Intrusion Tolerant Systems *

Partha P. Pal, Franklin Webber,
Richard E. Schantz and Joseph P. Loyall
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
{ppal, fwebber, rschantz, jloyall} @bbn.com

1. Background

Many intrusions on computer systems often target specific applications. By exploiting vulnerabilities in the infrastructure or in an application, intruders try to stop the application altogether or to make it behave in an abnormal way. In either case, their objective is to stop the application from being useful.

Considered abstractly, there is a competition between the application (trying to deliver useful service) and the attacker (trying to prevent the application from doing so) over resources at various levels. In order to perform, the application needs resources ranging from CPU time, memory and network bandwidth to higher level ones like data and objects. When an intruder gains enough control over one or more resources, the attack is successful. For instance, a DOS (denial of service) attack on an application such as a web-server results when an attacker manages to deprive the application of the required CPU or network resources.

The approach taken by traditional security engineering attempts to protect the infrastructure resources from intruders by establishing a preventive barrier. A barrier acts like a catch all guard protecting all resource consumers upstream without much interaction or cooperation between the protectorate and protectors. For instance, firewalls, a network layer barrier, do not normally interact with applications that use the network. Similarly, IDSs (Intrusion Detection Systems) operating at various system layers hardly ever cooperate among themselves or interact with other kinds of applications. While the world is gravitating towards more use of COTS components and more integration of diverse and distributed resources, security mechanisms attempting to prevent attacks are bound to be imperfect. This is evidenced in the recent "defense in depth" approach which calls for layering of defensive mechanisms of various capabilities so that an attacker faces multiple barriers and in order to succeed they have to overcome all of them.

Economic pressure dictates the use of COTS components rather than specialized ones. Component developers are under pressure to market their products quickly and hence often release products with errors and vulnerabilities waiting to be discovered. Interoperability requirements dictate that such systems be more open than closed. This openness and the distributed nature lead to more access points that an attacker could target. Finally, understanding and managing a system always lag its development and deployment: new attacks are always discovered *after* deployment. For these reasons, even the defense in depth approach cannot guarantee that critical systems will be completely shielded from the attackers. Some attacks will be able to compromise the barriers and alter the availability and quality of systems resources, and thereby affect the application.

Because of these reasons, we want to achieve intrusion tolerance by enabling the applications survive the effects of intrusions. The rest of the paper is organized as follows. In Section 2 we first present our position for this workshop, followed by some elaboration of key ideas. Section 3 describes our recent work in this and related areas. Section 4 concludes the paper.

2. Our Position: Support for Intrusion-aware Survivable Applications

We argue that development and support of intrusion-aware survivable applications, i.e., applications that react to intrusions and survive their consequences, are key problems in the area of intrusion tolerant systems. Even though the idea of intrusion-aware, survivable applications seems like a natural part of the defense in depth model, there is no easy and systematic way to support such applications in today's distributed systems infrastructure. We introduce the required support in the middleware that mediates between the application and the infrastructure.

*This work is sponsored by DARPA under contracts No. F30602-00-C-0172 and No. F30602-99-C-0188

A survivable application must incorporate a "survivability strategy"–a specification distinct from its functional requirements, for what to do when intrusions happen. Usually, these strategies involve adaptation and awareness of the environment and system resources. Our proposed middleware capabilities aim to make implementation and experimentation with survivability strategies more systematic, realistic and cost effective. Key aspects of our approach are outlined below.

- *Focus on symptoms:* If an application can survive the effects caused by an intrusion, it has effectively tolerated the intrusion. Our objective is to help the application in doing so. Consequently, we focus on addressing the symptoms caused by attacks, rather than preventing or detecting, diagnosing, containing or irradicating an attack. One could argue that we are recovering from an attack in an indirect way.

- *Adaptive and unpredictable response:* The ability to adapt to changing environmental and operational conditions is a key for surviving the symptoms of intrusions. However, in the context of pre-planned and coordinated attacks, adaptive responses that are predictable can be easily compromised. Therefore, adaptive responses must be unpredictable to the attacker.

- *Bridging the gap between the application and the infrastructure:* It is clear that the disconnect between an application and the infrastructure prevents the application from being easily aware of and rapidly responsive to changes in the availability and quality of resources. Advanced middleware such as QuO [6], which is an adaptive software layer operating on a distributed object base, helps bridge the gap. We utilize the QuO middleware to coordinate the capabilities required for supporting intrusion aware survivable applications.

The next three sections provide more details of our approach and identify several important issues that we are currently investigating.

2.1. Addressing the Symptoms

In order to devise an effective survivability strategy for an application, one has to first think about the application's survivability requirements: what kinds of intrusion are considered and what should be done to cope with them if they are even partially successful. We propose to formalize these in terms of symptoms (failures, if we follow fault-tolerance terminology) caused by the attacks as opposed to the attacks themselves. For example, instead of framing a strategy for "syn-flood" attacks, we focus on how such an attack may manifest itself in different system layers. At the application level, we may see one or more of the following symptoms:

- one or more requests blocked indefinitely.
- one or more requests timing out or throwing exception, even when retried multiple times.
- one or more objects crashing, perhaps repeatedly on restarts.

At the network level, we may see one or more of the following symptoms:

- abnormal traffic volume in a network segment.
- unexpected content in network traffic.
- overload/crash of network devices such as routers.

At the operating system level we may see:

- presence of unusual files (programs, scripts).
- presence of unusual processes and CPU load.
- unusual usage pattern of network interface and/or system calls

Some of these symptoms can result from natural causes however, in the context of intrusions, symptom occurrences do not follow any natural distribution often associated with "normal" faults. Symptoms may appear simultaneously, affecting multiple components of the system. They may also appear in stages, plaguing different parts of the system one after another. These are some of the factors that make coping up with intrusion symptoms harder. We are trying to address it by enhancing fault-tolerance techniques to deal with these issues and applying the enhanced techniques to different kinds of system resources.

Approaching intrusion-tolerance from the symptoms has both pros and cons. Intrusions can be very versatile. The same attack may manifest itself in different system layers in different forms under different environmental conditions. New attacks are always being devised. Trying to accurately specify all of the attacks or attack classes to tolerate can be a difficult task. Devising strategies in terms of symptoms helps us manage this complexity. A finite number of symptoms can (at least partially, if not fully) cover multiple known and unknown attacks because many attacks produce the same symptoms (such as object crash or network overload). On the other hand, not all symptoms are equally bad for every application and some may even be benign. An observed symptom may or may not be a positive indication of an intrusion either. However, if the application is equipped with a strategy to deal with most of

these symptoms, we believe we can achieve a significant amount of intrusion tolerance. One underlying problem that complicates the task here is that responding to every observed symptom may trigger unnecessary adaptation. If we are not careful, this may launch a self-denial of service. Several possibilities are under investigation. Symptoms may be categorized based on severity and the application may respond to the most severe ones and ignore the less severe ones. Sometimes, the severity of a symptom may change depending on external events. For instance, an application may choose to respond a network layer symptom only when external IDSs suspect an ongoing network attack.

Finally, formulating intrusion tolerance in terms of symptoms is not unprecedented. The kind of intrusions addressed by recent work on wrappers at NAI[1] and the work on protecting the win32 dlls at RST [4] can be viewed to be categorized in terms of symptoms, for instance, all attacks that attempt to write to the start-up folder.

2.2. Survivability Strategies Using Adaptive and Unpredictable Responses

Once we identify the symptoms to monitor for surviving, we devise the strategic responses to the symptoms. From one perspective, a strategy could be *pro-active* or *reactive*. A pro-active strategy dictates a course of action (COA) *in anticipation* or *in preparation* for a future attack. For instance, anticipating simultaneous attacks on multiple replicas, one may deploy a non-replicated stand-by which can be put in service at short notice. As an example of a pro-active strategy that has the "in preparation" flavor, consider interfacing with a firewall mechanism so that we can block IP traffic from an infected host when we know its identity. Reactive strategies involve measures taken in reaction to some observed event. For instance, when a host is under attack one might consider migrating objects and services from that host.

From another perspective, a strategy can be *defensive* or *tolerant*. A defensive strategy attempts to prevent the symptom from resurfacing, whereas a tolerant strategy attempts to find ways to avoid the symptom. An example of defensive strategy is to block all IP traffic from a suspected host. Migrating replicas from the suspected host would be an example of a tolerant strategy. These different views of strategies are not mutually exclusive, and a comprehensive strategy needed by an application will probably involve all perspectives. Devising the right strategy for a particular application is non-trivial. In the initial stage of our application, we use an ad-hoc case analysis to formulate a strategy.

Adaptation is a key capability for supporting any survivability strategy. Symptoms dictate the nature and form of the adaptation. For instance, repeated replica crash on a single host, a symptom observed at the application level, could be responded to by migrating the replica from the host in question to a different one. In the case of a pre-planned and coordinated attack, adaptive response by itself is not enough if the response can be predicted. If the attacker can accurately predict which host the replica will be migrated to, he can plan ahead to re-target the migrated replica in its new location. Therefore, adaptive responses should be unpredictable to the attacker. This way, pre-planning will be harder, and coordinating an ongoing attack to counter the adaptive responses will take longer and be less reliable. This in turn, will improve the application's survivability.

Incorporating unpredictability is also non-trivial. First, there has to be multiple options to choose from. Then, response selection mechanism should have the capability to select different response(s) to the same symptom at different times. Finally, not all of the responses may be equally effective. In particular, there may be one or more of the available options that must be engaged. The response selection mechanism must be aware of such constraints as well.

2.3. Bridging the Gap Between Application and Infrastructure

Attempting to build an application with a survivability strategy within the current distributed systems infrastructure , we find that there is a disconnect between what the application needs and what the infrastructure provides. To be specific, the following are key issues that need to be addressed by developing new capabilities in the middleware-space between the application and the infrastructure:

- *Awareness and control of infrastructure*: Attacks affect availability and quality of system resources. Since a survivable application needs to cope with these effects, awareness of the resource situation is very important. In order to respond to attack symptoms effectively, the application may need to exert some control over parts of the infrastructure as well. In general, the core issue here is to make the quality and availability of resources translucent, instead of opaque as is done in older middleware.

- *Resource redundancy and its management*: Pre-planned and coordinated attacks may cause resources to fail in an arbitrary manner, perhaps simultaneously or in stages. As in fault-tolerance, resource redundancy is critical for tolerating such failures, and resource managers must be capable of addressing Byzantine failures. All resources that

are critical for an application will have to be redundant and managed through an uniform interface so that the application can easily integrate with them to achieve the desired degree of awareness and control. Protection of resources and resource management systems is also very important. Continuing with the example adaptive response of migrating replicas, if the attacker can exploit the redundancy mechanism, he can force the replica to migrate to a host of his choice which may be easier to attack or has already been compromised. A new self-protected redundancy mechanism managing multiple resources and implementing Byzantine fault-tolerance algorithms is required. Byzantine fault-tolerant techniques are typically expensive, and the requirement for tolerating Byzantine failures in multiple system resources make it even more difficult. We are investigating engineering solutions that combine fault-tolerance and security techniques appropriately, and where graceful degradation is acceptable and the cost and quality of the solution is dynamically adjustable.

- *Coordination and management of adaptation*: Adaptive responses may involve changes in the application as well as changes in the way resources are managed. For a systematic implementation of a survivability strategy, this adaptive behavior needs to be separated from the application's functional behavior. We argue that the space between the application and the infrastructure is the appropriate place for it. Adaptative responses will usually involve coordination among multiple mechanisms that are also in the middle; these mechanisms will have various responsibilities ranging from resource management to system security. It may be the case that one adaptive response will preclude another. Some of these adaptive responses may be more expensive than others. One of the underlying issues involved here is the trade-off between adaptation and the application's performance.

3. Our Background and Recent Work

In our earlier work with QoS in distributed systems, we have developed individual resource management systems (AQuA, DIRM) [3, 2] that offer a certain desired quality to the QoS-aware application via an adaptive middleware called QuO [6]. This middleware allows the application to adjust itself when resource constraints do not meet the desired level. These applications display a minimal degree of inherent survivability. With the capability of introducing environment-awareness and adaptive behavior in distributed systems, we then started to look at survivability requirements of distributed applications. We presented our initial work towards this in an earlier paper[5] describing how integration with IDSs can lead to system agility and discussed various benefits of such integration. We have two ongoing projects in this area. One focuses on defensive strategies and how to build applications that incorporate defensive strategies. In the other, we are exploring intrusion tolerance by means of middleware coordinated adaptation that is unpredictable to the attacker.

4. Conclusion

Our ongoing work in adaptive middleware has led to the discovery that applications that are aware of changes in their environment can participate in their own defense, recognizing the symptoms of attacks and improving the detection and responses of security and defense mechanisms operating on their behalf. Traditional fault-tolerance techniques, if extended to support coordinated, malicious faults, appear promising to enhance the survival of applications. We are currently performing research to make fault-tolerance techniques suitable for the tolerance of intrusions in large-scale distributed systems. We are focusing on tolerating the symptoms of attacks; tolerating Byzantine failures that can be parts of malicious, coordinated attacks; and concentrating on unpredictable responses, both pro-active and reactive, that will prove difficult for intruders to exploit. In this paper have outlined our approach, and presented several key problems that we are currently investigating.

References

[1] L. Badger. Generic software wrappers. Internet URL http://www.pgp.com/research/nailabs/secure-execution/wrappers-overview.asp, 2000.

[2] BBN Distributed Systems Research Group, DIRM project team. DIRM technical overview. Internet URL http://www.dist-systems.bbn.com/projects/DIRM, 1998.

[3] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, D. Bakken, M. Berman, D. Karr, and R. E. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 245–253, October 1998.

[4] A. Ghosh. Sandboxing mobile code. Internet URL http://www.rstcorp.com/research/sandboxing/, 2000.

[5] J. P. Loyall, P. P. Pal, R. E. Schantz, and F. Webber. Building adaptive and agile applications using intrusion detection and response. In *Proceedings of the ISOC Network and Distributed Systems Security Conference*, Februrary 2000.

[6] J. A. Zinky, D. E. Bakken, and R. E. Schantz. Architectural support for Quality of Service for CORBA objects. *Theory and Practice of Object Systems*, 1(3):55–73, April 1997.