RC23765 (W0405-015) May 4, 2004
Computer Science

IBM Research Report

Elix0r: Cost-Effective Incident Response

Suresh Chari[1], Sudhakar Govindavajhala[2], Daisuke Nojiri[3], Josyula R. Rao[1], Michael Steiner[1]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]Department of Computer Science
Princeton University
35 Olden Street
Princeton, NJ 08544

[3]Department of Computer Science
UC Davis
1 Shields Avenue
Davis, CA 95616

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Elix0r: Cost-Effective Incident Response

Suresh Chari Sudhakar Govindavajhala Daisuke Nojiri
Josyula R. Rao Michael Steiner

Abstract

In this paper, we describe Elix0r, a system designed to plan proactively and respond automatically to security incidents such as fast moving intrusions and reported vulnerabilities. Elix0r attempts to contain such incidents while trying to minimize the impact to supported business processes. The containment actions are executed by a robust, flexible response infrastructure whose core is a rich and expressive scripting language designed explicitly for response. The system is designed to work in a wide variety of environments ranging from highly managed environments like DMZs to completely unmanaged networks.

1 Introduction

1.1 The Problem

Vulnerabilities in deployed computer systems and intrusions that exploit them are a major threat to enterprise networks and datacenter environments today. Such incidents are catastrophic, in the sense, that they immediately result in the interruption of critical business processes as well as cascading in the sense that they can spread and paralyze the entire target network. A recent example is that of the W32/Blaster worm which successfully infected approximately 300,000 machines on the Internet and then performed a denial of service attack against web servers distributing patches for the security vulnerability that the worm exploits. The Blaster worm is especially illustrative since the underlying vulnerability in the DCOM RPC service was known before the release of the worm.

When a system vulnerability is publicized or when activities indicative of an security intrusion are deduced, swift action must be taken to contain damage and to remediate the network so that critical business processes are not interrupted. Response mechanisms in practice today tend to be reactive, overwhelmingly manual and labor intensive, and largely ignore business process considerations. Effectively, system administrators scramble when notified of an incident with little automation to assist in their tasks.

This state of affairs is woefully inadequate for several reasons. Firstly, the damage to the infrastructure due to an intrusion such as the spread of a worm is so quick that a manual response is often too slow to be effective. Secondly and

more importantly, containing a security intrusion comprehensively is difficult to achieve by manually chosen response actions at the time of the incident. Even if this were possible, manual response actions tend to be error-prone and overkill and result in the unnecessary termination of network connectivity and/or server programs, which unduly penalizes unaffected systems and adds to the cost of response.

Thus, it is increasingly clear that any scalable response architecture should take advantage of proactive preparation to avoid scrambling at the time of the security incident and must automate as many aspects of analysis and execution as possible. Automation is perhaps the only way to ensure a timely and comprehensive containment response to vulnerabilities and security intrusions. With the high false positive rates of current intrusion detection systems, it is clear that the trigger for automated response must be manual.

While there have been some attempts[HM03, MM01] to automate response, these approaches ignore the impact of the response on the business process being realized by the target network. The implemented business process inherently assigns a value to each host element in the network and the value of the services it offers. A response action which terminates a service on a host or shuts down a host incurs a cost which is proportional to the value of these services/hosts. In the absence of such considerations, response actions can be correct but are not meaningful from a business perspective.

A very important requirement for automating response is a robust infrastructure that can be used to contain the impact of a vulnerability or intrusion quickly. While a number of network and systems management tools exist, there are very few languages, frameworks and tools that enable the execution of a wide spectrum of response actions securely and comprehensively. In the event of a security intrusion, such a response platform would be key to limiting the total number of hosts attacked and containing the damage.

In this paper, we describe Elix0r, a system designed to automate response actions, while optimizing the cost of response and ensuring that response actions are executed in a flexible and robust response framework.

1.2 Contributions

The Elix0r system has two main features which directly address the shortcomings of current proposals to automate response to vulnerabilities and intrusions:

- It attempts to use combinatorial optimization techniques to minimize the impact of response actions on the business process given the available control points in the target network.
- Response actions are executed in a framework comprising of a rich and expressive scripting language and a robust actuation platform.

In order to optimize the cost of response, we require that a value be assigned to the host elements in the network and to the services they offer. It is our belief that this valuation of hosts and services can be derived from the business process

supported by the target network. This is fairly straightforward in DMZs where it is much easier to quantify the value of elements. Alternately, our system can also work with qualitative assignment of values such as a high, medium or low style classification. The cost of a set of response actions is the resulting net loss in the value of the hosts and services in the target network. We assume that the discovery of all the hosts and services in the target network and the assignment of value from the business process is done periodically offline.

Elix0r makes no assumptions about what control points are available in the target network for response and does not mandate response capabilities for host or network elements. The cost of response will clearly be dependent on the availability of enough control points in the target network to actuate response actions. It is our aim to work in a wide variety of environments ranging from highly managed datacenter type environments where we could potentially have the fine grained ability to control or even reconfigure a service running on a host to completely unmanaged environments where the only response that can be taken will be on infrastructure elements such as switches and routers. Our optimization techniques will try to identify the best possible set of response actions, to contain a vulnerability or intrusion, given the control points actually resident on the target network.

A concept we find particularly advantageous in our design is that of a *target class signature*. This is a quick and succinct characterization of the class of hosts or services that are affected by the current vulnerability/intrusion. For instance, the target class signature for the W32/Blaster worm would be Win2000/XP machines running the DCOM RPC service. We envision that our system could be used for effective containment using successive refinement of target classes as more information about the affected class of machines becomes known.Target classes are also of use in proactive planning for response.

Elix0r does not address the issue of detection of vulnerabilities/ intrusions since there is a large universe of such tools readily available. Also, in the current design of our system the triggers for identifying and actuating response actions is manual through a target class signature given by the operator. Given the high degree of false positives in current detection systems, we believe that it is not feasible to have detection systems automatically triggering response actions.

When triggered, Elix0r will use the target class signature to identify an optimal set of response actions, given the current set of control points in the target network. We use combinatorial optimization techniques to minimize the cost of response actions. In a number of cases, we are able to reduce this problem of optimal containment to the graph-theoretic problem of computing the minimum cut[CLR90]. This optimization component will identify an explicit set of high level actions that need to be taken to contain this vulnerability/intrusion. Examples of such actions are disabling ports of a switch or router, reconfiguring filtering rules on switches and routers, terminating server programs, shutting down hosts, applying new local firewall rules etc. Since the minimum cut problem can be efficiently solved[CLR90], we can compute the optimal set of response actions for a number of cases. To effect such response strategies, we have architected and are prototyping a distributed response platform which provides an

abstract and coherent interface to the various previously identified actions and their platform dependencies. To support administrators in fail–safe and timely ad–hoc remediation actions, we also provide them with a simple yet powerful scripting language and shell. The key features of the language are the possibility to intelligently aggregate resources based on various characteristics and to execute operations jointly on the whole aggregate. We believe that the response platform component of our system is rich enough to cleanly express a large class of response actions and is a very valuable tool independent of what strategies are used in response.

Another important feature of Elix0r is that in a number of cases, responses can be planned proactively. For highly managed environments such as DMZs where elements have a high business value, we can proactively identify a set of prepared response plans for a number of target classes. This can be done by a wargaming-like offline simulation of available containment strategies as well as actuator placement to identify the best strategy for these target classes. For these cases, essentially the optimal response actions can be computed a priori. Upon receiving a trigger, one could directly jump to the deployment of such response actions. With similar war–gaming, we can also evaluate the most cost-effective selection and placement of actuators.

1.3 Related work

The closest related work [HM03, MM01] derives reactive strategies based on control-theoretic methods to act locally on host-intrusions. While they mention the possibility of using it offline, they seem to focus more on real-time planning. Furthermore, their current attack model is rather ad-hoc and they completely ignore the issue of sensors and their false positive and negative characteristics.

For the various parts of our preparation phase, we point out following related work. A somewhat similar use of network discovery tools for security is discussed in [VVZK02]. Automatically derived network information is exploited by ClearResponse from Psionic[1] in enhancing the filtering of alerts. However, none of these approaches exploit this information for improved response such as our target class signatures.

There are several research projects about defense systems against worms, but none has discussed business impact of protective actions. Moore, et. al. studied requirements for containing worms [MPS+03]. They compared signature-based filtering and black-list filtering. They compared several different ways to deploy signature-based filtering. Nojiri, et. al. proposed a peer-to-peer based defense system where defense systems exchange information about worm activities and filter traffic to prevent further infection [NRL03]. Their focus is on a large scale network such as the Internet, and their idea is based on simplified topology. Finally, previous work on cost functions covers [LFM+ar, WFFR01] and in particular [TK02]. The latter comes closest to our proposal. However, this is done only in the context of response and not taking into account the infection.

[1] `http://www.psionic.com/`

There have been several ideas on response in the literature. One of the earliest systems was Emerald [PN97]. However, their focus was on the aspect of global detection rather than response. There also has been work on taxonomies on response such as [CACP00]. Among the deployed solutions, CITRA [LCNS+02, SDB+02, SDW+01, SHS+01, SDS00] provides an infrastructure supporting global policies and coordination. However, their scope has been limited to DDoS. Furthermore, they do not deploy any planning phase to develop the strategies and have limited focus on survivability. SARA [LHO+01] is an extensible framework for coordinated response with focus on survivability. However, it also does not do any preparation and planning.

While there is a lot of work on a whole spectrum of response actions ranging from drastic actions such as blocking traffic to moderate response such as virus throttling [Wil02] or the system call delays of [SF00, Som02], there is hardly any work on measured responses minimizing disruptions in a global context.

2 Model

The goal of the Elix0r system is to protect a target network such as an enterprise network or a datacenter. Such a networked environment typically consists of a heterogeneity of host elements (such as personal computers, workstations and servers) connected by equally diverse network elements (such as switches, routers and gateways) that are typically connected to the Internet via firewalls. The design of Elix0r is intended to deal with both the case where the environment is fully managed, in that each host and network element is under the administrative control of a single entity as well as the more important case where the network has grown organically and it is likely that the host and network elements are in different administrative domains. Figure 1 shows a typical enterprise network topology layout. We see network elements (i.e. switches, routers, firewalls, hubs) and workstations, servers, personal computers, and the Internet. Between the two firewalls is a DMZ, where the mail server, the web server, and the DNS server are located.

2.1 The Resource View of the Network

We model such a networked environment as a graph of *resources* where a resource can be viewed as one of a host system, a service on the host system, a switch, a router etc. A resource is, informally, anything that provides value to the rest of the network and its users. In our model, a normally functioning resource can become *affected* due to software errors, misconfigurations etc. An affected machine can get *infected* when it is subverted by a security intrusion. Notification of such an infection can be obtained from an IDS. In some situations, it may not be possible to definitively ascertain the infection status of an affected machine. Such a machine is termed to be *suspect*. An affected machine that is neither infected nor suspect is termed to be *vulnerable*. Together,

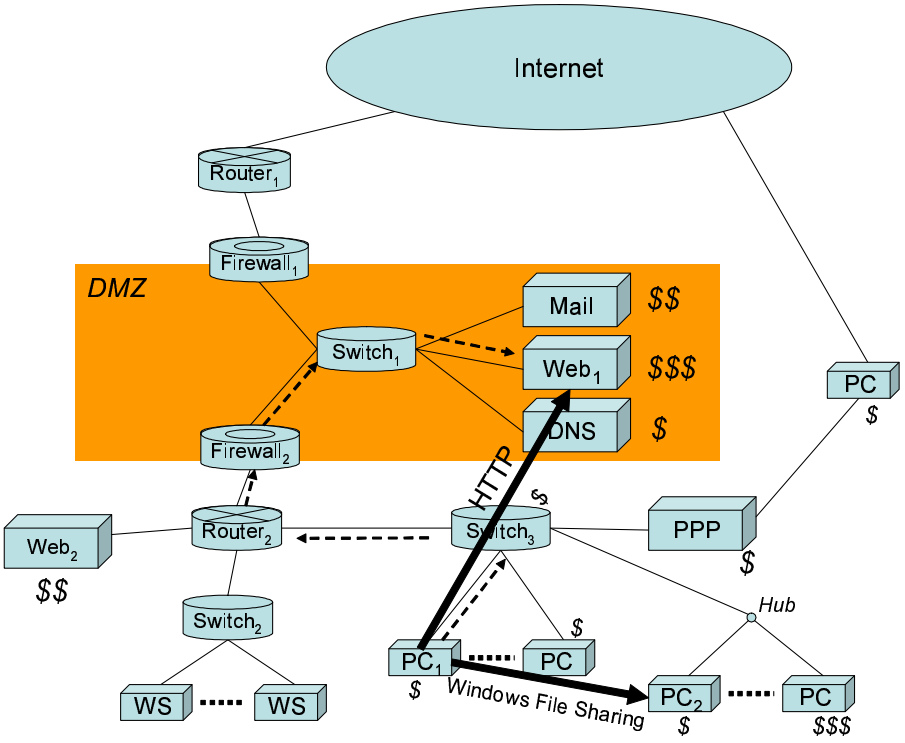

Figure 1: Representative topology

the classes of infected, suspect and vulnerable machines constitute the class of affected machines.

A key factor in the management of security incidents is the availability of *control points* in the network that enable an administrator to contain machines to either repair vulnerabilities or contain intrusions. The Elix0r system does not mandate specific response capabilities on any element; rather it works with existing response capabilities on each infrastructural element. For instance, a highly managed server may offer a fine-grained ability to control or even reconfigure a particular service running on the machine. On the other hand, an unmanaged host may offer no control points and thus, any response action affecting this machine must be done at the network infrastructural level. For instance, network switches such as the Cisco Catalyst family of switches offer a range of response possibilities ranging from simply turning a port off to very fine grained filtering of traffic on a particular TCP from a machine. The cost of containing an incident will directly depend on the amount of control we have on the elements in the topology.

2.2 Service Links

Most of the networked environments that Elix0r targets conceptually implement a *business process* of one kind or another. This business process is usually implicit in the network topology and infrastructure but can be made more explicit in other cases. A centerpiece of the Elix0r approach is to take advantage of business process considerations in preparing for and responding to notifications of vulnerabilities and security intrusions. In particular, we use the ideas of a *service link*, a *service link graph* and a *cost model* to capture salient aspects of the business process in implementing the Elix0r functionality.

A *service link* is a logical link directed from a client to a server to indicate that the client uses a service that the server provides. In Figure 1, we see a service link from pc_1 to web_1 which indicates pc_1 has access to web_1 using the http protocol. pc_1 has another service link to pc_2 using Windows file sharing service. Note that existence of a service link depends on the configuration of the network elements. For example, in Figure 1, if firewall_2 is blocking port 80 from pc_1, then pc_1 does not have a service link to web_1.

While business process dictates which hosts *need* to use which services, there are other links possible due to configuration of routers and switches. For instance, there might be a path available from a host A to B even though the business process does not mandate this. Since a security incident such as a worm infestation can propagate via this available path, our model will record a service link from host A to B.

A graph that comprises of the host and network elements as nodes and the service links as edges is termed the *service link graph*. If a target network offers services accessible from the Internet, we create a node W to represent the Internet and create service links from the W to the respective services. In our terminology, W is always considered to be suspect.

2.3 A Cost Model for Containment

The notion of a service link is a first step in capturing ongoing business activities in a network. To quantify the business value of these activities, we assign an asset value to each service link and to each host element in the network. Asset values quantify the importance of the asset from a business process perspective and can be based on a number of factors such as the criticality of the services a host provides, the sensitivity of data it stores etc.

Asset values of a service are based on the value of the service to the business process. For instance, a web server of a mail order clothing firm that supports e-business transactions from the Internet generates revenues at a rate that varies seasonally. At a first approximation, one can assign a asset value to the service link to represent the revenue that is generated on a particular day based on historical data. However, it may be difficult to assign values to service links such as the one between an employee's pc and a mail server. In this case, one can assign a qualitative value based on the degree of importance. For example, a service link from a customer service representative's pc to a mail server could be assigned a higher asset value than that of a service link from any other regular employee's pc that does not support such a crucial function. Any service link arising from existing paths through the network topology, which is not explicitly mandated by the business process, is assigned a zero value. The asset value of the service link represents the cost incurred by the business if response actions required the link to be terminated

Similarly, asset values are assigned to hosts. In our example, the back-end database server supporting the web server which stores details of payment information for mail order customers, would have an asset value proportional to the data that it stores.

In many security incidents, especially worm attacks, termination of service links provides an effective way of both containing infected and suspect machines and isolating vulnerable machines especially if the attack uses service linsk as a means of propagation. [2] Clearly, termination of a service link results in the loss of critical services in the network that affect the business process that it implements. In our model, the impact of terminating a service link is captured in terms of a cost, called the *primary cost*: it is the cost directly incurred by the business process if the link is terminated and is effectively the asset value of the service link described above.

Service link termination can be performed in several different ways. The actions that one can take will depend on the control points that are available on the network elements and the host elements on the service link. For example, in Figure 1, dotted arrows represent a infrastructural path from pc_1 to web_2. There are multiple ways to terminate this link: reconfiguring one of the switch, router and firewall on the path, or applying a new local firewall rule on web_1. On

[2] This is not necessarily true for contagious worms such as Nimda, though we do not discuss such worms in this paper. A contagious worm infects another host when a vulnerable host accesses the infected system when the victim download malicious data *i.e.* propagation is in the reverse direction of a service link.

the other hand, one has fewer choices to terminate service link from pc_1 to pc_2. If one has no other choices than disabling the port of a switch to terminate the link because of the lack of the capability of the switch, one is forced to terminate other service links. Some switches such as Cisco Catalyst series or Intel Media Switch family provide fine-grained filtering capability, hence, one does not need to affect other services to block the traffic from pc_1 to pc_2, while some switches do not provide such capability. We call this forced link termination due to the infrastructural constraints of network devices, a *secondary effect*, and the cost caused by secondary effect is called *secondary cost*. This is the collateral cost of terminating other service links in the process of terminating a given service link and directly reflects the amount of control we have on elements in the topology.

There are several other subtle complexities in the assignment of value and the computation of response cost which we omit here. For instance, we will ignore the service dependencies in the target network: A web server might depend on the services of an application server and a database server and might not work if either is terminated. This is a simple example of a conjunctive dependency. In the rest of the paper, we will just assume that the dependencies are disjunctive. Also, when computing the cost of response actions, we will ignore transitive costs, i.e., when a service is affected all the services which need this service are affected and so on. The complete details of value and cost modeling which extends prior work[TK02, VVZK02] will be deferred to a full version of the paper.

2.4 Target Class Signatures

Security vulnerabilities and intrusions can be characterized in terms of the characteristics of the *affected* target machines. We call this the *target class signature* for the security incident. Typically, it would be expressed in terms of the operating system and name and version of a vulnerable program. For example, the target class for the W32/Blaster worm would be Windows NT 4.0/2000/XP/ 2003 Server machines running the RPC DCOM interface. Similarly, the target class for the SQL Slammer/Sapphire worm is machines running SQL Server 2000 and Microsoft Desktop Engine 2000 on port 1434/UDP.

The key idea underlying the target class concept is that one can iterate over different target class signatures deployed in the network topology and prepare for the hypothetical eventuality that vulnerabilities in such a target class could be exploited in the future. This can be done by controlling deployments conformant with the risk to one's business process on such machines and ensuring the availability of a sufficient number of control points to contain such a target class should the threat materialize. In this way, the concept of target class is crucial to protecting against attacks with unknown signatures.

Once a target class has been identified, Elix0r forms a graph, called *target class graph* (see Figure 3). A target class graph is a directed graph, where the vertices denote hosts that belong to the target class and edges represent service links. Nodes are assigned their asset values while edges are assigned two non-negative values: primary cost and secondary cost as described in the previous

subsection.

3 Functional Description of the Elix0r System

In this section, we describe the functionality implemented by the Elix0r system. This functionality is implemented in two phases: the *offline* phase and the *online* phase. In the offline phase, the Elix0r system gathers detailed knowledge of the target network and the business process that it implements. This is intended to better prepare the infrastructure for the online phase. In the online phase, the Elix0r system, receives notifications about vulnerabilities/intrusions. Based on these notifications, it identifies the target class of machines affected and algorithmically determines the best strategy to respond to the vulnerability/intrusion that would cause minimal disruption to the business process. This strategy framed in terms of abstract high level actions is then implemented by mapping to low level response actions and orchestrating the execution of the actions across the infrastructure.

3.1 Offline phase

3.1.1 Discovery of Infrastructural Information

The Elix0r system uses existing tools to discover and gather as much information about the target network as possible. This includes information about all the host systems and network elements and the network topology. For each host system, Elix0r gathers detailed configuration information about the supported software and services such as the operating system and its version, running services, etc. Similarly, for each network element, Elix0r gathers information about the supported software.

Data discovered in this manner is indispensable from several perspectives. It enables (a) the discovery of the host system information and topology of the target network (b) the categorization of a host system as a member of a target class (c) the discovery of where more "valuable" host systems exist relative to other systems and (d) the discovery of control points in the infrastructure in terms of the response actions supported at host and network element and networking pathways that can be used to actuate them.

Elix0r relies on a number of available tools such as IBM Tivoli NetView [Net] or HP OpenView [Ope] considerations that can be used for discovering and maintaining such information.

3.1.2 Service links

Using the information collected during the discovery process and business process information, the Elix0r system can create a detailed graph of the services that each host uses and provides. This information is then used to create a *service link table* (see Figure 2). Since the existence of a service link depends on the configuration of the intermediate network elements and the service provider,

Service provider

Service consumer HTTP	Internet	Web_1	$Mail_1$	Web_2	PC_1
Internet		→Rt1→Fw1 →Sw1:0→	(NA)	(NA)	(NA)
Web_1	→Sw1:0→Fw 1→Rt1:0→		→	→Sw1:0→Fw 2→Sw2:1→	(NA)
$Mail_1$	→Sw1→Fw1 →Rt1:0→	→Sw1:1→		→Sw1:1→Fw 2→Sw2:0→	(NA)
Web_2	→Sw2:0→Fw 2→Sw1:1→Rt 1→	→Sw2:0→Fw 2→Sw1:1→			(NA)
PC_1	→Sw2:0→Fw 2→Sw1:1→F w1→Rt1→	→Sw2:0→Rt2 →Fw2→Sw1: 1→	(NA)	→Sw2:0→Rt2 →	

Figure 2: Service link table for HTTP

this information is used to create this table. In addition to depicting the services used in a network, the table enables one to quickly identify infrastructural network elements underlying each service link and how they are shared between service links.

Once the service links in a target network are clear, the next task is to create the service link graph introduced in Section 2 and to assign asset values to the host systems and service links depending on the business process that is being supported. This asset value assignment has to be done carefully and it is based on the cooperative work of the business line personnel, system administrators and individuals responsible for each server, workstation or personal computer in the target environment. This assignment is the basis of computing the primary costs.

Next, we compute the secondary costs. For each service link, we use the service link table to determine if there is a network device on the infrastructural path which provides filtering capability which is sufficiently fine-grained so that other service links using the network device are not affected. If such a device is found, the secondary cost of terminating the service link is zero. The Elix0r system records that termination of the service link can be done by filtering of the network device found and that the secondary cost of terminating the service link is zero. If no such network device is found, then the secondary cost is determined as follows: For each network device on the physical path:

1. Determine the filtering that terminates the given service link.

2. Compute the total primary cost of the other service links affected by this choice of filtering

At the end of the exercise, pick the network device and filtering that yields the smallest cost and use it as a secondary cost for this link. The Elix0r system records that termination of the service link can be done by filtering of the network device found and that the secondary cost of terminating the service link is the minimal cost computed.

Note that in the second step, one needs to find all other service links affected by choosing a filtering at a certain network element. Finding such service links depends on the type of the filtering. For example, if the filtering is done by disabling a port of a switch, one needs to look for all occurrences of the same port of the same switch in all the cells of the service link table. If the filtering is based on an IP address, all the service links of corresponding IP address are to be affected. Therefore, service link table must store information which enables the response system to produce the list of the links affected by a certain filtering deployment.

3.2 Online phase

We now describe each step of how the Elix0r response process from identifying affected hosts to containing and remediating them.

3.2.1 Detecting a vulnerability/intrusion

The first step in the response process is the receipt of a trigger that either a vulnerability has been discovered in a host system deployed in the target network or that an intrusion has been detected in the target network. The former can be obtained by running periodic scans using scanners such as the Microsoft Baseline Security Analyzer [Mic], Nessus [Nes], etc or even by out of band notification via an organization like CERT/CC. The latter can be obtained from network and/or host-based intrusion detection sensors such as Cisco's IDS [Cis], ISS's Proventia [ISS], Snort [Sno] or Symantec's Intruder Alert [Sym]. The Elix0r system assumes that it is provided with such a trigger and details of the exact method of vulnerability/intrusion detection and its accuracy (such as false positives etc) are not part of the Elix0r project.

3.2.2 Identifying a target class

Once a trigger is received, the Elix0r system first needs to identify the target class signature for the affected machines. A very specific signature enables the identification of the classes of infected, suspect and vulnerable hosts accurately. This allows Elix0r to focus its response actions on containing the infected and suspect machines and isolating the vulnerable machines leading to a lower cost of response.

A more granular signature can lead to a higher cost of response. For instance, anomaly detection at network traffic level may only provide limited information such as the TCP/UDP port in the signature. This forces us to protect the complete class of affected machines, and hence, the cost of the first response can

be unnecessarily high. Over time, the target class of the vulnerability/intrusion can be refined as more intelligence becomes available. This can be used to iteratively refine the response strategy to contain and separate the class of infected and suspect machines from those that are only vulnerable. For instance, initial reports of the Code Red worm indicated that it targeted Windows platform. Over time, as the worm was analyzed, its target class was refined to Windows NT or Windows 2000 platforms that host the Microsoft IIS server.

3.2.3 Constructing a target class graph

Once a target class is identified, Elix0r constructs a target class graph by looking at service link tables.

If a vulnerable host is located in the same LAN segment as a infected or suspect host and we are unable to tell whether it is already subverted, we should assume that it is. We also need to sort out all the *dependencies* among secondary effects of the target class graph. The notion of a dependency represents a list of service links which has a secondary effect on the same edge. For example, if edge e_1 and e_2 has a secondary effect on edge e_3, e_1 and e_2 are both in the dependency list of e_3. This notion is necessary to avoid double counting the secondary cost of e_3 when a response action terminates both service links e_1 and e_2. A formal definition of dependency and its usage are found in Section 4.

3.2.4 Illustrative example

Elix0r uses the target class graph to search for the most cost-effective containment. The target class graph captures the set of affected nodes. In the event of a worm infestation, the subsets of infected, suspect and vulnerable machines may also be known and the task is to find an optimal containment that would contain the infected and suspect machines to prevent the spread of the infection and to protect the vulnerable machines. In the case of a vulnerability notification, all the machines in the target class graph are considered vulnerable and the task is to isolate them from the Internet, denoted by the node W in our model (see Section 2) as W could be the source of exploits for the vulnerability. In either case, the containment should be done in a near-optimal manner keeping the cost model in mind.

To execute a containment, the Elix0r system will cut service links crossing the perimeter of a set of nodes. For containing a set of infected and suspect hosts, it is enough to cut the outbound service links and for a set of vulnerable machines, it is enough to cut the inbound service links. The assumption underlying such a containment strategy is that the intrusion uses service links to propagate. The actual containment itself can be done by physically reconfiguring network elements which appear on each service link. At the end of the containment exercise, all the nodes (hosts) and edges (service links) outside the infected and suspect containment set will be safe from the intrusion.

The cost of a containment is obtained by adding the the asset values of all nodes in the infected and suspect set, the primary costs of all *internal* edges,

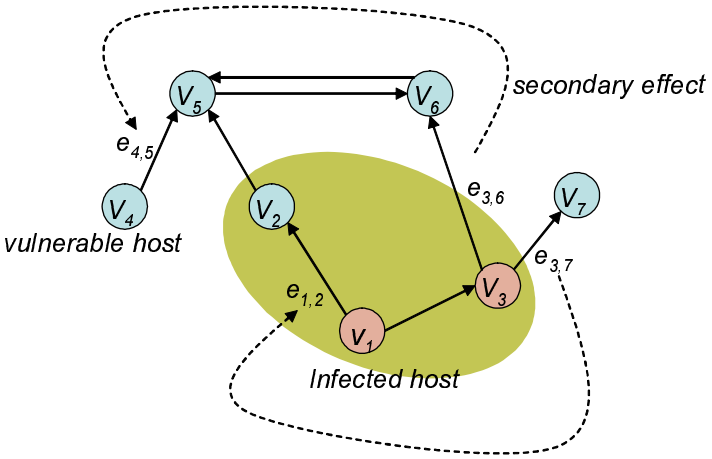

Figure 3: A sample target class graph

that is edges with both ends in a containment, the *incoming* edges to and the *outgoing* edges from the infected and suspect set, and the secondary costs of the *outgoing* edges. A formal description of cost computation is found in Section 4.

We informally illustrate cost computation with an example. In Figure 3, we see containment $\{v_1, v_2, v_3\}$ and initially infected nodes v_1 and v_3. In the containment, the edges $e_{1,2}$ and $e_{1,3}$ internal and edges $e_{2,5}$, $e_{3,6}$ and $e_{3,7}$ are outgoing edges. Since $e_{3,6}$ has a secondary effect on edge $e_{4,5}$, we need to add its secondary cost which is equal to the primary cost of $e_{4,5}$ in this case. Also, $e_{3,7}$ has a secondary effect on edge $e_{1,2}$, but we should not count this secondary cost since $e_{1,2}$ is in the containment and its primary cost is already included. Let a_i be the asset value of vertex v_i. Let $p_{i,j}$ and $s_{i,j}$ be the primary cost and secondary cost of edge $e_{i,j}$, respectively. The total cost of this particular containment is

$$(a_1 + a_2 + a_3) + (p_{1,2} + p_{1,3} + p_{2,5} + p_{3,6} + p_{3,7}) + s_{3,6}.$$

3.2.5 Searching for the optimal containment

After a target class graph is formed, the Elix0r runs an algorithm to search for the optimal containment. Because of the complexity of cost computation as shown in the previous section, we divide target class graphs into three groups: clean, tidy, and dirty. The system first identifies which class the graph belongs to, then runs the corresponding search algorithm. Definitions of concepts introduced informally in this section and details of the various algorithms are presented in Section 4. The containment search algorithms return the list of service links to be terminated.

3.2.6 Orchestrating response

Once Elix0r determines the set of service links that need to be terminated, it uses a orchestration platform to enforce response. This orchestration platform has also been designed so that it can be used as an independent tool by system administrators and service providers to script response strategies.

3.2.7 Extending the Off-line Phase: Wargaming, Singleton Strategy

In a static and completely managed environment such as a data center and a company's DMZ, the Elix0r systems can be used to war-game potential vulnerabilities and intrusion scenarios in the off-line phase. In such an exercise, the Elix0r system will simulate security incidents by randomly selecting a target class of machines that are under attack and will then compute the minimal cost of response. If this cost is not acceptable, the environment can be reconfigured to perhaps permit more control points with refined filtering capabilities till the cost of response becomes acceptable.

In many enterprise networks, organic evolution of the networks, has prompted network administrators to effectively adopt a tree structure for their networks where user machines are typically at the leaves of the tree. In such situations, it is quite often possible to contain each individual user machine by disabling the LAN drops to the physical room where it is located. We call such a containment strategy, a *singleton approach*. For such networks, the Elix0r system would discover and maintain information of the location of user machines and their associated LAN drops in the off-line phase. This can then be used at the time of a security incident to contain the affected machine.

4 Containment Strategies

In this section, we outline strategies to identify specific actions to contain security incidents such as vulnerabilities/ intrusions. At a high level, this is an optimization problem of choosing a set of actions which minimize the cost of containment where the constraints are the actual response capabilities of the elements in the target network. In a number of cases, we are able to reduce the problem of optimal containment to the graph-theoretic optimization problem of computing the minimum cut of a graph. We do not have a polynomial time algorithm for the most general case, but we present a few heuristics for this case. Section 4.1 describes mathematical preliminaries to recast the containment problem in a graph theoretic setting, define containments, the cost of containments and a formal statement of the optimization problem. In Sections 4.2.1 through 4.2.3 we consider a set of approximations to the general problem. For some of the cases, we present optimal solutions by reducing this to the minimum cut problem. In the most general case, we present a heuristic to approximate the optimal solution.

4.1 Mathematical Preliminaries

We begin with definition of the *resource model graph*: a weighted directed graph with node set as the set of all host resources and service links as edges.

Definition 4.1 *The* resource model graph is a directed graph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V} = \{\hat{v_1}, \hat{v_2}, \ldots, \hat{v_N}\}$ and $\hat{E} = \{\hat{e_1}, \hat{e_2}, \ldots, \hat{e_M}\}$ the set of all hosts and service links respectively. Each vertex $\hat{v_i}$ is assigned a weight a_i. Each edge $\hat{e_i}$ has two weights: primary cost p_i and secondary cost s_i.

As discussed before, the primary cost of the edge is the direct cost of terminating that particular service link whereas the secondary cost is the indirect cost incurred when an action is taken to terminate this service link. The first step in containment is to identify a target class graph defined as follows:

Definition 4.2 *A target class graph $G = (V, E)$ is a subgraph of $\hat{G}$ where nodes $V = \{v_1, v_2, \ldots, v_n\}$ are the hosts that belong to a certain target class, and the edges $E = \{e_1, e_2, \ldots, e_m\}$ are the service links found among $\{v_1, v_2, \ldots, v_n\}$.*

A containment is defined as a collection of hosts which are to be isolated from affecting the rest of the hosts in the target topology.

Definition 4.3 (containment) *Let I denote the set of nodes in G which are already found affected*[3]*. A containment C is a set of nodes of graph G, with all the affected nodes in it, namely, $I \subset C \subset V$.*

We need to formally capture all the costs incurred in the containment of the incident. Given a containment C, we group edges depending whether they are in a containment or on the boundary: $\theta(C)$ denotes the set of edges with both endpoints in C, $\delta(C)$ denote the set of edges with the tail end in C and the head end in $V \backslash C$. Note that these sets of edges have different impact on the total cost of the containment since we need to explicitly cut the service links in the set $\delta(C)$ whereas we may choose not to cut the links in $\theta(C)$.

There are several other costs to be accounted for: All the hosts in the set C will be contained and the services running on these nodes will be corrupted or lost. Thus, the cost must include the value of service links in the resource model graph with the tail end in the vertex set $\hat{V} \backslash V$ and the head end in the set C. We denote this set of edges $\lambda(C)$.

Using these definitions, we can define the cost of a containment. Informally, the total cost of containment C, denoted $\Gamma(C)$, is the sum of the primary costs of the edges in the set $\delta(C) \cup \theta(C) \cup \lambda(C)$, the secondary costs of the edges in $\delta(C)$ minus the sum of all the double-counted secondary costs. This follows directly from the definition of primary and secondary costs. As discussed before in Section 2.3 there can be overlap in the secondary costs of terminating links which is captured formally in the following definition.

[3]As mentioned, I always contains the special node W representing the Internet. In this section, we use the term affected to denote infected or suspect machines.

Definition 4.4 *Given a containment C, a dependency α is a maximal subset of edges in $\delta(C)$ which have a secondary effect on a common edge of $\hat{G}$. This common edge is dented e_α and its primary cost p_α.*

Let $D = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ be the set of all dependencies in the target graph. Let E_C denote a set of all the edges relevant to a containment C, that is, $E_C = \delta(C) \cup \theta(C) \cup \lambda(C)$. For a dependency α_i, if edge e_{α_i} is in E_C, then its cost has already been counted in the primary cost terms. Thus the secondary cost term in the informal definition of the cost of containment C counts the cost $|\delta(C) \cap \alpha_i|$ times. If the edge e_{α_i} is not in E_C we have double–counted its cost $|\delta(C) \cap \alpha_i| - 1$ times. Thus, the double–counting of secondary costs is given by the formula:

$$\varepsilon(\alpha_i, C) = \begin{cases} |\delta(C) \cap \alpha_i| \cdot p_{\alpha_i} & \text{if } e_{\alpha_i} \in E_C \\ (|\delta(C) \cap \alpha_i| - 1) \cdot p_{\alpha_i} & \text{if } e_{\alpha_i} \notin E_C \end{cases}$$

Formally, the total cost of containment C using:

$$\Gamma(C) \triangleq \sum_{v_i \in C} a_i + \sum_{e_i \in \delta(C) \cup \theta(C)} p_i + \sum_{e_i \in \delta(C)} s_i - \sum_{\alpha_i \in D} \varepsilon(\alpha_i, C)$$

The formal statement of the optimal containment problem follows directly:

Definition 4.5 (optimal containment problem) *Given a resource model graph $\hat{G}$, a target class graph G, a set I of hosts known to be affected, and set of dependencies D, find the minimum cost containment $\bar{C}$.*

4.2 Algorithmic strategies

With the formalism used to identify the totality of all costs associated with a containment, we present algorithms which find an optimal containment for certain special cases. While we have no polynomial time algorithm for the most general case, we present heuristics which seem to work well.

4.2.1 Special Case: No secondary effects

We first consider the special case when there are no secondary effects in terminating service links. In the formal notation, $s_i = 0$ for all edges $e_i \in G$, and the dependency set D is empty. This is representative of the case when we have a fine grain control of the hosts and infrastructure elements so that service links can be precisely terminated without affecting any other hosts or service links. This could be realized for example if we can precisely filter a pair of hosts and a particular port on switches, firewalls and routers.

In these graphs, it is straightforward to see that the smaller the containment is, the cheaper the total cost. The monotonicity can be seen by noting that there are no subtractions due to overlapping secondary effects. Thus the most optimal containment is to precisely quarantine exactly the known initial set of affected hosts. We defer a formal proof to the full version of the paper.

Theorem 4.6 *If $\langle G, D\rangle$ has no secondary effects or dependencies then $\bar{C} = I$*

4.2.2 Special Case: No dependencies

Next we consider the case when terminating service links may have secondary effects but there are no dependencies, *i.e.*, the secondary effect of terminating any link does not intersect with the secondary effect of terminating any other link. In the mathematical notation of the previous section, the dependency set D is empty. An example of such a scenario is when two different servers, say, a mail server and a DNS server are co-located and connected jointly to a port on a switch through a hub. Switching off the port on the switch because of a vulnerability in the mail server would also disable the DNS server. Thus there is a secondary effect to containing the mail server. Assuming that the rest of the topology can also be finely controlled close to the affected host, there are no dependencies. For this case, we are able to find an optimal containment by a reduction to the standard graph theoretic mincut problem defined as follows:

Definition 4.7 (mincut problem) *[CLR90] Given a directed graph $G = (V, E)$, special vertices s and t, a cut C is a subset of vertices V such that $s \in C$ and $t \notin C$. The cost of the cut is the sum of weights of all edges with the tail end in C and the head end in $V \backslash C$. The mincut problem is to find a minimum cost cut.*

While a containment of the target class graph is also a cut of the graph, there are key differences between the two problems: In the optimal containment problem, nodes are weighted, the cost of the containment includes edges with both ends in the cut and edges can have secondary costs which are counted only if they are cut edges.

We describe an algorithmic transformation which converts the optimal containment problem to the mincut problem. This can essentially be done in linear time so the complexity of computing the optimal containment is essentially the same as that of the mincut problem.

The transformation can be done in purely graph-theoretic terms. Let e_{ij} denote an edge from vertex v_i to vertex v_j. Figure 4 shows the transformation pictorially. The following are the algorithmic steps for the transformation:

1. Define 2 new vertices s and t in the graph.
2. For each vertex $v \in I$, draw an edge from s to v with weight ∞.
3. For each edge $e_{ij} \in E$, create two new vertices x_{ij} and y_{ij} in the graph. Assign weights to the edges as described in Figure 4.

Let G' denote a conversion of original graph G. Given a containment in G, note that there is exactly one possible corresponding cut C' on G' with a finite weight: From Figure 4, if v_i is in C, x_{ij} must be in C' because of the infinite weight on edge between v_i and $x_{i,j}$. Also, if v_j is in C, $y_{i,j}$ must be in C'.

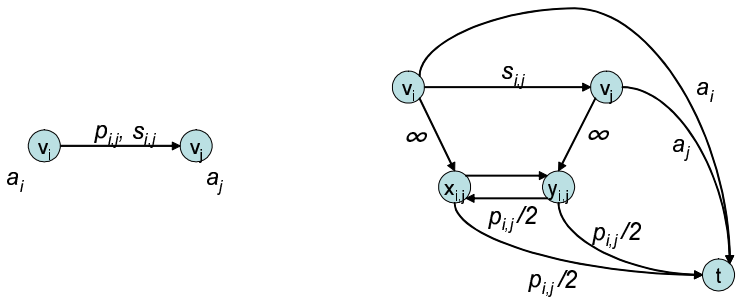

Figure 4: The transformation from the original graph to the converted graph. a_i and a_j are the node costs and p_i and p_j are the primary and secondary costs.

Similarly, one can argue that given a finite weight cut of the converted graph, there is only one finite weight containment of the original graph. The following can be proved easily and proofs are deferred to a full version.

Lemma 4.8 *If C is a containment on G, corresponding cut C' has the same cost on G' as the containment C in G.*

Theorem 4.9 *If $\langle G, D\rangle$ is a target class graph with no dependencies and cut C' is minimum on the converted graph G', corresponding containment C is optimal on G.*

There are a number of extensions of this case with no dependencies that can similarly be reduced to the problem of finding a minimum cut in a graph, the details of which are deferred to a full version.

4.2.3 General Case

In the most general setting, terminating service links has secondary effects and there are dependencies between the secondary effects of terminating links. We do not have a polynomial time algorithm to find the optimal containment and in fact, we conjecture that this might be NP-complete. Here, we present a simple heuristic which produces a local optimum but is guaranteed to terminate in polynomial time.

One-hop search This is a simple heuristic which does not guarantee a global optima, but it runs in polynomial time. Starting with the smallest containment C_0, which is equal to the set I of nodes known to be affected, the containment is gradually extended in the direction of the edge with nonzero secondary cost. Extension is done by putting a node to which the current containment has an edge with a secondary cost. If an extended containment has a smaller cost, this extended containment is kept and used to search for a better containment. Therefore, containment extension continues until there is no secondary effect on outgoing edges. Let $C_0 \rightarrow C_1 \rightarrow \cdots \rightarrow C_x$ denote this progression. Note

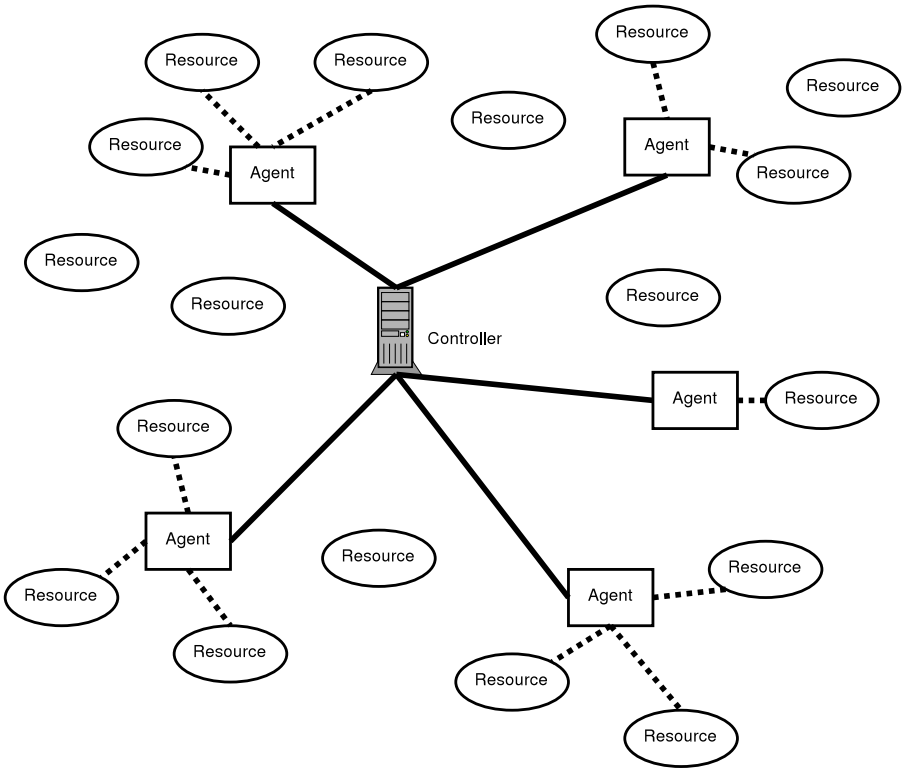

Figure 5: Architecture overview

that the C_x is not necessarily the optimal containment since there are cases when the greedy heuristic can choose a bad-move while extending the current containment. The detailed description of this simple heuristic and the associated analysis is deferred to a full version of the paper.

Using the algorithms in this sections, we can identify precisely which service links should be terminated to minimize cost of containment. These can be automatically executed using the response platform described in the following section.

5 A Platform for Orchestrating Response

5.1 Overview

As shown in Figure 5, the system consists of a central controller and multiple distributed agents. The agents provide the basic response action primitives whereas the controller coordinates these agents to provide for global response.

Each agent controls a set of resources providing control-points. It interacts with them in a resource-specific way, e.g., through an API or command-line

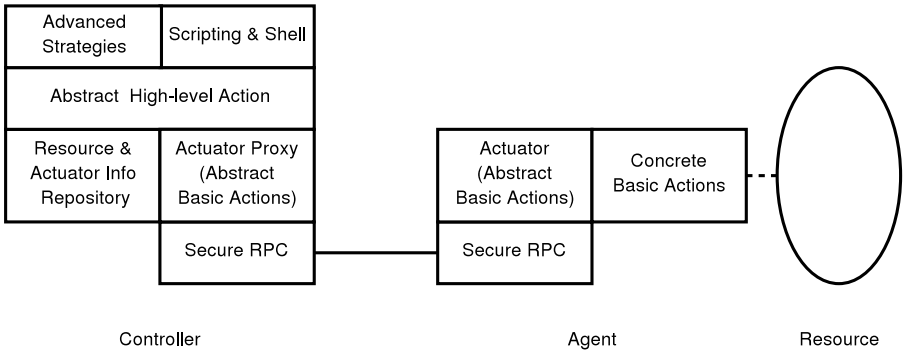

Figure 6: Architecture layering

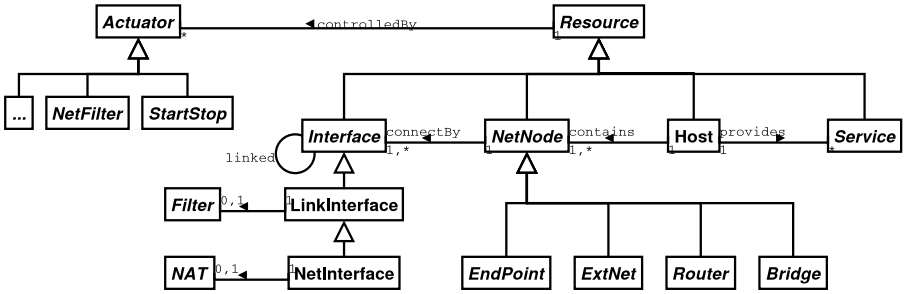

Figure 7: Relation among important classes in resource model

tools if agent and resource co-reside or through resource specific protocols such as SNMP [CMPS02] otherwise. The agents provide a unified and abstracted actuator interface to the corresponding basic response actions based on a robust and secure RPC service. This is depicted on the right-hand side of Figure 6.

As previously noted, in many if not most situations it is unrealistic to assume that the administrator owns and directly controls all resources, i.e., as shown in Figure 5 a number of resources will have no associated agent which can control them or query them about their state. However, for effective response these resources and their state has to be considered as well. Therefore, the controller will try to discover the existence and state of all network resources and actuators, manually as well as with tools such as IDD [GD02], describe them in a simple abstract resource model (see Figure 7 for a glimpse on this model) and maintain a corresponding repository.

When response is necessary, this resource database can be used to identify affected resources, e.g., based on target class signatures. It also assists in figuring out whether (and how) these resources can be directly controlled through actuators or whether response has to be indirect, e.g., because there is no directly controlling actuator or the actuator cannot be trusted as it is co-located

with an infected resource.

The resource model also defines in an actuator interface hierarchy the set of abstract basic actions previously mentioned. Examples for these abstract basic actions are the `StartStop` actuator which allows to enable or disable a resource such as a port (`LinkInterface` resource) on a switch or program (`Service` resource) on server, or the `NetFilter` actuator allowing to filter network traffic. Furthermore, the controller provides means to combine these basic actions into sets of high-level actions. In particular, we exploit the knowledge base of the resource model to allow for intelligent aggregation of resources and the combined execution of a set of actions on them. The aggregation can be done based on the characteristics of their underlying software and hardware, i.e., target class signatures; based on their business functionality, e.g., dependencies among service links; or topological information, most notably the identification of the set of choke points which provide actuators to isolate a set of resources.[4]

The goal of our infrastructure is two-fold: It certainly should enable the programming of (advanced yet involved) strategies such as the ones presented in the previous sections and their integration into an overall autonomic defense system covering also intrusion sensors, vulnerability scanners and/or patch-management systems. However, it should also enable the system administrator to manually effect response actions in an ad-hoc, timely and fail-safe fashion. While implementing the previously described components in a high-level programming framework, Java™ in our case, provides an excellent basis to achieves the first goal, it does not yet handle the second goal. To address the second goal, we extend the framework with a simple scripting language and shell which we describe in the following.

5.2 Scripting Language

The major operation for the system administrator to manually effect response is to enter, in the interactive shell, actions of the form "α `on` γ" where γ is a group of hosts and α is the commands to be executed on each of the host. In the following two sections we will focus on how, on the one hand, such host groupings can be defined, and, on the other hand, what kind of commands we allow. For the exact grammar of the language we refer the reader to Appendix A.

5.2.1 Commands and command grouping

Commands can be either basic or compound. The basic actions are abstract basic actions as mentioned in the previous sections. More concretely, we currently support:

`filter` *`parameters`* This action is used to block network connections based on various criteria like initiator and responder of the connection, the network protocol and the initiator or responder ports.

[4] An important factor in determining choke points is to assess the trustworthiness of the affected resources, e.g., for non-infected resources a collocated actuator is a fine and desirable choice whereas for infected resources such an actuator is an inappropriate choice.

`stop` *parameters* This action halts the host or the program or service identified by the optional argument.

`raise_alert_level` This action is used to increase the logging and alert level of the host.

We allow basic commands to be grouped by the operators `and` and `or` to form compound commands. The semantics are similar to the corresponding operators && and || in the C programming language.

5.2.2 Host groupings

The main power of the language is the ability to perform response actions on a (large) group of nodes which are aggregated using various criteria, e.g., to quickly identify all vulnerable nodes.

The language allows to define groups based on hostnames, IP addresses, subnetworks (e.g., `192.168.76.0/24` would identify all machines on the Class C network 192.168.76.0 and `Extranet` refers to machines external to the administrative domain) and also to invoke external programs such as network worm scanners to us feed with hosts. It also allows to compose groups based on the standard set operators on group. The most powerful two operators to group resources, however, are the following:

`select from` *hosts* `where` *cond* This selects the set of hosts from *hosts* which match the target class signature *cond*. Condition can be general predicate logic terms with predicates based on operating system, programs and services running on a host, their versions as well as actuator capabilities provided by that host.

`chokepoints of` *hosts* Applying the `chokepoints of` operator on a group *hosts* returns the closest group of actuators which allow to control traffic from and to the input group.

5.2.3 Execution and undo of actions

Given a command `command`, instantiating it on the group results in executing `command` on each of nodes in the group. This action execution returns a handle that can be used to identify the action should it be required to be undone. This handle can be queried to check if the action was successfully executed. An action is successful if and only if the underlying command is successful on all its nodes. While we do not attempt to provide atomicity of the actions, we make sure to first determine that the command is possible on all nodes based on information in the resource repository and do a best effort to make commands succeed.

We need not overemphasize the importance of having an `undo` operation in responding to a worm attack. The response time available to an administrator is very short and he is probably under huge stress, should he make a mistake. Providing an efficient undo capability gives the administrator the freedom to

```
# CERT advisory CA-2002-17.

 Vuln1 := select from intranet where
    program = apache version=1.2.2;
 Vuln2 := select from intranet where
    program = apache 1.3 <= version
    <= 1.3.24;

 Vuln := Vuln1  union  Vuln2;
 Infected := ‘network_worm_scanner‘;
% Set of vulnerable and infected nodes resp.

% Choke connections from  infected nodes.
 filter Infected as initiator on
    (chokepoints of Infected);

% Now, protect vulnerable nodes.
 Remaining := Vuln;

% Stop apache servers where we can
 StopSet := select from Remaining
    where capability stop program = apache;
 stop program = apache on StopSet ;


% update firewall rules on vulnerable nodes,
% if possible
 Remaining := Remaining diff StopSet;
 FilterSet := select from Remaining  where
    capability filter responderPort = 80 ;
 filter responderPort = 80
    FilterSet as responder on FilterSet;


% Filter connections on chokepoints
% for the remaining nodes
 Remaining := Remaining diff FilterSet;
 ChokeNodes :=  chokepoints of Remaining;
 filter port = 80
     Vuln as responder  on ChokeNodes;

% Raise alert level on vulnerable nodes.
  raise_alert_level on Vuln;
```

Figure 8: Responding to Apache vulnerability. CERT advisory CA-2002-17

err on the side of caution in case of an attack, like shutting down suspicious services, safe in the knowledge that undoing his actions is easy.

5.2.4 Examples

In June 2002, CERT had issued an advisory regarding a serious vulnerability in the popular web server Apache [Vula]. The vulnerability was in the handling of certain chunk-encoded HTTP 1.1 [FGM+99] requests that may allow remote attackers to execute arbitrary code. To illustrate how a system administrator could have responded to this incident with our tool, we give a sample script for this scenario in Figure 8. The script identifies first vulnerable and infected nodes and then tries various containment steps with the least intrusive first until all options are exhausted. For a second usage example of the language, see Appendix B.

5.3 Discussion and Outlook

We've built a first prototype of the previously described architecture. The core of the controller is written in Java™. The scripting language is based on this Java core and the JavaCC compiler compiler. The agents are all written in Perl and the secure RPC layer is currently simulated using ssh. As both tools are available on a wide variety of platforms, e.g., through cygwin on Windows, this does not limit the platform independence much and allows for faster prototyping than writing everything in Java.

Security of such a platform is obviously paramount. With ssh we get decent

confidentiality and integrity protection. As it is also crucial to timely deliver response actions even under attack, availability is also an important security requirement. While we do not yet address it, we are exploiting network reconfigurations as one way to address that issue. To limit the power of corruption of agents, the agent runs as an a priori unprivileged user with separate ssh credentials and basic concrete actions are selectively enabled with appropriately restricted additional privileges using sudo. Corruption of the controller is somewhat mitigated by the fact that no commands can loosen the security policy in terms of confidentiality.

The language may appear to be somewhat simple and verbose. However, this is intentional as the simplicity and the additional redundancy contribute greatly to its fail-safety. Given the potentially disastrous consequences of improper usage and the increased likelihood of mistakes given the pressure of dealing with security incidents, this is an essential design goal. The interactive shell is currently very limited. However, we expect that its utility can be enhanced by adding features such as cmdline-editing, history and command-completion for keywords, program, services, and versions based on querying the underlying resource repository.

In above discussion, we have presented only three abstract actions. While they are certainly crucial to an effective response, we are also exploring other useful abstract actions such as traffic throttling, restarting service nodes at lower levels of service which, for example, do not require change in persistant state, etc.

6 Conclusions

We proposed an automated intrusion response system which produces appropriate actions timely and correctly with minimum impact on business. We used target class signatures and service links to form a logical representation of the target network. This abstraction of the target network helps us search for the optimal containment efficiently by using graph theoretic algorithms. A containment search algorithm for dirty graphs will be developed in future work.

References

[ACM02] ACM. *18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, December 2002.

[CACP00] Jr. Curtis A. Carver and Udo W. Pooch. An intrusion response taxonomy and its role in automatic intrusion response. In *Proceedings of the first IEEE Information Assurance and Security Workshop*, West Point, NY, USA, June 2000.

[Cis] Cisco. IDS. `http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/index.shtml`.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1990.

[CMPS02] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and applicability statements for internet standard management framework. Internet Request for Comment RFC 3410, Internet Engineering Task Force, December 2002.

[DIS01] *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2001)*. IEEE Computer Society Press, June 2001.

[FGM+99] Roy T. Fielding, Jim Gettys, Jeff Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Internet Request for Comment RFC 2616, Internet Engineering Task Force, June 1999.

[GD02] Dieter Gantenbein and Luca Deri. Categorizing computing assets according to communication patterns. In E. Gregori et. al., editor, *Advanced Lectures on Networking : NETWORKING 2002 Tutorials*, volume 2497 of *Lecture Notes in Computer Science*, pages 83–100. Springer-Verlag, Berlin Germany, May 2002.

[HM03] Vu A. Ha and David J. Musliner. Balancing safety against performance: Tradeoffs in internet security. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36)*, Big Island, Hawaii, January 2003.

[ISO96] Information technology — Syntactic metalanguage — Extended BNF. Technical Report 14977, ISO/IEC JTC1/SC22, 1996.

[ISS] ISS. Proventia. `http://www.iss.net/products_services/`.

[LCNS+02] W. La Cholter, P. Narasimhan, D. Sterne, R. Balupari, K. Djahandari, A. Mani, and S. Murphy. IBAN: intrusion blocker based on active networks. In *Proceedings DARPA Active Networks Conference and Exposition (DANCE 2002)*, pages 182– 192, San Francisco, CA, USA, May 2002.

[LFM+ar] Wenke Lee, Wei Fan, Matthew Miller, Sal Stolfo, and Erez Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, to appear.

[LHO+01] Scott M. Lewandowski, Daniel J. Van Hook, Gerald C. O'Leary, Joshua W. Haines, and Lee M. Rossey. SARA: Survivable autonomic response architecture. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2001)* [DIS01].

[Mic] Microsoft. Baseline Security Analyzer. `http://www.microsoft.com/technet/security/tools/mbsahome.mspx`.

[MM01] D.J. Musliner and J.M. Maloney. Reasoning about timeliness for computer security reactions: CIRCA and AIA experiment 001. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2001)* [DIS01], pages 299–307.

[MPS+03] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The spread of the sapphire/slammer worm. Technical report, February 2003.

[Nes] Nessus. `http://www.nessus.org/`.

[Net] IBM Tivoli NetView. `http://www.tivoli.com/`.

[NRL03] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX 2003)*. IEEE Computer Society Press, April 2003.

[Ope] HP OpenView. `http://www.openview.hp.com`.

[PN97] Phillip A. Porras and Peter G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *20th National Information Systems Security Conference*, pages 353–365, October 1997.

[SDB+02] D. Sterne, K. Djahandari, R. Balupari, W. La Cholter, B. Babson, B. Wilson, P. Narasimhan, A. Purtell, D. Schnackenberg, and S. Linden. Active network based DDoS defense. In *Proceedings DARPA Active Networks Conference and Exposition (DANCE 2002)*, San Francisco, CA, USA, May 2002.

[SDS00] Dan Schnackenberg, Kelly Djahandari, and Dan Sterne. Infrastructure for intrusion detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2000)*. IEEE Computer Society Press, January 2000.

[SDW+01] Dan Sterne, Kelly Djahandari, Brett Wilson, Bill Babson, Dan Schnackenberg, Harley Holliday, and Travis Reid. Autonomic response to distributed denial of service attacks. In W. Lee, L. Mé, and A. Wespi, editors, *Recent Advances in Intrusion Detection — Proceedings of the 4th International Symposium (RAID 2001)*, volume 2212 of *Lecture Notes in Computer Science*, pages 134–149, Davis, CA, USA, October 2001. Springer-Verlag, Berlin Germany.

[SF00] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, August 2000. USENIX.

[SHS+01] Dan Schnackenberg, Harley Holliday, Randall Smith, Kelly Djahandari, and Dan Sterne. Cooperative intrusion traceback and response architecture (CITRA). In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2001)* [DIS01].

[Sno] Snort. `http://www.snort.org`.

[Som02] Anil Somayaji. *Operating System Stability and Security through Process Homeostasis*. PhD thesis, University of New Mexico, July 2002.

[Sym] Symantec. Intruder Alert. `http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=171&EID=0`.

[TK02] Thomas Toth and Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *18th Annual Computer Security Applications Conference* [ACM02].

[Vula] CERT Advisory CA-2002-17 Apache Web Server Chunk Handling Vulnerability. `http://www.cert.org/advisories/CA-2002-17.html`.

[Vulb] CERT Advisory CA-2003-20 Apache Web Server Chunk Handling Vulnerability. `http://www.cert.org/advisories/CA-2003-20.html`.

[VVZK02] Giovanni Vigna, Fredrik Valeur, Jingyu Zhou, and Richard A. Kemmerer. Composable tools for network discovery and security analysis. In *18th Annual Computer Security Applications Conference* [ACM02].

[WFFR01] Huaqing Wei, Deb Frinke, Olivia Frinke, and Chris Ritter. Cost-benefit analysis for network intrusion detection systems. In *CSI 28th Annual Computer Security Conference*, October 2001.

[Wil02] Matthew M Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *18th Annual Computer Security Applications Conference* [ACM02].

A Grammar of scripting language

Find following the grammar described of the scripting language defined in EBNF [ISO96]:

```
Program          := Stmt , { ";" Stmt } ;
Stmt             := Action | Assign | Undo ;
Action           := Cmd "on" HostGrp ;
Assign           := VarID ":=" ( Cmd | HostGrp | Action ) ;
Undo             := "undo" VarID ;

Cmd              := BasicCmd | CompoundCmd ;
CompoundCmd      := ["("] Cmd { ( "or" | "and" ) Cmd } [")"] ;
BasicCmd         := VarID | StopCmd | FilterCmd | AlertCmd ;
StopCmd          := "stop" [ ( "service" | "program" ) "=" Name ] ;
FilterCmd        := "filter" { HostGrp [ "as initiator" | "as responder" ] }
                    [ "responderPort =" PortParam ] [ "initPort =" PortParam
                    ] ;
AlertCmd         := "raise_alert_level" ;

HostGrp          := BasicHostGrp | CompoundHostGrp ;
BasicHostGrp     := VarID | Host | Net | TCS | Chokepoint | SourceCmd| "("
                    HostGrp ")";
CompoundHostGrp := ["("] HostGrp { ( "union" | "intersect" | "diff" ) HostGrp
                    }- [")"];
Host             := Hostname | IP;
Net              := "Intranet" | "Extranet" | IP"/"Num | IP"/"IPMask ;
TCS              := "select from" HostGrp "where" TCSCond ;
Chokepoint       := "chokepoints of" ["trusted" | "untrusted"] HostGrp ;

SourceCmd        := "`" Name "`"
TCSCond          := BasicTCSCond | CompoundTCSCond ;
BasicTCSCond     := (("service" | "program" | "os" ) "=" Name [VersionInfo] |
                    "capability" BasicCmd ;
CompoundTCSCond := "not" TCSCond | ["("] TCSCond { ( "and" | "or" ) TCSCond }-
                    [")"] ;

VersionInfo      := [ "VersionID" compOp ] "version" compOp "VersionID" ;
compOp           := "=" | "<" | ">" | ">=" | "<=" ;
IP               := { Num }-"."{ Num }-"."{ Num }-"."{ Num }- ;
IPMask           := IP;
Num              := { "Digit" }- ;
VarID            := Name ;
Hostname         := Name ;
PortParam        := Num "/"("TCP" | "UDP" | "RPC") ;
Protocol         := Name ;
```

```
# CERT advisory CA-2003-20.

 xp := select from intranet where
    os = windows version = xp;
 winserver := select from intranet where
    os = windows  version = server2003;
 nt := select from intranet where
    os = windows  version = nt4.0;
 win2k := select from intranet where
    os = windows  version = 2000;


 Vuln := xp union  Vuln2 union nt
          union win2k;
 Infected := ‘network_worm_scanner‘;
% Set of vulnerable and infected nodes resp.

% Choke connections from  infected nodes.
 filter Infected as initiator on
    (chokepoints of Infected);

% Now, protect vulnerable nodes.
 Remaining := Vuln;

% Stop the service where we can
 StopSet := select from Remaining
    where capability stop
    service = dcom-rpc;
 stop service = dcom-rpc on StopSet ;


 Remaining := Remaining diff StopSet;

% Stop the machine where we can
% The service host refers to the machine.
 HaltSet := select from Remaining
    where capability stop service = host;
 stop service = host on HaltSet ;


% update firewall rules on vulnerable nodes,
% if possible
 Remaining := Remaining diff HaltSet;
 FilterSet := select from Remaining  where
    capability filter responderPort = 135/TCP ;
 filter responderPort = 135/TCP
    FilterSet as responder on FilterSet;


% Filter connections on chokepoints
% for the remaining nodes
 Remaining := Remaining diff FilterSet;
 ChokeNodes :=  chokepoints of Remaining;
 filter port = 135/TCP
     Vuln as responder  on ChokeNodes;

% Raise alert level on vulnerable nodes.
  raise_alert_level on Vuln;

% Actually, Blaster may use other ports too.
% Thus, we need to do the above action for ports
% 135/UDP, 139/TCP, 139/UDP etc. also.
```

Figure 9: Responding to Blaster worm. CERT advisory CA-2003-20

B Example: W32/Blaster worm

Microsoft Blaster worm appeared in August 2003 and it uses a vulnerability in RPC DCOM module in Windows XP, Windows NT 4.0, Windows Server 2003 and Windows 2000 [Vulb]. About three hundred thousand machines were affected. It uses UDP/TCP 135 and 139 ports. The exploit allows the attacker to execute arbitrary code with full privileges. The response script shown in Figure 9 would have been one way to tackle with a blaster incident using our platform and scripting language.