hosts. In addition to the network of monitors on each computer, a "communicating process" should be created and attached to the created to the critical process to broadcast the "alarm" status on each host.

Two types of unauthorized shutdowns must be monitored within this larger system: 1) a monitoring process is shut down out of order of that host's SS and 2) a "communicating process" is shut down. In the event that a type (1) shutdown occurs, the communicating process should broadcast to all hosts in the network that an alarm has been raised in its system. In the event of a type (2) shutdown, the communicating process on each other host must realize that it can no longer communicate with that original host and raise an alarm.



Using this protocol, an improper shutdown sequence on any individual machine will raise an alarm on all connected remote hosts. Thus even if an attacker manages to gain control of one host and shuts off all its monitors and CP, other hosts in the network can identify that there is a potential unauthorized shutdown on that host and react accordingly. With a large number of computers, the probability of simultaneously shutting off the monitors on all hosts is reasonably remote.

Summary of Models

Attack Model	Attacker Capabilities	Proposed Solution
A	Cannot see interprocess traffic	Simple cycle of monitors
B	Can see interprocess traffic	Complete graph of monitors
C	Can see interprocess traffic and can shut off multiple processes simultaneously	Complete graph of monitors and remote monitoring
Memory (not considered)	Can read memory to determine next in sequence	Polymorphic code, specially designed IDS hardware
Bootup (not considered)	Can learn information about sequence at bootup	TPM

4. Alternative Protocols

There are two alternative protocols that could be implemented to achieve similar results. However, each of them has their own advantages and disadvantages.

The first alternative is a different approach to the protocol presented to address attack model (C). As opposed to a system in which each computer can have its own unique SS and interacts with other computers via a communicating process, a distributed system, in which one SS applies to all computers, could be employed.

In this system, the seed for the random cycle on the first computer is passed to all others in the network. Each computer then creates a similar network of processes on its machine. Once these networks are set up, monitoring processes communicate across hosts such that all processes "ping" all other processes, but the SS1 of each only cares about the SS2 of each and so on. In order to shut down the system without alarms, every SS1 process in the network must initially be shut off followed by every SS2, and so on until all monitors are turned off.

The main advantage to this protocol is the fact that it will be much more sensitive to any attempts to shut off processes on any of the linked hosts. However, the drawbacks for this plan are numerous and outweigh any gain. Firstly, the amount of additional network traffic generated over the proposed protocol is greater by orders of magnitude. Secondly, the necessity for passing the seed for the random cycle to a remote host introduces yet another possible point of vulnerability for the system. Lastly, the enmeshed nature of this network makes it that much harder to extricate one host from the network, even if done so with proper authorization.

The second alternative represents a paradigm shift in terms of approaching a solution.

The idea for this protocol is again to create a directed cycle of monitoring processes (not necessarily random); however, the processes need not be shut down in any specific order. Processes continuously run and monitor the next in the sequence; yet if a process or host wishes to exit the network ("leaving process"), it must inform the process that is monitoring it that it wishes to leave the network by passing it a special code. The "leaving process" then securely passes along the information concerning the process it is monitoring to the process that is monitoring it. In this way, even though a process has left, a cycle of monitoring processes and the protection it provides are preserved.

The advantages to this protocol lie in the reduction in the inter-process and inter-host traffic as well as provide a method by which monitors can leave the network with proper authorization. On the other hand, the disadvantages include the computational overhead necessary for secure communication between processes and the fact that the validation code for "leave requests" must be stored in memory (where it may be subject to attack).

5. Related Research

Interestingly, the concept is similar in nature to nuclear weapon safety systems. It is natural that dangerous weapons be prevented from accidental discharge, hence safety mechanisms have been developed to prevent unwanted discharge unless and until a specific unique code is provided to the weapon. Whereas one desires to prevent the unauthorized or accidental discharge (turning on) of a nuclear weapon, hence requiring a safety mechanism, this paper proposes a system to prevent the unauthorized or accidental disengagement (turning off) of a monitoring system, hence requiring a means of generating an alarm whenever an illegitimate attempt is made to render a monitoring system inactive.

Nuclear weapons are carried by planes, mounted on rockets, and are generally deployed with little fear of accidental arming and detonation. In order to detonate a nuclear device, energy must reach the critical components of the bomb. Safety protocol thus requires that an energy barrier must surround these components, thereby isolating the components from any significant amount of energy. Energy gateways, known as stronglinks, are embedded within this energy barrier. A stronglink serves as a physical gateway in the barrier for energy to pass through to the inner critical components. At rest, the stronglink is as impenetrable to energy as the rest of the barrier. However, upon receiving a correct unique signal (UQS), the stronglink effectively opens up and allows energy to pass into the interior of the barrier. The stronglink and its UQS are constructed such that only an "unambiguous communication of intent" will allow the bomb to detonate. The stronglink contains a "UQS discriminator" that evaluates each sequence event in succession and responds accordingly, i.e. allowing the next event to be processed in the event of a correct UQS event and permanently sealing the barrier in the event of an incorrect UQS event. The UQS is designed such that the probabilities of a random electrical (or other medium of transmission) signal occurring under abnormal circumstances or unauthenticated agents generating the correct sequence are very low. All these safety mechanisms combined provide a very high degree of safety for nuclear weapons.

It is this interaction between the UQS and stronglink in nuclear devices that inspired a solution to our problem. While it was clear that a cycle of mutually checking monitors should prevent attempted shutdowns from going unrecognized, it remained to be seen how such a scheme could be implemented without storing any shutdown key information in memory. It is here that the stronglink and UQS system provided the key insight. Whereas the goal in nuclear safety protocols is to refuse to arm the device if even one UQS event is incorrect, our task presents the dual problem, i.e. to raise an alarm if even one step in the shutdown sequence (SS) is incorrect and to allow the user to proceed with shutting down the next process unhindered if the sequence is correct. Effectively, the key would be implied and validated by the very structure of the monitoring network.

[2] describes methods for limiting the types of randomization to avoid sequences that can be commonly entered either by accidental or by guess. Given that the main focus of these restriction were to preclude typical signals emitted by electrical discharges caused by damage or failure, these methods need not be applied to the protocol.

In addition, [2] describes procedures for keeping the UQS generator and stronglink separate until there is an intent to "detonate". This prevents any unauthorized actions form occurring, malicious or not. This concept should be applied to the proposed protocols to provide a further measure of protection against unauthorized shutdowns. If the correct sequence were stored on a remote host or in some segregated and protected section of memory, the monitoring system's sequence would remain unknown should an attacker fail to additionally gain access to the remote host or area of memory.

6. Conclusions and Further Research

Ultimately, we believe that the general techniques presented herein can serve as the basis for more secure monitoring systems. However, further research is required to convert this protocol into a scalable and reliable product.

The first necessary area of further exploration relates to the use of a TPM chip in such a way as to ensure that the monitoring processes are loaded in an uncorrupt state.

While we have implemented a software-based version of a system incorporating these protocols, an ideal system would be implemented as close to the hardware layer as possible. ***Specially designed hardware components gain even more importance when the "threat model" is expanded to include an attacker's ability to introduce corrupt code to the monitoring processes.*** By incorporating hardware into the system, a designer can reduce the likelihood of a successful attack as an attacker would be required to physically alter the machines in order to disable the system.

In addition to hardware components, the proposed monitoring system can be further protected by incorporating polymorphic code into its design. Implemented correctly, the bootup process can then be designed such that each monitor in the network is ***unique***; thus if an attacker can comprise or disable a monitor by reading its memory, s/he cannot leverage that information to read the memory of others. The monitoring system therefore becomes that much harder to disable.

Another area of further research surrounds multicast protocols for sending out status messages between the communicating processes in each computer. With a large number of computers in a mutually protecting monitoring system, the amount of inter-computer traffic generated with a fully-connected graph of one-to-one connections can become onerous for the network. Sending messages to a router's multicast address with an appropriately designed LSA protocol should simplify the delivery mechanisms significantly and make for a more practical system.

References

[1] D. Plummer, W. Greenwood. The History of Nuclear Weapon Safety Devices. Sandia National Laboratories. June 8, 1998.
[2] The Unique Signal Concept for Detonation Safety in Nuclear Devices. UC-706. Sandia National Laboratories, System Studies Department. December 1992.

Acknowledgments

This work was partially supported by a Department of Homeland Security I3P/DHS Contract: 2006-CS-001-000001-02 entitled Cyber Security Collaboration and Information Sharing, Insider Threat Project, subcontracted through the Institute for Information Infrastructure Protection hosted at Dartmouth College.